Deep Learning Techniques in Data Augmentation and Neural Network Design

Joseph Ely Lemley

College of Engineering and Informatics
National University of Ireland, Galway

This dissertation is submitted for the degree of
Doctor of Philosophy

Supervisor: Prof. Peter Corcoran May 2020

“Pray, Mr. Babbage, if you put into the machine wrong figures, will the right answers come out? ”

~Charles Babbage - Passages from the Life of a Philosopher (1864)

Table of contents

List of figures **xvii**

List of tables **xix**

Nomenclature **xxi**

1 Introduction **1**

1.1 Introduction 1

1.1.1 Historical Perspective of this Work 2

1.2 Overview of Published Work 4

1.2.1 Contributions to Papers Published as Journal Articles 4

1.2.2 Selected Conference Papers 6

1.2.3 Other Published Work 7

1.2.4 List of Peer Reviewed First Author Publications (Starting with Journals) 8

1.2.5 List of Peer Reviewed Non-First Author Publications (Starting with Journals) 8

1.2.6 List of Filed Patents 9

1.2.7 Contribution Taxonomy 9

1.3 Organization of Remainder of Thesis 10

2 Smart Augmentation **15**

2.1 Research Questions 17

2.1.1 A First Research Question 17

2.1.2 How Does SA Compare with Traditional Augmentation? 18

2.1.3 Investigations into Smart Augmentation's Hyper-Parameters 19

2.1.4 SA on Multiple Datasets 21

2.1.5 Smart Augmentation and Overfitting 22

2.1.6 Summary of Findings from Research Questions 23

2.2 Research Impact 24

2.2.1 Learnable Data Augmentation 25
2.2.2 Alternative Learned Augmentation Techniques 26
2.2.3 Methods Based on GANs and Statistical Generative Techniques . . 27
2.2.4 Data Augmentation for Audio 28
2.2.5 Other Recent Augmentation Techniques 30
2.2.6 Discussion 31
2.3 A Discussion on Smart Augmentation and GANs 31
2.3.1 Differences 32
2.3.2 Similarities 33
2.3.3 GASA 33
2.4 Smart Augmentation and other Similar Techniques 34

3 Eye Gaze 37
3.1 Exploring Eye Gaze Estimation for DMS 38
3.2 Convolutional Neural Network Implementation for Eye-Gaze Estimation on Low-Quality Consumer Imaging Systems 39
3.2.1 Contributions of this Work 40
3.2.2 Impact of Random Resizing as Augmentation 41
3.2.3 Models Examined 41
3.2.4 Discussion and Overview of Major Findings 44
3.3 Extending the Idea to Augmented Environments 45

4 Transfer Learning of Temporal Information for Driver Action Classification And Semi Parallel Deep Neural Networks 49
4.1 Transfer Learning of Temporal Information for Driver Action Classification 49
4.1.1 Frame Based Methods and the Need for Temporal Information . . . 50
4.1.2 Data Used for Training and Testing the Temporal and Frame Based Networks 50
4.1.3 Research Questions 52
4.1.4 Augmentation 53
4.1.5 Experiments 53
4.1.6 Summary of Findings 55
4.2 Semi-Parallel Deep Neural Network (SPDNN) Hybrid Architecture, First Application on Depth from Monocular Camera 56
4.2.1 How to Construct a Semi-Parallel Deep Neural Network with Graph Contraction and Labeling 57

References **61**

Appendix A Deep Learning for Consumer Devices and Services **67**

Appendix B Smart Augmentation - Learning an Optimal Data Augmentation Strategy **87**

Appendix C Transfer Learning of Temporal Information for Driver Action Classification **101**

Appendix D Convolutional Neural Network Implementation for Eye-Gaze Estimation on Low-Quality Consumer Imaging Systems **109**

Appendix E Eye Tracking in Augmented Spaces: a Deep Learning Approach **121**

Appendix F Semi-Parallel Deep Neural Network (SPDNN) Hybrid Architecture, First Application on Depth from Monocular Camera **129**

Appendix G Contributions to Deep Learning Education **151**

G.1 Deep Learning For Consumer Devices and Services 152

G.2 Contributions to Deep Learning Education 152

G.2.1 Mobile Device Technology: A First DL Lab 152

G.2.2 The Edge AI Workshop at IEEE GEM 2018 154

G.2.3 Mobile Device Technology and ESAP labs 156

Appendix H Worksheet for MDT lab 7 (EE5116) on Deep Learning **159**

Appendix I GEM 2018 Workshop **163**

Appendix J Deep learning labs in 2018 **169**

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 80,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Joseph Ely Lemley
May 2020

Acknowledgements

To fully express my appreciation and to acknowledge every person who deserves to be listed here would force me to violate the 80,000 word maximum guideline for this thesis, and thus I have written as much as I think I will be able to get away with. First of all, I thank my supervisor, Peter Corcoran. I was fortunate to have such a good and supportive supervisor for my PhD studies. Any time I needed another GPU for experiments, wanted to test an idea with an embedded prototype or other equipment, he always made sure I had everything I needed. Professor Corcoran was the one who initially suggested that I apply to the Irish Research council for a PhD position here in Galway. I'll always be appreciative for the many opportunities he has given and continues to give me. I look forward to writing more papers and IP disclosures with you in 2020 and beyond. I'd like to thank my industry supervisor, Chris Dainty. It is rare to have the opportunity to interact with a scholar of your accomplishments and yet you always had time for me. I still have a lot to learn from you and am ever grateful for your support and suggestions throughout my PhD and in writing this thesis. I'm honoured to work on the Heliaus project with you and I look forward to all the collaborations the future holds. Alexandru Drimbarean, you were my manager at Fotonation for most of my PhD. You were one of my first contacts in Fotonation while I was still in Ellensburg working on my master's research. I deeply appreciate every discussion and idea we've exchanged and your faith and trust in me through all these years. You gave me my first DMS related projects, an eye gaze project that eventually became a paper and patent, and the head pose project. You mentored me as I became a manager at Fotonation and I will always be grateful for you. Petronel, you promised that you'd make sure I was never bored and you've kept that promise. There are very few PhD students who have had the opportunity to train neural networks that are implemented in real products and to lead talented teams of R&D engineers while working on their thesis. I'm grateful for your trust and confidence and for giving me so many interesting tasks to work on and all the cool cameras to play with. I look forward to our continued collaborations, we have so many cool things to make! The opportunities I have had here in Galway could never have happened without the amazing support and mentorship I received back home in Washington State so I would like to acknowledge but a small subset of those who deserve acknowledgement, who were responsible for me going to Ireland to work on a PhD. Boris Kovlechuck for not only being an excellent professor but also providing me with my first experience in working in a research environment with your Computational Intelligence and Visualization lab. I learned so much from you and will always look back on our time together with fondness. I hope we'll see eachother again, and maybe long enough to write another paper together. Razvan Andonie for teaching me practically everything I know about Artificial Intelligence, for introducing me to neural networks, genetic algorithms, and for all the research ideas over the years when I was working on my undergraduate degree and my master's degree. Speaking of which, thank you for administering the MS in Computational Science program at CWU and inviting me to attend as a student on a research assistantship. I greatly enjoy collaborating with you and hope that we have much more research and teaching to collaborate on in the future. James Schwing as the chair of the Computer Science Department at CWU for all the extra effort you made for every student. You enabled the beginning of my exposure to research and had faith in me when I had very little experience or knowledge. Sharon Martin for teaching me the skills necessary to succeed in a university setting and for believing in me and supporting my goal of getting a PhD even before I'd completed my first year of university studies. I truly don't think I would be here without your early guidance and patience. Thank you. I thank Steve Robeck for sticking by me and befriending me when all seemed hopeless in middle school. My parents, Enid and Joseph W Lemley for always believing in me and supporting my desire to learn, wherever it took me. The Apple II GS you bought me when I was 8 was the first computer I ever learned to program on. I don't know if I would have had such an aptitude for computer science without this early exposure to programming. It is impossible to express how much I appreciate you both. My loving wife, Kimberly Sowell, who has supported me throughout my masters and PhD, whose love, resourcefulness, and creativity have brought so much joy to my life. You've made sure I've had everything I've needed to focus on research during these times and you've never complained about the late nights I spend working. I will always be grateful for my colleagues and friends at Fotonation, NUIG and the C3 Imaging lab. We have had so many amazing experiences together. Shabab Bazrafkan, when I first arrived to the office in Galway, I told you that I really looked forward to collaborating with you on Deep learning projects. I had no idea that this would mean 4 papers, 2 patents, and more research projects than I can remember, or that we'd fly all over the world together and teach a workshop on Edge AI. Galway was much more fun with you here and I miss you. Come back and we'll open the tremoço stand – or at least I'll take your Salsa class. I'll have more time for that kind of thing with my thesis out of the way. Thank you for your friendship. Adrian Ungureanu, on my first day in Ireland, before I had met anyone else, you met me in Galway, a city I'd never been to in a country I'd never been to before, made sure I had everything I needed, and eventually became my roommate. Thank you for your friendship. Hossein Javidnia, for your friendship and support during my PhD and for tolerating me for the 3 years you sat beside me. Shejin Thavalengal, in a way you paved the way for all of us; you had already graduated by the time I arrived. Your help and friendship were appreciated. Also, this thesis was written using the LaTeX template you used for your thesis. It only took a couple changes to get it to work with overleaf. Thanks! Thank you Joe Desbonnet for your knowledge, support, and friendship. Thank you Ashkan Parsi, Aoife McDonagh, and Viktor Varkarakis for the friendship and for all the help in running the labs at NUIG for MDT and ESAP. Thank you Anuradha Kar, Tudor Nedelcu, Asma Aman, Faisal khan, and Timothy Cognard for all the lunchroom discussions and all the experiences we've shared together. Thank you Claudia Costache for always knowing what forms to file and how to navigate this whole process. And all of you for the for all the birthdays, parties, travels, discussions, and company. Your friendship means so much to me and I've learned so much from you. Alin and Diana, you were the first ML Engineers that I was a Manager for. We built so many cool things together! Thank you for all the late nights doing acquisitions and tuning networks. You took what I started and transformed it into the best single camera driver monitoring system available. And of course everyone at Fotonation/Xperi, from whom I've learned so much. You've all been so welcoming and wonderful. I will never be able to fully express how grateful I am for all of you. A person only has one chance to be a PhD student and I am very thankful to everyone I have met in this journey. And of course the Irish Research Council and Fotonation/Xperi for generously funding my PhD.

Abstract

In recent years, deep learning has revolutionized computer vision and has been applied to a range of problems where it often achieves accuracies equal to or greater than those obtainable by individual human experts. This research improves on the state-of-the-art by proposing, implementing, and testing new models, architectures, and training methods that are more efficient while maintaining or improving the accuracy of previous methods. Special attention is focused on improvements that facilitate the specific needs of resource-constrained devices such as smartphones, and embedded systems, and in cases where obtaining sufficient data is difficult. For this reason, the topic of data augmentation is a major theme of this work.

Due to the ever greater need for smarter embedded devices, my research has focused on novel network designs and data augmentation techniques for a wide range of diverse tasks, connected only by the need for more efficient architectures and more data – in many cases improving the accuracy over previous works in the process.

List of figures

2.1 High Level Overview of Smart Augmentation 16
2.2 Datasets Used to Evaluate Smart Augmentation 22
2.3 Training and Validation Losses for SA 23
2.4 Generative Adversarial Smart Augmentation 34

3.1 Eye Gaze Networks Included in Xperi Driver Monitoring System at CVPR 2019 39
3.2 Network Architecture for Eye Gaze Networks 41
3.3 Data Flow for Eye Gaze Networks 42
3.4 Cases when Traditional Methods Outperfrom Deep Learning 46

4.1 Transfer Learning with C3D 54
4.2 Labeling and Contraction of SPDNN 57
4.3 Illustration of 8 Networks Before Graph Contraction with SPDNN 58
4.4 Illustration of Networks After Graph Contraction 59

G.1 Example Edge-AI Kit 155

List of tables

1.1 Contributions to the Main Ideas of Selected Papers 10
1.2 Contributions to the Rationalization and Context of Selected Papers 11
1.3 Contributions to the Experiments and Implementation of Selected Papers . . 11
1.4 Contributions to the Manuscript Preparation of Selected Papers 12
1.5 Contributions to the Background of Selected Papers 12

3.1 Result of Distance Simulation and Augmentation Experiments 40
3.2 Frames Per Second of CNN on Commodity Hardware 43
3.3 Comparison of Proposed Model and Other Published Works on the MPII-Gaze Database 44

4.1 Top Performing Approaches (>= 30%) 55
4.2 The Best Results on the Validation Set After Reaching 100% on the Training Set 55

Nomenclature

Acronyms / Abbreviations

ADAM	Adaptive Moment Estimation
AE	AutoEncoder
ANN	Artificial Neural Networks
CNN	Convolutional Neural Networks
DMS	Driver Monitoring System
DSP	Digital Signal Processor
ELU	Exponential Linear Unit
GAN	Generative Adversarial Network
GD	Gradient Descent
GPU	Graphics Processing Unit
MAC	Multiply-Accumulate Operation
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
SA	Smart Augmentation
SGD	Stochastic Gradient Descent

Chapter 1

Introduction

1.1 Introduction

From the “no free lunch” theorems [1] we know that there is no one machine learning method that will perform better than any other for arbitrary data. This means that even the most sophisticated convolutional neural network may perform no better, or even worse than linear regression for some problems. For this reason, it is necessary to develop network architectures and training methods for a specific task or subset of tasks.

Fortunately, various universal approximation theorems [2] also tell us that deep neural networks are able to approximate almost any function the computer vision researcher might be interested in solving, which means that, as long as we can express our problem as a “black box” function with an output and an input, a neural network can approximate, or model the desired output from that input. Although this theorem allows for optimism that nearly any given task can be learned by a neural network, it provides no mechanism to do so and no guarantee that such a network can even be learned with conventional training methods such as backpropagation [3] with Stochastic Gradient Descent (SGD).

To achieve the goal of improving upon deep learning systems for computer vision, two aspects of such systems are addressed. These are network architecture and training. Training methods and network architectures are often, but not always, interrelated. For example, our work on Smart Augmentation addresses the common problem of neural networks not having enough data to be properly trained; it does this by inventing a new architecture that learns to generate data at the same time it learns to accomplish the task it was trained on. Likewise, our work on transfer learning for driver behavior monitoring demonstrates the counter-intuitive fact that primitive temporal features from a neural network trained on a task such as sports movies can be used to significantly improve results on the driver behavior monitoring task.

In this case, the improvements were entirely from training methodology although the choice of architecture (a model that could utilize temporal information) was also a factor.

1.1.1 Historical Perspective of this Work

A brief history of AI with a focus on Neural networks is provided in this subsection with the goal of framing this work within its historical context.

One of the most important limitations of neural networks is that there are no suitable explanations for why they work as well as they do. This lack of explainability has led to several periods of stagnation known as AI winters. During the AI winters, funding for research on neural networks almost vanished and the spending of public money for AI work was discontinued at many institutions.

Nature is a seemingly endless source of inspiration for the development of learning systems. The fact that intelligence exists in nature was always a simple counter example for those skeptical about the possibility of artificial intelligence, but is insufficient to demonstrate feasibility. The fact that it exists in many forms and to varying degrees provides hope that we can develop methods to approximate or emulate some forms of intelligence.

This sentiment was also echoed by one of the pioneers of computer science and artificial intelligence: Alan Turing. Turing believed that the human brain could be modeled as an unorganized device with random weights which would be updated with reinforcement learning. [4]

However, for funding agencies requiring measurable improvement and clear objectives and results, these aspirations were largely unconvincing. Top Neural networks and machine learning conferences such as Conference on Neural Information Processing Systems frequently rejected papers on deep learning and neural networks simply because the topic was undesirable even in the mid 2000s.

Artificial Neural networks operate on highly simplified ideas of an incomplete knowledge of how the human brain works. It was therefore easy to dismiss progress on neural network research until 2012 when progress on deep learning research by a small number of academics, working with efficient new hardware designed for gaming systems, demonstrated that the performance of these networks on real tasks not only outperformed other machine learning models but also outperformed human operators at the same set of tasks [5].

The Darthmoth proposal, which was written in the 1950s, was written under the assumption that "Every aspect of learning or other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it" [6].

From a distance, the status of AI research today is not so different from 1955 when the authors of the Darthmoth proposal wrote:

> *Neuron Nets*
>
> How can a set of (hypothetical) neurons be arranged so as to form concepts. Considerable theoretical and experimental work has been done on this problem by Uttley, Rashevsky and his group, Farley and Clark, Pitts and McCulloch, Minsky, Rochester and Holland, and others. Partial results have been obtained but the problem needs more theoretical work [6]."

This lack of theoretical understanding and the dominance of empirical results still characterize this field and have been some of the main criticisms of the neural network approach to AI to this day. Today we simply have better empirical results and slightly better algorithms than in the 1950s but few core concepts have changed. In fact, the majority of what we call deep learning today would be recognizable to any student of neural networks in the mid 1990s with a few exceptions:

- The adoption of the Rectified Linear Unit (RELU) activation function (sigmoid and tanh were common at the time).
- The ability to train deeper networks using a variety of techniques that help avoid vanishing gradients (ie resnet blocks [7] and inception modules [8]).
- The popularity of convolutional layers in neural networks.
- The large datasets we use to train and test them.
- Powerful GPU based hardware with which to train them.
- Large amounts of very fast memory.

Convolutional Neural Networks were popularized by Lecun et al. [9], who used them successfully for handwritten digit classification. These networks are inspired by the organization of the visual cortex and allow spatial information to be more efficiently learned. CNNs can be used on input with any number of dimensions, but due to their success in pictures, are most popularly implemented for 2D input plus color channels. Other popular types of CNNs include 1D CNNs, which are commonly used for time series, and 3D CNNs, which can be used for volumetric data or time series data where the third dimension represents either spatial frames or temporal frames [5].

In recent times deep neural networks have become essential to a wide range of tasks but the most common implementations of these networks require expensive hardware that will not fit in todays consumer devices or when they are deployed on such devices often quickly drain the device battery.

Furthermore, gathering data for these networks is often time consuming and prone to error, leading to poor training.

This thesis builds on previous empirical work in adapting neural networks to consumer devices, focusing on dataset augmentation and deep learning techniques primarily for computer vision applications in consumer electronics.

1.2 Overview of Published Work

In this section the contributions of this thesis are summarized. As the presented research was performed in collaboration with others, it is important to provide information about which contributions are those of the author versus those of coauthors. These contributions start with the three journal papers, followed by conference papers that were subject to peer review, then other published work, and finally unpublished work.

1.2.1 Contributions to Papers Published as Journal Articles

Smart Augmentation

Data augmentation involves modifying data from its original form. It is done with the hope that a different perspective or transformation of the data will help the network to learn a better representation of the desired task. For example a convolutional neural network may not understand that an object that has been rotated is the same as one that is not rotated, and therefore augmentation with rotated images is common. Augmentation is a critical component of training a neural network, but it is important to carefully consider the types of augmentation one does beforehand. This can be illustrated by considering the digit classification task. If digits are rotated more than 90 degrees, 9s will be indistinguishable from 6s. Although some rotation invariance may be desired for digit recognition, the orientation of digit objects is necessary to understand them properly and therefore arbitrary rotations result in confusion and will prevent proper training of a neural network. One way of thinking about augmentation is to consider it a form of regularization that operates on the input space.

The idea of learnable augmentation derives from the recent history of computer vision algorithms and the fact that many of them are not used in deep learning because the network learns to create similar filters during training if they are needed [10]. In many fields, these traditional filters have been entirely replaced by neural networks.

The fundamental research question posed by learnable augmentation is: Can a neural network learn to perform the labor intensive process of data augmentation? Can a network or a collection of networks learn not just how to perform a task, but also how to create new data that will help it learn that task?

This question had not been studied before the fall of 2016 when the research presented in this thesis started. The research showed that the answer to this question is: yes, neural networks can learn to modify or augment data during training in such a way as to reduce the error on a dataset.

Shortly after these results were published, a number of other researchers built on this work and performed a range of additional experiments. A recent survey of these works and other related works are provided in section 2.2.

My contributions were the initial idea of a network that could learn to perform data augmentation, planning and performing the experiments, and writing most of the paper. I also adapted the code to new datasets and situations. Shabab Bazrafkan made very important contributions to further refining the idea, writing the core neural network code, designing figures, and composing some parts of the paper.

Smart augmentation was published in [11] as a journal paper. A conference paper based on the original paper was also published [12] and results were further disseminated at the March 27 Fourth Edinburgh Deep Learning Workshop 2017, and at the Embedded Vision Summit, Santa Clara California in 2018 [13]. The slides from the embeded vision summit can be accessed at https://aran.library.nuigalway.ie/handle/10379/7480.

The primary work on Smart Augmentation is included in Appendix B and a detailed overview of Smart Augmentation, its relation to other techniques, and the core theme of this work are given in chapter 2.

Eye Gaze Estimation for DMS

The eye gaze estimation project started as an industry project at FotoNation Ireland. There was a need for an eye gaze solution that would run on an embeded device in a vehicle in real time with strict accuracy and performance, measured in multiply accumulate operations (MACs) requirements.

As a feasibility study, we developed a parallel academic project on the same topic, but using public (visible spectrum) data and methods. The academic portion of this work was published in Transactions on Consumer Electronics [14]. This method was also further developed, trained on NIR data, and integrated into an emended DMS solution. A patent application was filed on these further innovations, but due to the commercially sensitive nature of the know-how and datasets, no academic research papers were published on the final (NIR) implementation of this research.

My contributions were in the networks and methodology, running and designing the experiments, writing the code except for parts of the data loader, and writing all of the paper except for the related work section. Anuradha Kar wrote the related work section and served as an expert advisor for any questions related to norms in eye gaze research. Alexandru Drimbarean performed a managerial role, ensuring that deadlines were met and MAC requirements were not exceeded for the industry portion of the project. He also proposed the project.

The journal paper that came out of this work is included in Appendix D, while a conference paper describing an extension of the same network to augmented spaces is in Appendix E.

The core contributions of both papers, with emphasis on the journal paper, are discussed in detail in chapter 3.

Semi Parallel Deep Neural Networks

Semi Parallel Deep Neural Networks (SPDNN) are a new algorithm to merge different types of neural networks together in such a way as to benefit from each without duplication in computation.

The contribution was to take an existing technique for merging networks and to generalize it with a graph based approach.

It was discovered that if each neural network layer were uniquely labeled and then represented as a graph, that it would then be possible to use graph contraction to come up with a single network that had many of the advantages of others. The results of this research was published in [15] and a patent application was also submitted with both Shabab Bazrafkan and myself as co-inventors.

The journal paper on SPDNN for which I am an a coauthor is included in appendix F and a detailed discussion of the graph based algorithm that I helped develop is described in section 4.2.

1.2.2 Selected Conference Papers

Transfer Learning of Temporal Information for Driver Action Classification

In Transfer Learning of Temporal Information for Driver Action Classification [16] a set of frames depicting driver behaviors was analyzed with a CNN. A variety of network types were used such as regular 2D CNNs, 3D CNNs, Recurrent Neural Networks(RNNS) and various hybrids of these approaches. A transfer learning technique based on a 3D CNN that had been pre-trained on YouTube sports videos ranked the highest. My contributions included the programming, designing the experiments, organizing the data, and writing all of the paper except for the section on the cone shape of information flow and related figures, which my coauthor Shabab Bazrafkan contributed to. This conference paper is included in Appendix C and a detailed description of the approach is included in chapter 4.1.

Eye Tracking in Augmented Spaces: A Deep Learning Approach

In Eye Tracking in Augmented Spaces: A Deep Learning Approach [17], two forms of eye gaze estimation with differing needs were analyzed: Eye gaze estimation in AR/VR contexts where head-mounted eye-facing cameras are available, and cases where the camera is at a distance. The study concluded that current CNNs are able to outperform traditional methods for the latter case but not the former, as high resolution images of the eyes allow for very precise angle approximation techniques.

I performed all experiments and programming as well as writing the sections of the paper related to experiments, results, and conclusions. The introduction was written by both Anuradha Kar and myself, and the literature review section on eye gaze methods was written by Anuradha Kar. This conference paper is included in Appendix E and a detailed discussion of it is included in section 3.3.

1.2.3 Other Published Work

In Deep Learning for Consumer Devices and Services: Pushing the limits for machine learning, artificial intelligence, and computer vision, a discussion of deep learning focused on consumer devices is given [5]. Shabab Bazrafkan wrote section 2, Convolutional Neural Networks, while I wrote the rest. Although this paper did not propose a new algorithm or methodology, it was written in an accessible way and helped to introduce new researchers to the subject. The paper provides an introduction to neural networks, starting with historical usage and ending with the latest architectures and tools. This paper is currently my second most cited publication and won a "Best Paper" award from the IEEE consumer electronics society in 2018. Four follow-up papers have now been published, drawing from other works. Such articles promote and explain CNN techniques to a broader audience. Further details on this series of publications can be found in Appendix G.1.

1.2.4 List of Peer Reviewed First Author Publications (Starting with Journals)

1. J. Lemley, S. Bazrafkan, and P. Corcoran, "Smart Augmentation-Learning an Optimal Data Augmentation Strategy," *IEEE Access,* vol. 5, pp. 5858-5869, 2017.[11]

2. J. Lemley, A. Kar, A. Drimbarean, and P. Corcoran, "Convolutional Neural Network Implementation for Eye-Gaze Estimation on Low-Quality Consumer Imaging Systems". *IEEE Transactions on Consumer Electronics*, vol. 65, no. 2, pp. 179-187, 2019 [14].

3. J. Lemley, S. Bazrafkan, and P. Corcoran, "Deep learning for consumer devices and services: Pushing the limits for machine learning, artificial intelligence, and computer vision." *IEEE Consumer Electronics Magazine*, vol. 6, no. 2, pp. 48-56,2017. [5]

4. J. Lemley, S. Bazrafkan, and P. Corcoran. "Transfer Learning of Temporal Information for Driver Action Classification," in *Proceedings of the 28th Modern Artificial Intelligence and Cognitive Science Conference 2017,* vol. 1964, 2017, pp. 123-128. [16]

5. J. Lemley, S. Bazrafkan, and P. Corcoran, "Learning Data Augmentation for Consumer Devices and Services," in *2018 IEEE International Conference on Consumer Electronics (ICCE).* IEEE, 2018, pp. 1-3. [12]

6. J. Lemley, A. Kar, and P. Corcoran. "Eye Tracking in Augmented Spaces: A Deep Learning Approach" in *2018 IEEE Games, Entertainment, Media Conference (GEM),* IEEE, 2018, pp. 1-6 [17].

1.2.5 List of Peer Reviewed Non-First Author Publications (Starting with Journals)

7. S. Bazrafkan, H. Javidnia, J. Lemley and P. Corcoran, "Semiparallel deep neural network hybrid architecture: first application on depth from monocular camera," *Journal of Electronic Imaging,* vol. 27, no. 4, pp. 0430341 1-19, 2018 [15].

8. P. Corcoran, J. Lemley, C. Costache and V. Varkarakis. "Deep Learning for Consumer Devices & Services 2 - AI gets Embedded at the Edge", *IEEE Consumer Electronics Magazine,* vol. 8, no. 5, pp. 10-19, 2019.

9. A. McDonagh, J. Lemley, R. Cassidy and P. Corcoran. "Synthesizing Game Audio Using Deep Neural Networks," *2018 IEEE Games, Entertainment, Media Conference (GEM)*, IEEE, 2018, pp. 1-6 [18].

1.2.6 List of Filed Patents

1. S. Bazrafkan and J. Lemley, "Method for Synthesizing a Neural Network," US Patent App. 15/413,283. [19]

2. S. Bazrafkan and J. Lemley, "Method of Training a Neural Network," US Patent App. 15/413,312. [20]

3. L. Dutu, M. Dumitru-Guzu, S. Mathe, J. Lemley, "Simultaneous Gaze and Eyelid Opening Estimation from Both Eyes". US Patent App. 16/005,610.

4. J. Lemley, A. McDonaugh, R. Cassidy, "Hybrid Audio Synthesis Using Neural Networks". PCT/US2019/040,739

1.2.7 Contribution Taxonomy

As this publication based thesis includes work that was done in collaboration with others, this section provides an overview of the contributions of each author to each major paper. The 6 tables in this section are based on a generalization of the Contributor Role Taxonomy (CRediT) introduced by Brand et al [21]. The CRediT approach has been adopted by journals in several fields to specify the contributions of individual authors.

In the CRediT Taxonomy each author's contributions are measured as a percentage point on 14 roles. These are: Conceptualization, Data curation, Formal Analysis, Funding acquisition, Investigation, Methodology, Project administration, Resources, Software, Supervision, Validation, Visualization, Writing – original draft, Writing – review & editing

In this thesis I adopt a more concise generalization of this taxonomy which encapsulates the major criteria, specifically:

1. Main Idea - Involving Conceptualization.

2. Experiments and Implementation, which includes Methodology, validation, data curation, formal analysis and software development.

3. Rationalization and Context, which includes investigation and formalization.

4. Manuscript Preparation which includes all aspects of writing manuscript preperation including Writing – original draft, Writing – review & editing, and Visualization except those specified in the next critera.

5. Background - Includes work done to place the research efforts in a wider context of literature in a given field, this may include some aspects of writing (literature reviews) and informs aspects of project administration and supervision and ensuring that methodology used is typical of that used in the area of publication.

This generalization has the weakness that it ignores most aspects of funding, project administration, resources or supervision but otherwise encapsulates the main points that would determine primary authorship. In cases where an author's contribution is missing, their contribution was solely in these ignored categories. The two coauthors that this applies to are Peter Corcoran, my supervisor, and Alexandru Drimbarean, my direct manager at Fotonation/Xperi. Tables 1.1, 1.2, 1.3, 1.4, and 1.5 list author contributions according to the above 5 criteria. Authors are listed by initial where JL means Joseph Lemley, SB means Shabab Bazrafkan, AK means Anuradha Kar, PC means Peter Corcoran, and HJ means Hossein Javidnia. Contribution percent is listed at a resolution of 10%. Further information on contributions and motivations are given in section 1.2 and in the relevant chapters.

Table 1.1 Contributions to the Main Ideas of Selected Papers

Paper	*Contribution Percent*
Smart Augmentation: Learning an Optimal Data Augmentation Strategy [11]	JL 100%
Convolutional Neural Network Implementation for Eye-Gaze Estimation on Low-Quality Consumer Imaging Systems [14]	JL 100%
Eye tracking in augmented spaces: A deep learning approach [17]	JL 100%
Semiparallel deep neural network hybrid architecture: first application on depth from monocular camera [15]	SB 50% HJ 50%
Deep Learning for Consumer Devices and Services: Pushing the limits for machine learning, artificial intelligence, and computer vision. [5]	JL 60% SB 40%
Transfer Learning of Temporal Information for Driver Action Classification. [16]	JL 100%

1.3 Organization of Remainder of Thesis

The remaining chapters contain a summary of motivations and details about the three qualifying journal articles as well as additional supporting work, published and unpub-

Table 1.2 Contributions to the Rationalization and Context of selected papers

Paper	*Contribution Percent*
Smart Augmentation: Learning an Optimal Data Augmentation Strategy [11]	JL 50% SB 50%
Convolutional Neural Network Implementation for Eye-Gaze Estimation on Low-Quality Consumer Imaging Systems [14]	JL 100%
Eye tracking in augmented spaces: A deep learning approach [17]	JL 100%
Semiparallel deep neural network hybrid architecture: first application on depth from monocular camera [15]	SB 50% HJ 30% JL 20%
Deep Learning for Consumer Devices and Services: Pushing the limits for machine learning, artificial intelligence, and computer vision. [5]	JL 100%
Transfer Learning of Temporal Information for Driver Action Classification. [16]	JL 80% SB 20%

Table 1.3 Contributions to the Experiments and Implementation of Selected Papers

Paper	*Contribution Percent*
Smart Augmentation: Learning an Optimal Data Augmentation Strategy [11]	JL 90% SB 10%
Convolutional Neural Network Implementation for Eye-Gaze Estimation on Low-Quality Consumer Imaging Systems [14]	JL 100%
Eye tracking in augmented spaces: A deep learning approach [17]	JL 100%
Semiparallel deep neural network hybrid architecture: first application on depth from monocular camera [15]	SB 50% HJ 50%
Deep Learning for Consumer Devices and Services: Pushing the limits for machine learning, artificial intelligence, and computer vision. [5]	N/A (Introductory Review Paper)
Transfer Learning of Temporal Information for Driver Action Classification [16]	JL 100%

Table 1.4 Contributions to the Manuscript Preparation of Selected Papers

Paper	*Contribution Percent*
Smart Augmentation: Learning an Optimal Data Augmentation Strategy [11]	JL 90% SB 10%
Convolutional Neural Network Implementation for Eye-Gaze Estimation on Low-Quality Consumer Imaging Systems [14]	JL 90% AK 10%
Eye tracking in augmented spaces: A deep learning approach [17]	JL 90% AK 10%
Semiparallel deep neural network hybrid architecture: first application on depth from monocular camera [15]	SB 45% HJ 45% JL 10%
Deep Learning for Consumer Devices and Services: Pushing the limits for machine learning, artificial intelligence, and computer vision. [5]	JL 70% SB 20% PC 10%
Transfer Learning of Temporal Information for Driver Action Classification [16]	JL 90% SB 10%

Table 1.5 Contributions to the Background of Selected Papers

Paper	*Contribution Percent*
Smart Augmentation: Learning an Optimal Data Augmentation Strategy [11]	JL 90% SB 10%
Convolutional Neural Network Implementation for Eye-Gaze Estimation on Low-Quality Consumer Imaging Systems [14]	JL 50% AK 50%
Eye tracking in augmented spaces: A deep learning approach [17]	JL 50% AK 50%
Semiparallel deep neural network hybrid architecture: first application on depth from monocular camera [15]	SB 45% HJ 45% JL 10%
Deep Learning for Consumer Devices and Services: Pushing the limits for machine learning, artificial intelligence, and computer vision. [5]	JL 100%
Transfer Learning of Temporal Information for Driver Action Classification [16]	JL 90% SB 10%

lished. As the attached articles (see appendix) contain extensive literature reviews and methodological details, neither a literature review nor a methodology chapter is included. The source code to duplicate the experiments described in this thesis is available at https://github.com/joelemley/PhD.

Chapter 2 contains a comprehensive overview of the smart augmentation technique and its research impact and comparison with related generative and meta learning techniques.

Chapter 3 contains an overview of the work done on eye gaze estimation, including details that were omitted in the attached journal article due to journal policies (such as page limits and the anti-commercialism policy).

Chapter 4 contains a summary of other published and unpublished work relating to the core theme of this thesis. This includes my work on driver monitoring using transfer learning, my contributions to SPDNN.

An overview of the 4 articles I have published with Consumer Electronics Magazine on deep learning, edge AI, data augmentation and contributions to Deep Learning education are included in appendix G.

Chapter 2

Smart Augmentation

Data augmentation involves modifying data from its original form. It is done in the expectation that a different perspective or transformation of the data will help the network to learn a better representation of the desired task.

For example a convolutional neural network may not understand that an object that has been rotated is the same as one that is not rotated, and therefore augmentation with rotated images is common. For illustration, an image of a face may not be recognized as a face if a person turns slightly and the network has not been trained with faces in a variety of head orientations.

Augmentation is a critical component of training a neural network, but it is important to carefully consider the types of augmentation one does beforehand and match these carefully to the task at hand. This can be illustrated by considering the digit classification task. If digits are rotated more than 90 degrees, 9s will be indistinguishable from 6s. Although some rotation invariance may be desired for digit recognition, the primary orientation of digit objects is essential to understand them properly and therefore arbitrary rotations result in confusion and will prevent proper training of a neural network.

The primary work on Smart Augmentation is included in Appendix B and a detailed overview of Smart Augmentation, its relation to other techniques and the core theme of this work are given in the remainder of this chapter.

Relating to the central theme of data augmentation for consumer devices, smart augmentation seeks to address the problem of limited and incomplete data. Inspired by the recent history of computer vision where features that were previously "hand crafted" began to be replaced by learned features, it was desired to explore whether the same progress could be made with augmentation as the objective in addition to classification.

The idea of Smart Augmentation is to investigate whether relevant improvements in the design of CNNs can be achieved by learning the data augmentation task. Specifically, Smart

Augmentation is a method of creating a network that learns the augmentation task at the same time as a classifier network learns to classify input. The augmenter network is called Network A and the network that performs a given task is called Network B. A combined loss function incorporates α and β coefficients to the losses of both network A and network B - described in detail in the relevant appendix.

Traditional data augmentation is a very manual process which can involve quite a bit of guesswork and expertise. For some problems/datasets the choice of augmentation strategy can be more important than the architecture of the network [10]. This led to the desire to investigate the ability to learn the augmentation task during training.

Smart augmentation works by using 2 networks:

- The network that learns the desired task.

- The network that learns to perform augmentations for the network that learns the desired task.

The former is called network B and the latter network A.

Network A selects, at random, 2 or more images from the same class

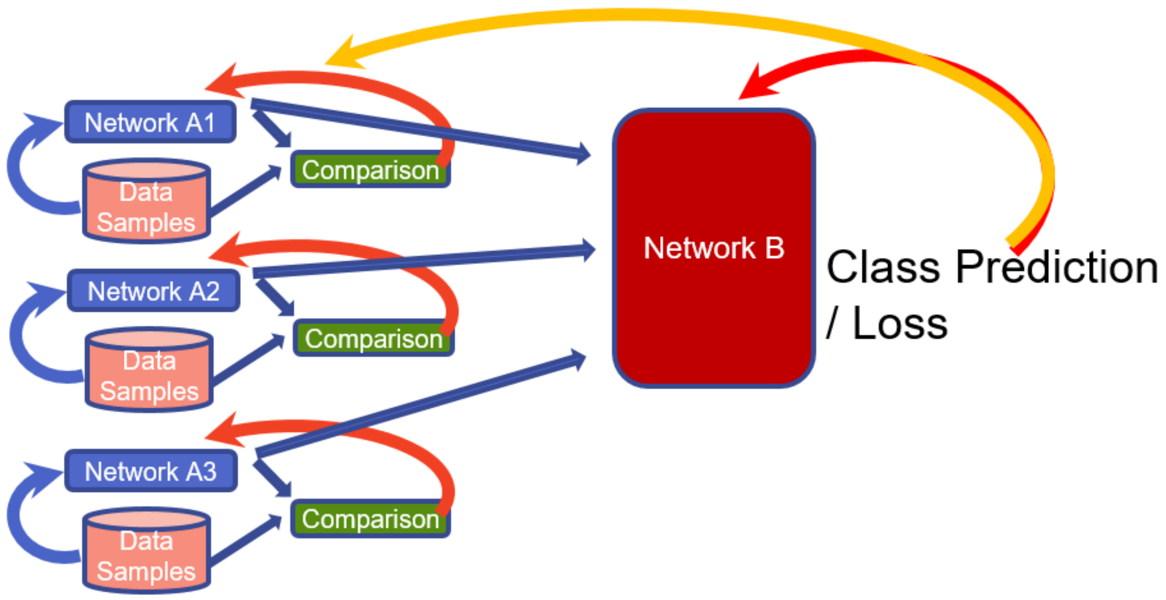


Fig. 2.1 This figure shows a high level overview of smart augmentation for a 3 class problem. To add classes, simply increase the number of network As.

2.1 Research Questions

Smart Augmentation (SA) started with one primary research question: "Can a neural network learn to perform the augmentation task from the data alone?"

After discovering that neural networks were indeed capable of learning to perform some augmentations, a series of research questions were designed to investigate the performance of the technique.

The experimental and methodological details of all the experiments used to answer the questions in this section are included in the main paper on Smart Augmentation in appendix B. The various network configurations can be found in table 1 of the appended paper, while the results are included in table II for experiments related to face datasets and table III for experiments related to the MIT places dataset.

2.1.1 A First Research Question

The first question was: "Is there any difference in accuracy between using SA and not using it? (Is SA effective?)". Although this is useful as a guiding concept, it is too vague to form a useful testable hypothesis. Thus a more precise question would be evaluated instead: Can a meta-learning approach such as the proposed SA technique increase accuracy for at least one task when evaluated on at least one dataset of interest? To answer this later question one must simply select a task and a dataset and attempt to train it using the SA methodology. For simplicity, a binary classification task was chosen: Perceived gender recognition from frontal face images and for the dataset I chose ARFACES, a highly constrained dataset of face portraits. Thus the hypothesis became "Will at least one specific implementation of an SA-like meta-learning approach increase gender recognition accuracy when trained and evaluated on ARFACES using a standard, subject-exclusive training, validation, and testing split?".

To examine this question, 16 experiments were performed, summarized in the following subsections. The full details of these experiments are included in table 1 of the main Smart Augmentation paper (Appendix B), while the results are included in table II and in the corresponding experiments, results, and methodology sections. These experiments demonstrated that the answer to this question was "yes" with increases in accuracy ranging from 90% to 95% depending on the parameters used compared with a baseline model that had 88% accuracy when using the proposed technique.

2.1.2 How Does SA Compare with Traditional Augmentation?

Upon demonstrating that SA was able to increase accuracy on a specific dataset was sufficient to validate the main idea in a very limited case, for real use cases it is helpful to know how it compares to the use of traditional augmentation. After all, if traditional augmentation is just as good, why would one bother with the added complexity of Smart Augmentation?

It would, of course, be impossible to compare SA with every possible traditional augmentation (as the number of possible variations are infinite even in the case of simple rotation). Therefore the first task was to select a representative set of traditional augmentations to compare with. The selected examples were: flipping, Gaussian blur, and rotation (-5,2,0,2,5 degrees with the axis of rotation at the center of the image). A dataset was created from ARFACES with every combination of these augmentations to form a new dataset called db1a.

When the baseline network, without smart augmentation, was trained on this dataset, the resulting accuracy increased to 89%, which is still less than the worst performing SA, indicating that, at least in this one case, the use of smart augmentation was better than the chosen traditional augmentations. However, no claim can be made for every traditional augmentation. It is possible, even likely with sufficient effort and time, that some combination of traditional augmentations could be found that would generate better results. However, the benefit of SA is that it replaces the tedious nature of trying many different manual augmentations. It does this by leveraging the ability of a neural network to learn an optimal augmentation strategy.

What Happens When Smart Augmentation is Trained on Traditionally Augmented Data?

The previous research question examined the use of traditional augmentation compared to a network that was trained with only SA but what would happen if the two approaches were combined? In other words, if the best approach from the experiments designed to answer the question posited in section 2.1.1 was trained on the traditionally augmented data, would the accuracy increase or decrease on the same test set? To answer this question, the same db1a, containing traditionally augmented images from ARFACES was used as training data for two networks trained with SA. The accuracy achieved with this approach was 95.67% which was a similar level of accuracy as when no traditional augmentation was used for this configuration.

2.1.3 Investigations into Smart Augmentation's Hyper-Parameters

The SA technique introduces new hyper-parameters and choices into the training pipeline, these include the α and β coefficients, the number of input channels, whether to use a separate network A for each class, or to use just one network A. In this subsection, research questions designed to investigate these hyper-parameters are described.

Why Does The Number of Input Channels Need to be At Least Two?

The smart augmentation technique uses randomly selected images from the same class placed in at least 2 channels as input. An obvious first question is: Is it really necessary to use at least 2 channels? What would happen if only 1 were used? Although it seemed pivotal to the main idea of smart augmentation, it was important to justify this requirement experimentally.

To investigate the need for this condition I repeated the experiment from section 2.1.1 but only supplied one image with all other parameters being the same as the best configuration previously found for SA on the ARFACES dataset. This resulted in a reported accuracy of 86.99%, which is worse than when smart augmentation was not used at all and seems to illustrate the importance of this rule.

This was true when one network A was used for each class, but surprisingly it did not have much of an impact when only one network A was used for both classes. In the case of only a single network A and a single input channel, the resulting accuracy was a 92.77% which was better than when no smart augmentation was used, but still less than the best single channel approach (95.38%).

It therefore seems prudent to use at least two input channels, especially when two network As are used.

The Number of Input Channels

Previously in this thesis, the requirement of using at least two input channels was examined. A natural question to ask is: Does varying the number of channels over 2 have an impact on network performance? If so, is there an optimal number of channels that can be found?

Experiments investigating this issue were performed for the ARFACES dataset, with the number of inputs set to vary between 2 and 8 for both single network A and one network A per class approaches.

These experiments indicated wide differences in performance resulting from selection of the number of inputs with as much as a 5% difference in accuracy depending on the number of inputs used. Unfortunately no linear correlation could be found between the number of inputs and the accuracy, and thus it is important to leave this parameter as a choice for the user. Investigations on the possibility of learning this parameter for a specific dataset are left for future research, as the experimental effort involved would be extensive and solutions to learn the parameter from the input are non-obvious.

How Many Network As are Needed?

There are two approaches studied for Smart augmentation. One approach uses a single "network A", which tries to learn to generate sample images for all the classes in the network. The second uses one network A for each class.

While using one network for each class makes some intuitive sense, it also increases training time and causes scalability concerns, increasing excess training time to be twice as long for a 2 class problem is reasonable, but for a 1000 class problem (such as ImageNet) it would make use of the method infeasible. Thus the suggestion that one network A is used for each class requires evidence of an increase in accuracy to justify the computational cost.

If the networks with just 1 and 0 input channels are excluded, there is an average (mean) increase in accuracy from 92.94 to 93.19 when multiple network As are used, with the median accuracy going from 92.49 to 93.35. This indicates that the use of multiple network As provides some benefit across datasets and parameters studied.

Does Altering the α and β Parameters Change the Results?

The SA technique includes two coefficients which together control the portion of the loss total that comes from network B (the classifier) against the portion that comes from the network As. The coefficient to the loss from network A is called α and the coefficient to the loss from network B is called β.

While running the experiments, alpha and beta values that seemed to work well were empirically decided, but these decisions were made prior to any final results while investigating network convergence during training. It occurred to me that an experiment should be designed to document the impact that the alpha and beta parameters have on network performance.

Therefore an experiment identical to that from section 2.1.1, except using the airport and abbey classes of the MIT places dataset instead of the faces dataset, was performed.

Four experiments containing two variations of alpha and beta (0.3,0.7) vs (0.7,0.3) were performed and allowed to run until the end. As expected, varying the loss coefficients had an impact on accuracy with ($\alpha: 0.7, \beta: 0.3$) reaching 99% accuracy, whereas ($\alpha: 0.3, \beta: 0.7$) obtained 97.87 for a network with two network As and 98.75 for the approach with just one network A.

2.1.4 SA on Multiple Datasets

The next research question in the Smart augmentation paper was: If smart augmentation is effective, is it effective across a variety of different datasets?, and if so, how does this vary as the datasets are increasingly unconstrained? This question is important because it is common to find methods that work well on highly constrained data but work poorly as the data becomes increasingly unconstrained. A popular example of this is the use of Eigenfaces [22] for face recognition. In that approach, the authors assume at the training stage that faces will be provided in a frontal format with an approximately similar size (i.e. mugshot) and that Principle Component Analysis (PCA) will determine the 2D features using 2D eigenvalues from the PCA analysis. While this analysis provides a very interesting set of features, with lower-order components that clearly resemble the global structure of a face (hence, Eigenfaces) the resulting recognition analysis will fail if faces are not upright and if they do not lie within a specific size range. Since it has been shown that CNNs can learn filters that are similar to those used in classic image processing [10], it is important to investigate the sensitivity of a given method to change in the nature of task difficulty (ie classification of constrained faces where Eigenfaces perform well vs unconstrained where it fails).

For this, 4 increasingly unconstrained datasets (AR faces [23], FERET [24], Adience [25], and Mit Places) were used (examples in figure 2.2). The AR Faces dataset, used for the majority of the initial experiments on the proposed augmentation approach, is composed of 4000 frontal faces of male and female subjects. Every subject has similar pose, lighting, and frontal orientation and they are all cropped in the same way. This dataset is easy for both conventional and DL algorithms.

Next the FERET dataset was used. FERET is still a highly constrained dataset by most standards but it contains some variation in pose and facial occlusions (glasses, beards, etc). Most traditional algorithms still perform well on FERET but not as well as they do on AR Faces.

Adience was the most challenging of the face datasets used. It was gathered automatically from Flickr albums and contains examples with poor lighting conditions, obstructions, multiple subjects, and difficult pose angles. Traditional algorithms do not perform well on this dataset, but methods based on deep learning do.

Experiments were repeated on all three face datasets and the average accuracy was recorded. Once it was established that smart augmentation worked well on all three face datasets, a fourth final challenge on a non-face-related task was chosen - The MIT Places dataset.

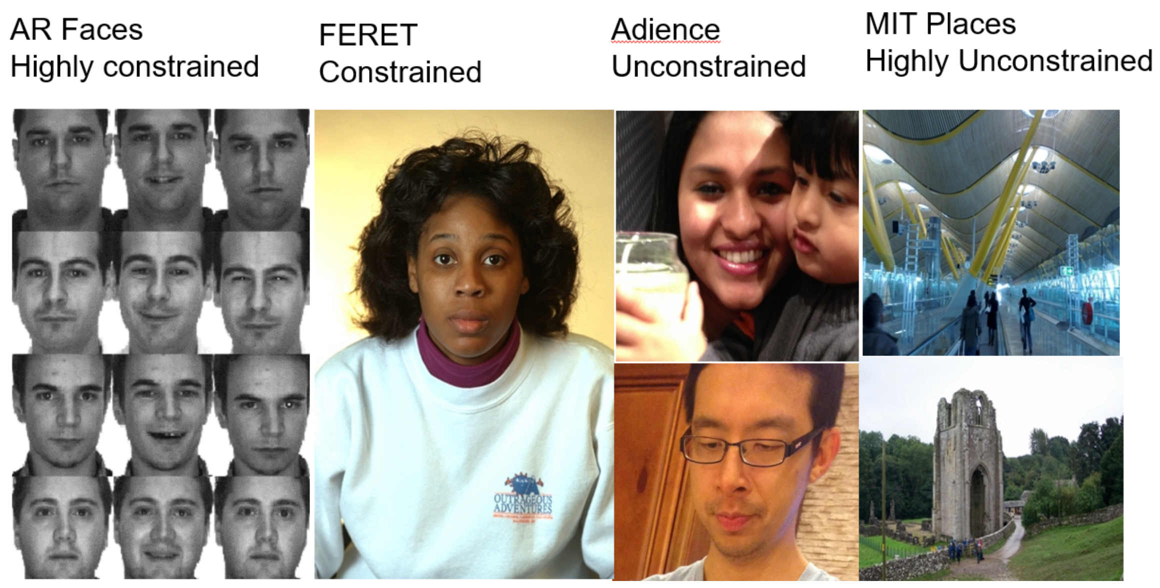


Fig. 2.2 Selected images from each of the 4 datasets used to evaluate Smart Augmentation demonstrate the progressively unconstrained nature of the tested images. This was necessary to examine the relationship between the complexity of the scene and the usefulness of the technique.

Since the Smart Augmentation experiments required one network A for each class, and because of the increase in training time this entailed, only 2 classes from the MIT places dataset were used: Abbey and Airport. Experiments followed the same methodology as used previously on the face datasets and demonstrated 99% accuracy with the use of smart augmentation compared to 96.5% without.

It was found that the use of smart augmentation improved accuracy over traditional augmentation on a variety of increasingly unconstrained data.

2.1.5 Smart Augmentation and Overfitting

When initially designing the smart augmentation concept, a very good question was asked: "Won't that just cause overfitting?"

This is a reasonable concern since the role of network A is to increase the accuracy of a target network on, the same training set that the target network is already being trained on and increasing accuracy on the training set is useless without a corresponding increase in accuracy on a validation or testing set.

One indication of overfitting is if the loss on the training set is going down (often close to zero) while the loss on the validation set goes up. Assuming both the testing set and the training set come from the same distribution, this test can show that overfitting has occurred but it can not verify conclusively that overfitting has not occurred. Training and testing loss for 1000 epochs of training using smart augmentation on the ARFACES database (see figure 2.3) did not indicate any overfitting and no evidence of overfitting was found in any of the experiments conducted.

More surprisingly, validation loss was lower with smart augmentation than without, indicating the possibility that instead of increasing overfitting, the use of smart augmentation decreased it.

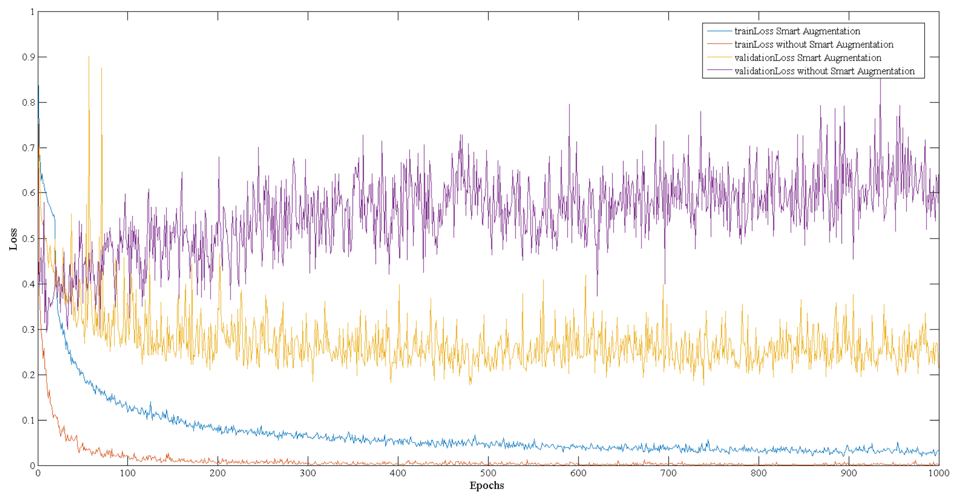


Fig. 2.3 This figure shows training and validation losses for 1000 epochs of training using the SA technique on the AR faces dataset. As explained in 2.1.5 this can serve as a test for overfitting.

2.1.6 Summary of Findings from Research Questions

The experiments provoked by these research questions demonstrated that the augmentation process can be automated, specifically in nontrivial cases where two or more samples of a certain class are merged in nonlinear ways, resulting in improved generalization of a target network. The results indicate that a deep neural network can be used to learn the augmentation task in this way at the same time the task is being learned. A convergence graph, supplied in the paper, demonstrated that smart augmentation can be used to reduce overfitting during the training process and reduce the error during testing. Details of these experiments and the methadology can be seen in B.

An increase in accuracy from 3.5 to 6.7 percentage points over baseline models was achieved depending on the dataset and experiment, and no linear correlation between the number of samples mixed by network A and accuracy was found, so long as at least 2 samples are used.

It was also shown that Smart Augmentation is effective at reducing error and decreasing overfitting, and that this is true regardless of how unconstrained the database is.

Thirdly, these experiments demonstrated that better accuracy could be achieved with smart augmentation than with traditional augmentation alone. It was found that altering the α and β parameters of the loss function slightly impacts results, but optimal parameters remain elusive and thus must be tuned by the experimenter.

2.2 Research Impact

Much of the the early work on Smart Augmentaion has already made an impact. This section describes new papers that are related to the original work on smart augmentation, many of which cite the original paper on the topic of learnable data augmentation with neural networks. For related work published before Smart Augmentation, see appendix B.

Thus in this subsection, the work in this thesis on data augmentation is put into a broader context. Focus is given to literature that is directly related to forms of learned augmentation, most of which were published after the original article on Smart Augmentation.

A review paper based on this section has been accepted for publication in Consumer Electronics Magazine.

In early 2017, the first papers on a new form of learnable augmentation were introduced. Since then, a number of other articles have been published exploring the idea of smart augmentation.

These techniques have in common that, instead of designing the augmentation process before training, they use artificial neural networks to learn the augmentation task at some point in the training pipeline. In this subsection, the latest techniques for using deep neural networks, and related techniques to generate data for augmentation, are explored.

Recent research has indicated that augmentation may be superior to other regularization approaches. For example, Hernandez et al. [26] [27] studies how augmentation can be used instead of regularization. They explain that many types of regularization (weight decay, dropout), waste model capacity by blindly eliminating learned information, and argue that data augmentation can do a better job at promoting generalization while more efficiently using network weight information.

Hernandez defines explicit regularization as techniques that are "specifically and solely designed to constrain the effective capacity of a given model in order to reduce overfitting. Furthermore, explicit regularizers are not a structural or essential part of the network architecture, the data or the learning algorithm and can typically be added or removed easily." [26]

He goes further to define implicit regularization as "the reduction of the generalization error or overfitting provided by characteristics of the network architecture, the training data or the learning algorithm, which are not specifically designed to constrain the effective capacity of the given model." [26]

Although these definitions are not universally accepted, and the number of datasets tested may be insufficient for such bold claims, they allowed the authors of that paper to classify data augmentation as a form of implicit regularization and form a series of experiments documenting how both forms of regularization impact model performance. These experiments demonstrated that in the majority of cases, using augmentation alone was a superior strategy to using explicit regularization or even augmentation and explicit regularization together.

2.2.1 Learnable Data Augmentation

Learning the data augmentation task with a neural network is a natural progression of the recent history of computer vision algorithms. Traditional computer vision involved creating a filter/feature detector, or set of filters/feature detectors, and transforming them in such a way as to allow a computer to make decisions about their content. The "Learning" part would often involve support vector machines (SVM), and any high dimensional features were often reduced with Principle Component Analysis (PCA) or Linear Discriminant analysis (LDA). For most computer vision tasks today, these methods, some of which were state of the art a little more than 5 years ago, are now archaic and only recommended in cases where Deep Learning performs poorly, typically when dealing with very small datasets or when Deep Learning is unnecessary, such as when an exact solution exists or when the task is simple enough to not need it. The problem with the traditional approaches is that they all require a person to decide ahead of time what features are most useful for a given task.

The phenomenal recent advancements in face recognition, object classification, voice recognition, and other tasks for which Deep Neural Networks are commonly applied are partly due to the fact that CNNs do not use hand-engineered features. Instead, they learn how to generate filters for input data as they learn their task.

The first paper describing fully learnable data augmentation, where all components of the augmentation pipeline are learned using an artificial neural network, appears to have been the paper on "Smart Augmentation" [11]. However, when researching related work for this thesis a paper that used a non neural network approach with the idea of learnable augmentation that was published few months earlier in 2016 [28] was found. Hauberg used statistical models of transformations for learning data augmentation tasks. Hauberg's approach can be considered the first approach to learning to augment data from the data itself using a statistical model, whereas Smart Augmentation [11] is likely the first approach to learn to augment data from the data itself using an artificial neural network.

2.2.2 Alternative Learned Augmentation Techniques

This subsection describe techniques for learned augmentation that do not use GANs or traditional statistical modeling and have been shown to perform well on image data.

Learning to Augment Data with a Neural Network

Another approach, similar to smart augmentation for learning augmentations that best improve a classifier called "neural augmentation" is discussed by Wang and Perez [29]. Neural augmentation takes two random images from a class and trains an augmenter that tries to generate images that reduce the loss of a target network. Neural augmentation has a number of similarities to smart augmentation and was reported to substantially improve accuracy (from 85.5% to 91.5% on dogs vs cats, and from 70.5% to 77.0% on cats vs fish classification). Neural augmentation appears to have been the result of a class project at Stanford and some of the author's experiments are similar to the early experiments performed when testing the idea of Smart Augmentation. Although they never published their technique as a journal or conference paper, their work is cited by many of the same papers that cite my work.

Augmentation Policy Learning

One alternative approach to learnable augmentation, called auto-augment, is to use machine learning to identify known augmentation techniques that cause better results for a specific dataset during training. This is a promising hybrid approach that allows learnable task specific augmentation policies, introduced in [30], that may be especially useful in cases where several traditional augmentations are suspected to work well. This approach can also be blended with approaches in the previous two subsections. Lin [31] and Lim [32] have proposed updated architectures for auto-augment that show speed improvements over the initial proposal.

Another approach to augmentation policy learning suggested by Ho et al. [33] was able to match the accuracy of auto-augment (state of the art) with 1000x less computational requirements on a number of datasets. Their method, called Population Based Augmentation (PBA), generates non stationary augmentation schedules, meaning a new optimal augmentation strategy is learned for each epoch.

Evolutionary Image Augmentation

An interesting type of learned augmentation using evolutionary image processing was investigated by Fujita et al in [34]. An image transformation tree using Automatic Construction of Tree-Structural Image Transformations (ACTIT) is used to create augmented images that are shown to improve results on various two class problems.

2.2.3 Methods Based on GANs and Statistical Generative Techniques

Generative models for data augmentation attempt to model the data found in the dataset and then use the generator to create unique image samples.

An early approach to learnable augmentation was suggested by Hauberg et al. [28]. They demonstrated a fascinating statistical approach to generative augmentation by modeling the transformations found within a given dataset. In this idea, augmentations are a type of deformation. A parametric model is used for generating transformations of input images. Promising results were reported on MNIST-like datasets of hand-written digits, but no further results have been reported for more challenging datasets. After the model has been trained to generate images similar to those in the dataset, generated images are sampled randomly from the parametric model, allowing for a large number of realistic transformations. A similar statistical model was used by Acero et al. [35] to augment MRI images to improve results on the cardiac segmentation task.

In addition to techniques based on classic statistical generative models, generative adversarial networks have also been used for the task of learnable data augmentation. Compared with classic generative models, this has the advantage that no assumptions are made about the types of transformations that should be allowed in a pre-augmentation step. Instead, GANs learn the distribution of the training set by competing against a discriminator and this competitive process alone determines the types of data that the GAN generates.

Several recent works on using GANs for augmentation have been published. For example, [36] used the GAN target loss to improve human pose estimation in a class-specific way. This paper is distinguished in that it learns to generate data at the same time it learns to perform its task. An interesting feature of this approach is the use of a "reward and penalty strategy" during training. Such strategies are common in reinforcement learning, but uncommon in GANs. As with all GAN techniques, the generator (augmenter) network attempts to find weaknesses in the discriminator. This corresponds to generating "hard" augmentations.

Another GAN approach to data augmentation is given by Ratner et al. [37]. A generative sequence model is trained over a transformation function. Like the method introduced by

Peng et al. this method uses reinforcment learning to generate data points that may be useful in the data augmentation task.

Wang and Perez [29] use a Cyclegan to generate images for use in data augmentation, but report that traditional augmentation techniques improved their results more than the GAN. Their inspiration for the use of style GANs was to train networks to address the problem whereby video data that is collected in one condition (daylight) when used for training will cause problems in other weather and lighting conditions (ie, fog and night).

Utilizing GANs for data augmentation is further studied by Poduturi in [38]. Poduturi used the GAN's latent vector to generate images for augmentation. A detailed analysis of error sources is provided and various augmentation techniques are ranked with results from the GAN performing competitively.

Antoniou et al. [39] use a conditional GAN to generate augmented samples, resulting in improved accuracy on a number of datasets. They call their method DAGAN (Data Augmentation Generative Adversarial Networks). They also apply their method to few shot learning and show that their method can substantially increase accuracy when only small amounts of data are available for training.

Zang et al. [40] extend a classic GAN with an augmentation module and a modified loss function they call 2k loss. They call their technique Deep Adversarial Data Augmentation (DADA). Like other class-aware GANs, the discriminator is adapted to return multiple class probabilities. The data augmentation module and classifier are learned at the same time and demonstrate state-of-the-art results on a number of small datasets that would typically be difficult for GANs to learn. Zang reports competitive results on EEG, breast imaging, and tumor classification tasks.

Finally Tran et al. [41] introduced a Bayesian approach to data augmentation, using generalized expectation maximization, whereby augmented data are treated as missing variables. These variables are sampled from the distribution learned from the training set. The proposed GAN is based on AC-GAN.

2.2.4 Data Augmentation for Audio

A majority of the papers published on data augmentation focus on visual data, and the most popular techniques are not relevant to audio data. With the rise of smart speakers and intelligent audio assistants, it is important not to neglect data augmentation for neural networks that operate on audio. For this reason a subsection is devoted specifically to dealing with the augmentation of audio data. The majority of these techniques augment on spectrogram data (linear or mel).

The interest in applying augmentation to audio data comes from a collaboration with Aoife Mcdonagh when she was starting her research in deep learning for audio. Although most of the work that was intended for audio augmentation never occurred due to new priorities, it is still worth discussing this thesis.

As this work never progressed to the point of using it for data augmentation, we instead applied it to the problem of generating unique audio for video games.

The paper that came out of my collaboration proposed a GAN for unique audio generation that can be applied to augmentation in Synthesizing Game Audio Using Deep Neural Networks [18] by latent space interpolation or by randomly sampling the Z vector. Although no evaluation of performance on augmentation tasks was done in the paper, the similarity of the method to GAN-based augmentation techniques provides reason to believe the method will work for this task. Raw waveforms are used rather than spectrograms. In a way this represented an attempt to apply one of the core ideas that made Smart Augmentation work to audio by training a network to learn to combine features learned from sounds to create new distinct sounds. In that work, the latent space of the Z vector was used to synthesize unique audio that has the properties of two or more audio clips, just as smart augmentation created new images that had the properties of two or more images.

My contribution was the original idea of using a GAN to synthesize unique audio using latent space interpolation of two classes, writing parts of the paper, and processing the output from the trained GAN to gather data. The majority of the work was performed by a colleague. Ideas for extending this work using a recurrent neural network are included in a filed patent application.

The remainder of this subsection includes a short review of audio augmentation techniques.

Shluter et al. [42] investigates label-preserving transformations (augmentation strategies) for audio signals with a focus on singing voice detection, and measures the techniques that result in the best performance. They train a CNN to perform classification on spectrograms of voice sounds modified with the following transformations:

- Noise (dropout and Gaussian) applied directly to the spectograms
- Pitch shifting and time stretching by scaling spectograms vertically (for pitch) or horizontally (for time) followed by additional processing.
- Loudness (by randomly scaling linear spectograms)
- Random frequency filters
- Mixing two music excerpts using linear spectograms.

It was found that random frequency and pitch shifting improved results significantly, but the remaining filters were ineffective or reduced accuracy. State of the art results were achieved when using frequency randomization and pitch shifting. Some of these filters can be implemented in the Audio Degradation Toolbox, in matlab, or in the munda Musical Data Augmentation package in python. These packages were both designed for performing audio augmentation.

Park et al. used a similar approach with the introduction of an audio augmentation method they call SpecAugment that operates on mel spectograms [43]. They demonstrate SpecAugment for automatic speech recognition without the need for a language model (although they saw further improvements when language models were used). State of the art results were achieved on LibriSpeech and Swichboard 300h datasets. Warping and masking (on time and frequency) policies are used for augmentation.

Vatolkin et al. proposed an evolutionary optimization technique to identify what they call "smart data", that is, sound samples that contain the most relevant information for a given class [44]. They apply their proposed technique on vocal activity detection.

2.2.5 Other Recent Augmentation Techniques

In this section, new augmentation techniques that, although they would not qualify as smart augmentation or learned augmentation in a strict sense, contain a learned component or show a significant improvement from a new augmentation technique.

Devries et al. describes a new method of data augmentation called "cutout". The approach works by applying a fixed-size zero mask to a random location of each input image during training. Surprisingly, this simple technique yielded state-of-the-art results on the CIFAR-10, CIFAR-100, and SVHN datasets in August 2017. This illustrates the ability of simple transformations to greatly improve network performance [45].

Another interesting recent technique called MixUp [46] [47], provides a method for automatic augmentation by interpolation. This method has been shown to mitigate adversarial attacks on neural networks. The explanation for why mixup provides protection from adversarial examples is that the decision boundaries produced by mixup transition smoothly and linearly from one class to another. Mixup is related to other forms of learned augmentation because the interpolations are learned from the data.

A method is presented by Duan et al. that uses information about which neurons activate during training to create augmented images and found that their method improved accuracy by 11.31 percent on the CUB200-2011 dataset [48]. This approach is somewhere between a learnable "smart" augmentation and traditional augmentation.

2.2.6 Discussion

Learnable augmentation is a new, highly active sub-field of deep learning research that has the goal of reducing the human labor inherent in selecting, designing, and validating data augmentation. Common approaches to learnable augmentation include policy learning, learning to generate images with a GAN or other network, and building statistical models. It is interesting that, while a wide variety of methods work well on image data, there are far fewer methods that work well on audio data. The investigation of smart augmentation for audio data thus remains an interesting possibility for future work.

2.3 A Discussion on Smart Augmentation and GANs

Shortly after I first came up with the idea of smart augmentation and ran the initial experiments, a patent application needed to be prepared. One of the benefits of being placed in a company with so many talented engineers is that rapid practical feedback is often possible: especially when an idea is being considered for patentability. One engineer was particularly adamant that our smart augmentation idea was a type of GAN and the arguments in this section are based off my counter argument to him on why the method should not be considered a GAN.

In this subsection, important differences and similarities between Smart Augmentation and a popular related technique - Generative Adversarial Networks are discussed.

Generative Adversarial Networks (GAN) were developed in 2014 by Goodfellow, et al [49]. The main idea was to synthesize examples of observed data in an unsupervised way by use of competing networks. To explain how this works, the paper uses the analogy of police vs counterfeiters.

One network (the counterfeiter) attempts to trick the other network (the police) into thinking that a generated image is legitimate. The second network (the police) attempts to learn how to detect the counterfeits in a minimax fashion. The goal is that through this competitive process, the counterfeiter network will be able to produce images that are indistinguishable from real images.

Specifically, the generative model in a GAN generates samples by passing random noise through a multi-layer perceptron. The discriminative model is also a multi-layer perceptron. The models are trained using backpropagation and dropout.

Generative Adversarial Networks use competition to generate an image which is not used for augmentation but is instead used to make images that are “similar” to other images that the discriminator has seen before.

Additionally, Generative Adversarial Networks and smart augmentation have differing objectives. In Generative Adversarial Networks they make more images that are similar to images that have been seen. This is the entire goal. With Smart Augmentation the idea is to train a network to use the joint information between image samples to improve a second network which later operates independently. Furthermore, Smart augmentation is not a process for generating data. It is a process for training a separate network that includes a generative component.

Importantly, Smart Augmentation is considered to be working if it generates data that improves a target network, regardless of whether that data looks "good". The performance of GANs is measured based on their ability to create "realistic" images that seem similar to others in their class. The subjective nature of evaluating GANs has led to subjective metrics such as mean opinion score, inception score [50], or Fréchet Inception Distance [51] in evaluating GAN performance. Smart Augmentation is evaluated based on easily and objectively measurable criteria: The amount by which the target network has improved (in terms of accuracy, ROC scores, precision/recall, or other classification metrics).

2.3.1 Differences

Unlike GANs, Smart Augmentation does not use competition, and utilizes a loss that has no relation to that used by GANs. Furthermore generative adversarial models are unsupervised learning tools that disregard class labels, whereas our model is supervised and has a network tuned to the best augmentation strategy for each class.

Generative Adversarial Networks also require large amounts of unlabeled data, whereas our method is designed to work with classes that may not have very many samples.

With GANs, one typically uses the discriminator only during training, using the trained generator to create content as a "stand alone" network. In smart augmentation the opposite approach is used. The generator, our network A, is discarded and only the classifier is used for the given task.

The Smart Augmentation model learns to merge k images based on similarity to a specific arbitrarily chosen image in the same class, as well as the loss function of the target network. Generative Adversarial Networks learn to create images similar to images in the dataset in a different way: instead of learning to merge images they learn to generate images by performing minimax optimization between the generator and discrimator. Therefore both GANS and SA use information from the loss function of more than one network, but the information used and what they do with that information is different.

Another difference is that SA does not pass noise through a multilayer perceptron. Instead, multiple images in channels are passed as input to a network that learns to modify them. It is important to note here that there do exist GAN models that use variation in the training set as the "noise" instead of the original random input idea while retaining other aspects of GANs. The most popular of these is CycleGAN [52]. Unlike regular unconditional Gans, which learn to map random noise to an image, CycleGAN learns to map an image to another image as well as the reverse operation. Specifically it learns an objective for bidirectional mapping between a source image x and a target image y. For such networks, this argument about the input does not apply, however "CycleGAN" was published concurrently with "Smart Augmentation".

2.3.2 Similarities

Generative Adversarial Networks also have a network “A” that generates data (typically images or sound). Network A in Generative Adversarial Networks also uses information about how network B (the discriminator) is interpreting its input. This similarity is only in the abstract sense because the nature of the information used and what it does with the input are different. Generative Adversarial Networks and network A in the Smart Augmentation system both create artificial images that are similar to other images in their data set, although the methods by which this is accomplished are different.

Both networks’ models have a “generator” and “discriminator” component. With Smart Augmentation Network A is a “generator” and network B could be considered the “discriminator” although this same terminology is not as meaningful for Smart Augmentation.

2.3.3 GASA

The unique properties of GANS and Smart augmentation inspired an experiment whereby a GAN was used in place of network A with a classifier and similar loss structure. The research question in this case was "Will the generator of a GAN (specifically BEGAN [53]), with input from a classifier’s loss during training, improve the training of a target classifier. This implementation was called GASA (Generative Adversarial Smart Augmentation) and the architecture is shown in figure 2.4. It was predicted that this would not improve the results because the GAN would be pushed to making images that are as close as possible to those which were already well understood by the classifier.

Our initial experiments indicated that this assumption was correct. GASA was not successful as our experiments indicated no improvement over not augmenting at all. Further GAN/SA experiments for data augmentation were not attempted due to this negative result and more pressing research priorities, but this negative result does not imply that it would be impossible for a successful GASA-like idea to work in the future.

Although the work on GASA was not used for augmentation, the idea was later successfully adapted by my coauthor to perform the opposite task: to improve the generator of GAN by attaching an auxiliary classifier or regression network.

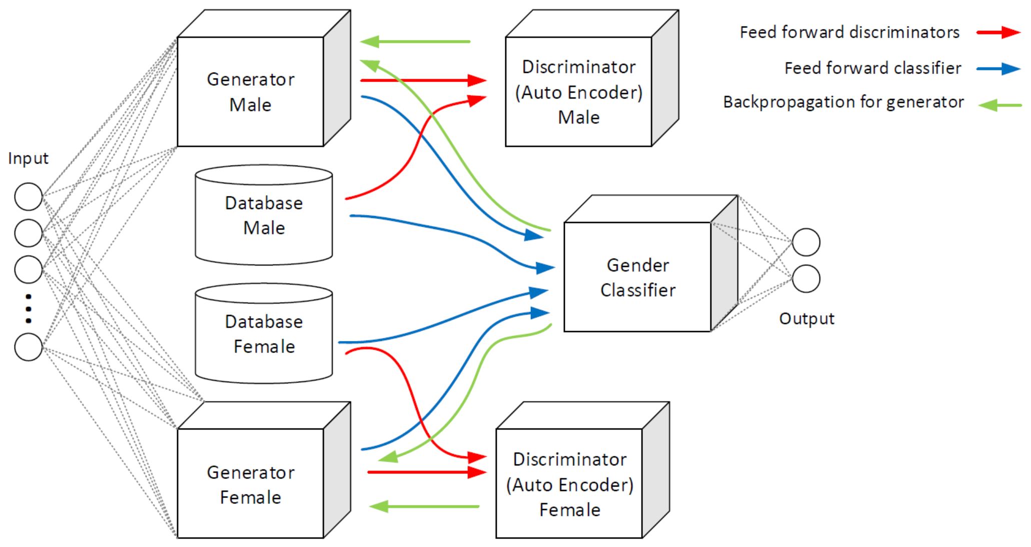


Fig. 2.4 Diagram of unsuccessful GASA idea built for the perceived gender recognition task. In this case two generators take the place of two network As in the smart augmentation model.

2.4 Smart Augmentation and other Similar Techniques

The concept of using one network to train another is not unique. One example is the so called "student teacher networks" or just "Student Neural Networks" where one network is used to train another network in such a way as to mimic its output [54]. A similar concept is used for neural cryptography which uses a neural key exchange mechanism to secure communication [55]. The method shows promise in the future because it is not vulnerable to the varieties of cryptographic attacks that are theorized to become feasible with quantum computers or other unforeseen advancements in factorization algorithms.

A brief overview of the main differences between smart augmentation and the above mentioned methods follows:

1. Neither student/Teacher networks nor neural cryptographic networks use an augmentation strategy as part of their design.

2. Neither Student/Teacher networks nor neural cryptographic networks mix or merge loss functions during training.

3. In the design of Student/Teacher networks, network “A” is the teacher from which network B learns to classify the same material. In neural cryptography network A and network B negotiate a key exchange mechanism. Our network A generates Augmented data and takes in multiple inputs at once while network A and network B learn from eachother. The other networks don’t do this.

4. These networks do not have selector functions that allow comparison with existing data.

Both Smart Augmentation and the above networks use one network to train another - this is common to many meta-learning approaches. In neural cryptography, the training process happens concurrently (i.e., network A and B are “trained” simultaneously). This is the same with Smart Augmentation. Student/Teacher networks do not typically share this commonality.

Chapter 3

Eye Gaze

Eye gaze tracking and gaze-based human computer interactions in modern consumer devices are an important aspect of exploring human interface design (HID). Eye gaze has been used to derive human behavioral cues, as an input modality and for achieving immersive user experiences in virtual and augmented reality systems.

After decades of research on desktop-based gaze estimation techniques, the focus has recently shifted to building eye gaze applications for dynamic platforms such as driver monitoring systems [56] and handheld devices [57]. For an automobile driver, eye based cues such as levels of gaze variation, speed of eyelid movements and eye closure can be indicative of a driver's cognitive state. These can be useful inputs for intelligent vehicles to understand driver attentiveness levels, lane change intent, and vehicle control in the presence of obstacles to avoid accidents [58]. Handheld devices like smartphones and tablets form unique platforms for gaze tracking applications wherein gaze may be used as an input modality for device control, activating safety features and novel user interface (UI) designs [59]. Eye gaze estimation is also an essential component on the path to autonomous driving as the vehicle needs to know when it is safe to surrender control to a driver.

The most challenging aspect of these modern gaze applications includes operation under dynamic user conditions and unconstrained environments. Further requirements for implementing a consumer-grade gaze-tracking system include real-time high-accuracy operation, minimal or no calibration, and robustness to user head movements and varied lighting conditions. Therefore accurate and reliable gaze tracking typically demands high quality cameras and special equipment like narrow angle lenses, external illumination, and stereo setups [60] for capturing eye region features with sufficient details. As a result, gaze estimation systems frequently become costly with complicated setups, which are unsuitable for generic and consumer applications.

Therefore a major challenge of gaze-based consumer electronics design involves maximizing system performance while reducing costs and system complexities.

Gaze-tracking algorithms can be broadly classified into two types: model-based methods and appearance-based methods [61]. Appearance-based methods operate directly on the eye images. Examples of model-based methods include 2D and 3D models that use near infrared (NIR) illumination to create corneal reflections and track them with respect to the pupil center to estimate the gaze vector. These require polynomial or geometric approximations of the human eye to obtain the gaze direction or the point of gaze. Appearance-based methods use eye region images to extract content information such as local features, shape, and texture of eye regions, to estimate gaze direction.

Contemporary research on gaze tracking measures accuracy in a wide variety of ways [60]. For example, commonly used measures include angular resolution in degrees [62], gaze recognition rates in percentage [63], and shifts in number of pixels or distance in cm/mm between gaze [64] and target locations.

3.1 Exploring Eye Gaze Estimation for DMS

This research on eye gaze followed from the earlier work on driver monitoring systems [16] and was part of the larger participation in training neural networks for the Fotonation/Xperi driver monitoring system technologies and algorithms.

Some of this work, due to the significant improvements in performance over alternative approaches became central to the Fotonation's first DMS system. This chapter contains an overview of work that can be publicly disclosed. Other significant improvements were later made but cannot be fully detailed here due to the commercially sensitive nature of Driver Monitoring systems.

However driver monitoring systems based on it are currently being used commercially, have been shown at CVPR 2019, and helped to win the Irish research innovation award. These developments eventually resulted in the opportunity to lead a small team of R&D engineers developing a series of neural networks for driver monitoring tasks such as head pose, eye gaze, occlusions, driver action recognition, and other core driver monitoring and occupant monitoring technologies.

Fig. 3.1 CVPR 2019 demo which included an eye gaze network that I contributed to as part of a complete driver monitoring system.

3.2 Convolutional Neural Network Implementation for Eye-Gaze Estimation on Low-Quality Consumer Imaging Systems

This section summarizes the work taken from my journal article [14] about the implementation of an eye gaze estimation system for a driver monitoring system, and continues with the theme of investigating advanced data augmentation strategies and increasing the performance (both speed and accuracy) of convolutional neural networks in consumer applications.

The major contributions are a new CNN for eye gaze estimation and analysis of a method for improving invariance to user distance by data augmentation. A major challenge of gaze-based consumer electronics design involves maximizing system performance while reducing costs and system complexities.

Specifically, it describes work on a calibration-free method for appearance-based gaze estimation that is suitable for consumer applications and low cost hardware with real time requirements, using a Convolutional Neural Network (CNN). An analysis of both model architecture and an augmentation approach is provided.

When deploying eye gaze solutions for consumer devices, there are two important aspects to consider: accuracy and efficiency. This work addresses both issues by demonstrating improved accuracy and also by reducing the number of multiplications needed for predictions, thus increasing efficiency. It should be noted that the total number of matrix multiplications needed to obtain predictions from a convolutional neural network is determined by the size of the convolutional kernels, the step size, the number of nodes in each layer, and the number of layers [10]. These multiplications are often measured in multiply-accumulate operations (MACs).

3.2.1 Contributions of this Work

From the perspective of developing a deep learning model for gaze estimation, the task can either be considered a regression task or a classification task. Although both are useful, regression provides the greatest predictive flexibility and thus this paper treats the eye gaze estimation task as a regression problem with the goal of finding a gaze angle (ϕ,θ) that corresponds with a low resolution eye image such as one taken from a distance with a simple RGB webcam mounted on a dashboard.

A hardware optimized network is implemented with demonstrated suitability for deployment on such consumer devices in terms of memory requirements and speed. This network achieves superior accuracy using a dual channel input technique when compared against other state-of-the-art CNN-based gaze tracking methods for unconstrained, low resolution eye tracking.

Table 3.1 Result of Distance Simulation and Augmentation Experiments

Resolution	Unaugmented error	Augmented error
60 x 36	4.63 degrees	4.918
52 x 31	9.90 degrees	4.94
26 x 16	10.10 degrees	4.98

This table shows the impact of camera distance and augmentation for a trained eye gaze model on angle accuracy.

This demonstrates that the model is sensitive to changes in distance. In the next section, an experiment is performed to see if data augmentation can be used to improve upon this.

3.2.2 Impact of Random Resizing as Augmentation

Data augmentation has been shown in many studies [65] to have a large impact on model performance, but augmenting to increase accuracy on a wide range of distances appears to be neglected in literature on eye gaze. To further improve accuracy, the dataset was augmented with multiple randomly chosen resolutions to match the full range of desired distances. To help reduce the chance that the network would learn the specific interpolation method used, Nearest is used in the training set, but Lanczos filtering is used in the testing set.

These results indicate that augmenting the images with distances that are likely to be encountered in real world usage situations is an effective way to increase accuracy and succeeds in achieving some invariance to subject distance.

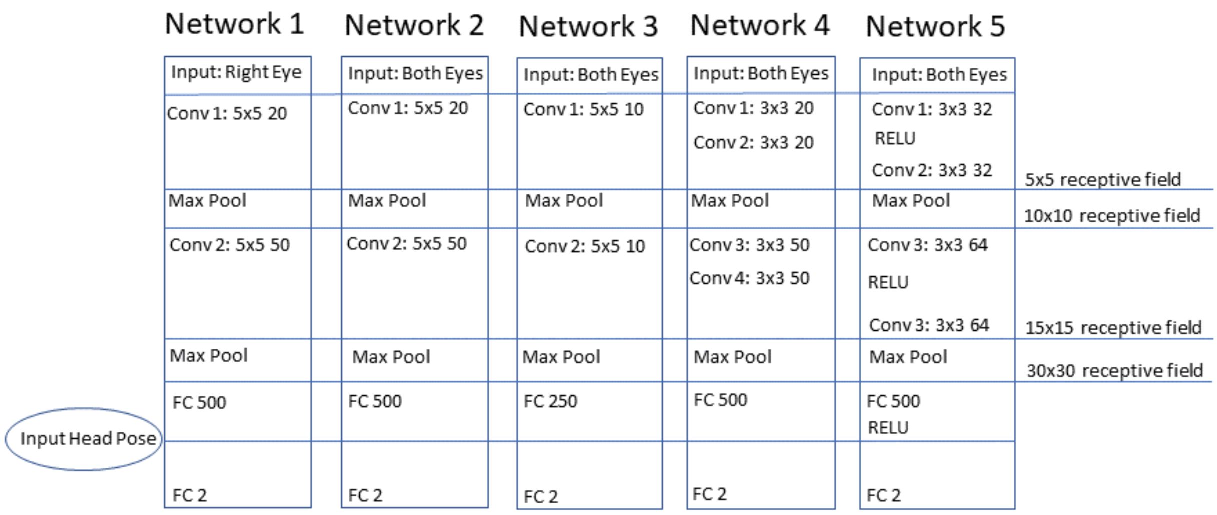


Fig. 3.2 This figure contains network architecture details for the 5 networks examined in this section. See Appendix D for more details.

3.2.3 Models Examined

The Five Networks

Five networks used in this subsection were evaluated on commodity processors for both accuracy and speed. The result of speed experiments are in the table 3.2. A comparison of the best performing network with related work is shown in table 3.3, and the results of an experiment examining an augmentation technique using these networks are shown in table 3.1 the architectural details are shown in figure 3.2 the data flow is shown in figure 3.3. Additional experiments, results, and methodology are provided in appendix B.

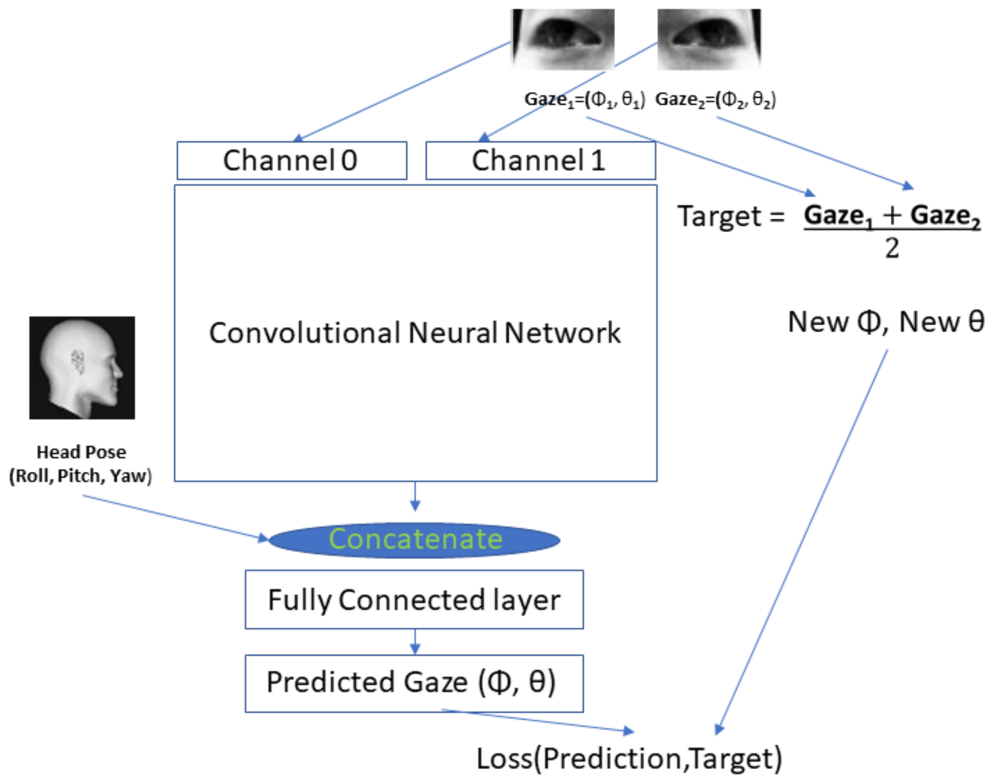


Fig. 3.3 This diagram provides a visual overview of the data flow for networks 2-5 used for eye gaze estimation. Layer details differ for different networks and are illustrated in Figure 3.2.

The first network used (Network 1 in figure 3.2) was the “one channel” approach used in [66], with which I compare. In contrast to the new approaches, this method performs a separate inference for each eye and flips the right eye. The reason the right eye is flipped, and the corresponding vectors recalculated, is to allow the network to learn a single model for a given eye patch without consideration to the side of the face it is on. My first step was to duplicate the results of [66] on the same network using the same architecture to establish a reliable baseline for performance on the mpii-gaze database.

The next network architecture (Network 2 in figure 3.2) involved using both eyes in separate channels, input to the network at the same time. Experiments with this network showed that this architecture change increased accuracy.

A variation of network 2, Network 3 in figure 3.2 contains half as many outputs per layer and provided a significant increase in speed at the cost of a slight reduction in accuracy.

The fourth network replaces network 2 with 3x3 kernels (as shown in figure 3.2 such that the receptive field is maintained but without additional changes. This network was included to differentiate between results that occured due to the increase in outputs and addition of nonlinearities in network 5 from those due to the 3x3 kernel replacement.

The fifth network (network 5 in figure 3.2) adds RELU nonlinearities to network 4. This is the best performing model I examined for eye gaze. The replacement of 5x5 kernels with 3x3 kernels was motivated by the needs of an embedded system where the corresponding reduction in multiplications with the same receptive field was important for performance and implementation reasons.

Table 3.2 Frames Per Second of CNN on Commodity Hardware

Model	ARM Cortex-A53	AMD 1950X Threadripper CPU	NVIDIA 1080 TI GPU
Network 1 [66]	20.64	473.11	3984.09
Network 2	39.81	960.06	8078.47
Network 3	92.59	3080.80	12607.90
Network 4	29.68	592.02	6134.02
Network 5	19.11	400.50	5500.37
VGG16 [67]	0.0043	0.64	120.4

All units are in frames per second. Details of these networks are discussed in appendix D. As can be seen from this table, networks 2-4 perform faster on all tested hardware than the comparison networks (Network 1 and VGG16) while providing greater accuracy. Network 4 has similar performance to network 1 while increasing accuracy. The well known VGG16 network is included to allow the reader to see the speed gain over this type of architecture. VGG16 is an improvement over Alexnet, which was used to achieve 4.8 percent error on the same dataset [68].

Table 3.2 shows the results of an experiment to measure the runtime of the 5 networks with the popular VGG16 included for comparison.

The embedded processor used for these experiments was the ARM Cortex-A53, a typical 64 bit processor used in embedded and mobile systems, and available on a well-known embedded prototyping platform: the Raspberry PI. Tests used only one core of this processor.

The second processor, an AMD 1950X Threadripper, is a popular high end workstation central processing unit (CPU). For a fair comparison, only one thread was used for tests. Lastly the GPU used is a popular high end consumer GPU targeted at video gamers, the NVIDIA 1080 TI GPU. As can be seen in Table 3.2, network 3 is significantly faster than the

Table 3.3 Comparison of Proposed Model and Other Published Works on the MPII-Gaze Database

Citation	Error (degrees)
Baltrusaitis et al [69]	9.96
Wood et al [70]	9.58
Shrivastava et al [71]	7.8
Nie et al [72]	7.1
Zhang et al [66]	6.1
Zhang et al [68]	4.8
Proposed	3.65

This table shows the best performing network from the proposed networks compared with other reported results on the MPII-Gaze dataset. See appendixF for methodology.

other networks while network 5 provides a good balance between speed and accuracy. In some cases network 3 may be preferred due to increased speed and competitive accuracy.

In order to facilitate a fair comparison with [66] the loss function for all networks follow that which was used by the authors of that paper, and is calculated according to the following equations, where ϕ and θ are the predicted gaze and $\hat{\phi}$ $\hat{\theta}$ are the corresponding ground truth.

$$norm_1 = \sqrt{(-1 \cdot cos(\phi) \cdot sin(\theta))^2 + (-1 \cdot sin(\phi))^2 + (-1 \cdot cos(\phi) \cdot cos(\theta))^2}$$

$$norm_2 = \sqrt{(-1 \cdot cos(\hat{\phi}) \cdot sin(\hat{\theta}))^2 + (-1 \cdot sin(\hat{\phi}))^2 + (-1 \cdot cos(\hat{\phi}) \cdot cos(\hat{\theta}))^2}$$

$$\begin{aligned} angle = (-1 \cdot cos(\phi) \cdot sin(\theta)) \cdot (-1 \cdot cos(\hat{\phi}) \cdot sin(\hat{\theta})) + \\ sin(\theta) \cdot sin(\hat{\theta}) + (-1 \cdot cos(\phi) \cdot cos(\theta)) \cdot (-1 \cdot cos(\hat{\phi}) \cdot cos(\hat{\theta})) \end{aligned}$$

$$loss = \frac{acos(\frac{angle}{norm_1 \cdot norm_2}) \cdot 180}{\pi}$$

3.2.4 Discussion and Overview of Major Findings

It was found that by changing the network architecture to accept two eye crops, one for the left and one for the right eye in two input channels, and merging the gaze vectors and the position vectors, I was able to improve accuracy over that reported in Zhang et al [66]. It was then shown that the increased performance could be achieved by halving the number of nodes in each layer with a slight decrease in accuracy.

An experiment was done that demonstrated that the error more than doubles when the network is exposed to images that have been artificially resized to simulate a wide range of distances between the subject and the camera. This problem had not been addressed in previous work. The solution to this was to augment the data with random simulated distances. Finally, an improved network architecture that outperforms previously published works while reducing the number of multiplications, and thus increasing efficiency, was proposed.

These results show that using information from both eyes in the neural network can increase accuracy. In this experiment, adding additional eye information from the opposite eye enabled improved results over individual eyes, helping the network make sense of low quality images with ambiguous gaze. As expected, in all cases, the deeper network had the best performance. This research demonstrated the sensitivity of such models to variations in distance and how data augmentation can be used to overcome this. Most importantly, a new, compact, hardware-friendly architecture designed for use in small consumer electronics has been introduced and evaluated on the eye gaze task.

When evaluated on MPII Gaze, the proposed model performs favorably even when compared with much larger networks in the literature. Running an optimized CNN based algorithm such as this can provide a high-performance, low-energy solution for continuous eye-tracking in next generation consumer electronic products.

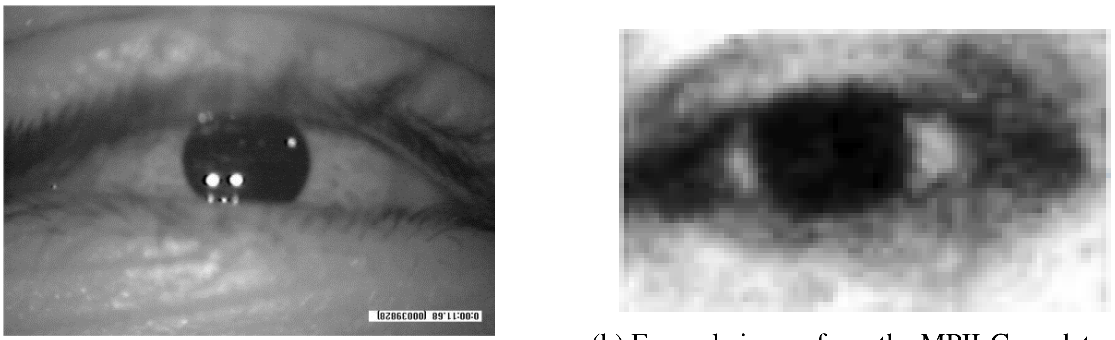
In future work it would be interesting to see if Smart Augmentation could provide further improvements but this would require either posing the eye gaze task as a classification task or updating Smart augmentation to work with regression tasks, and neither of these options would be trivial to implement with Smart Augmentation.

3.3 Extending the Idea to Augmented Environments

The use of deep learning for estimating eye gaze in augmented spaces estimation is investigated in this section. The work in this section was presented at GEM 2018 in Galway Ireland. The corresponding publication is attached in appendix E.

There are two primary ways of designing systems to facilitate interaction with augmented spaces. The first involves the use of AR/VR systems where an eye facing camera is attached to the AR/VR system. This provides clear, sharp eye images that are usually a fixed distance from the user.

The other approach is to use a single camera or array of cameras/sensors at a distance. This approach typically results in lower quality images but (in the single camera case) is less expensive to implement and allows a single sensor to estimate the gaze of more than one



(a) Example image from the McMurrough dataset, showing subject's right eye. This is typical of AR/VR systems that utilize eye facing cameras to estimate gaze.

(b) Example image from the MPII-Gaze dataset, showing subject's right eye typical of gaze tracking systems that use low quality cameras that are further from the subject.

Fig. 3.4 Traditional methods work best on images that are typical of an AR/VR system such as 3.4a but convolutional neural networks perform the best for low quality images such as those shown in 3.4b

subject without requiring the use of head gear. This approach is typical in DMS systems and is covered in detail in the previous section.

The use of deep learning for gaze estimation in augmented spaces for AR/VR use cases is not well explored in the literature, and dedicated network models or datasets for gaze estimation in such environments are not publicly available.

High Resolution Datasets for AR/VR Gaze Estimation

As mentioned previously, there are not enough public datasets from which deep learning systems can be trained (or evaluated) for eye gaze estimation utilizing head-mounted eye-facing cameras. The only suitable publicly available dataset found is the one developed by McMurrough et al called the "Point of Gaze (PoG) Eye Tracking Dataset" [73]. Unfortunately, this dataset only has images of the right eye and therefore may not be used for AR applications where knowing what a person is looking at in 3D space involves calculating the intersection of two gaze vectors. This information is still useful because there are typically only a few objects that collide with a given gaze vector that are within a person's field of view and these can all be assumed to be the gaze target in an augmented or virtual space. Despite this limitation, the PoG Eye Tracking Dataset is the most suitable publicly available dataset captured with a head-mounted eye tracker and was therefore used for this research. An example of typical images from PoG and MPII-Gaze are shown in figure 3.4

Because only right eyes were available, the network described at the start of this chapter could not be used directly. Instead, a single eye version of this network was used in addition to other architectures described in appendix E.

To create the Point of Gaze dataset, twenty participants (18 men, two women) were asked to track target points on a video display while wearing an Applied Science Laboratories Mobile Eye™ infrared monocular recording device. The participants' right eye is centered in the video frame and is annotated for specific target points. The dataset is composed of 20 subjects with ages ranging from 21 to 54. The dataset is annotated with head pose and eye gaze information. Eye images are recorded with a resolution of 768 X 480 pixels at 29.97 Hz frame rate.

Summary of Experiments and Results

Several experiments were performed on the Point of Gaze dataset using state of the art deep learning tools but it was found that, contrary to the case of images taken at a distance, Deep learning models did not perform as well for images that are typical of AR/VR setups as typical traditional approaches.

One consideration when comparing or evaluating these results is that the average accuracy includes frames where eyes are not open. This is a deliberate choice, as CNNs may be capable of estimating gaze of even closed eyes therefore comparing these results with those that disregard closed or partly open eyes would be misleading. Although this methodological detail could explain some of the decrease in performance, it is insufficient to explain it fully as the decrease in accuracy is observed even in fully visible eyes.

As deep learning begins to surpass traditional techniques in many eye gaze tasks, it is of interest to investigate its potential for gaze estimation on AR/VR setups. In this research, several CNN architectures were used to try to improve upon traditional gaze estimation techniques for AR/VR use cases.

Although the error +- (1.329cm x 4.2246cm) of the best trained model, based on Xception [74], would make it suitable for many gaze estimation tasks, the pure CNN model still underperformed traditional methods.

Chapter 4

Transfer Learning of Temporal Information for Driver Action Classification And Semi Parallel Deep Neural Networks

In this chapter work that does not form the core of this research, but nonetheless supports the advancement of the application of machine learning and deep neural networks within the limitations that arise on small consumer devices, is documented.

4.1 Transfer Learning of Temporal Information for Driver Action Classification

In early 2017, after the first work on Smart Augmentation, research on driver monitoring systems became a priority at fotonation/Xperi as an early feasibility study on their implementation. This early work later led to the publications on eye gaze as part of the efforts of the in-the-cabin monitoring long term research and development group at fotonation/XPERI which this work was performed.

This section contains an overview of the findings from this original feasibility study, and the corresponding conference paper "Transfer Learning of Temporal Information for Driver Action Classification". This paper was presented orally at MAICS 2017 in Dayton Ohio. Neural Network Architectures that utilized a temporal component for the driver monitoring task were explored and compared with the more commonly used frame-based approaches.

This early exploratory work on driver monitoring systems was never implemented in a product, but it informed later work that became part of successive generations of AI-based driver and occupant monitoring systems. It also led to my current role as a project manager for the Fotonation's Heliaus project team, an H2020 project with the goal of developing driver monitoring systems using thermal cameras. Further information on this project can be viewed here http://www.heliaus.eu).

4.1.1 Frame Based Methods and the Need for Temporal Information

Although frame based methods for determining action can perform with high accuracy for some tasks such as "eating" or "smoking", other tasks cannot be effectively differentiated without information from more than one frame. For example, by looking at a single frame it may be difficult to determine if a person is putting down a cup or picking up a cup, or more importantly, taking control of the steering wheel or letting go of the steering wheel. In many cases, the added temporal component can also improve the robustness of algorithms that already work well for single frame tasks, especially in the case where one frame is ambiguous but the following or preceding frame can be understood with high confidence. By including a concept of time in a model, the network is often better able to make predictions. Additionally, temporal information allows for the concept of action that is more similar to the way many animals process visual information, where greater attention is paid to things that move, or the way that humans use temporal information for peripheral vision.

As we approach the limits of frame-based methods, there is a desire to further improve deep learning algorithms by utilizing temporal information, which is information between multiple frames taken sequentially to give a more complete idea of what is happening. Using a single frame, it is trivial to train a classifier to determine if a person is holding a glass, but difficult or impossible to train a classifier to understand if the glass is being picked up or put down. Even distinguishing jogging from walking can be difficult without a time component.

4.1.2 Data Used for Training and Testing the Temporal and Frame Based Networks

Correct classification of image data can depend on features learned in multiple sequential frames. In this section the problem of learning action from video data with an emphasis on driver behavior monitoring is studied. An insufficient quantity of high quality labeled data is a major problem in machine learning research. This is especially true when deep neural networks are used.

Although some sufficiently large, general purpose image databases exist for action recognition, most of these are limited to single frames. This kind of data requires that the action recognition task is applied without considering temporal information (information from previous and next frames of a video sequence).

The largest database for driver behavior monitoring that could be found was "Distracted Driver Dataset", provided as part of a Kaggle challenge in mid-2016. Although this database is intended for single frame classification, it is possible to identify the original frame sequences from which movies can be created. These movies can then be used for learning a limited amount of temporal information.

The Distracted Driver Dataset was provided as part of a Kaggle Competition in 2016. The dataset was created by filming actors on a closed driving course engaging in various distracted and undistracted behaviors. It should be noted that these images were obtained in a controlled environment and the car was not actually being driven. It was being pulled by a truck instead. The objective of the competition was to correctly classify still images into 10 categories.

The training set of the distracted driver database contains frames of 26 subjects displaying several of the following behaviors/actions:

1. c0: safe driving
2. c1: texting - right
3. c2: talking on the phone - right
4. c3: texting - left
5. c4: talking on the phone - left
6. c5: operating the radio
7. c6: drinking
8. c7: reaching behind
9. c8: hair and makeup
10. c9: talking to passenger

Although all the images in the supplied training set are still images, it is possible to reconstruct the original "movies" based on their order in the CSV file supplied with ground truth annotations.

While classifying frames for driver monitoring is an interesting problem, which I would go on to develop techniques for later with the industry partner, I wanted to see if anything could be learned from the temporal information in the movies. Instead of using individual frames as required for the competition, short movie clips were used.

4.1.3 Research Questions

The two major research questions that formed the motivation for this work were:

1. Can temporal information be used to improve accurate classification of driver actions?
2. Can low-level information about temporal information from an unrelated problem be successfully used to better understand driver actions in videos?

To answer these questions, several networks based on CNNs and RNNs were used with and without augmentation.

To answer the second question, a transfer learning approach was used. Transfer learning is the process of transferring knowledge that has already been learned by one neural network into another one. This is often accomplished by copying the learned weights and biases from one or more layers of a fully trained network to a different network. Transfer learning can be used to overcome overfitting issues and to speed up the training process for a related task.

One important paper on the use of transfer learning with 3D CNNs was written by Tran et al. [75]. That paper makes a compelling case for the use of 3D CNNs for understanding video data. Their method, which they named C3D, compared favorably to other published results on 5 of 6 generic action datasets used. They also showed that their network learns information about both motion and appearance, first learning appearance and then motion. One problem with this design is that only relatively short action sequences (16 frames) can be learned. The best results were obtained using this approach, as explained further.

Another approach, which was new at the time this research was conducted, combines LSTMs with convolutions and is introduced by Xing et al in [76]. Although the focus of their paper is forecasting precipitation, their method is generally applicable to the task of gathering long and short term time information from video sequences. Several experiments involving this method were performed as explained later, but none of them performed well, likely because of the lack of data.

Addressing the problem of insufficient data to train a neural network, [77] introduced a large, automatically generated, database gathered from YouTube clips called the sport 1M dataset. They showed that a transfer learning approach is effective at gaining accuracy on UCF 101 when a network is first trained on sports 1M. They evaluate their method on the UCF - 101, a database that contains over 12,000 videos with 101 human action classes [78].

4.1.4 Augmentation

Augmentation is a recurring theme in this thesis and the driver monitoring study was no exception.

In experiments where augmentation was applied, the ImageDataGenerator class within Keras was used. This class is used to dynamically create augmented images during training given a set of parameters. Since the standard implementation of ImageDataGenerator only supports 2D data, so it was extended to properly apply the transformations to video data. This modification involved ensuring that the same transformation was applied to every frame of a clip instead of treating each frame as an individual image with a potentially different transformation. Transformations included rotation (random 5 and 15 degrees left and right), and translation (up to 10% on width and height).

4.1.5 Experiments

In this section, experiments on the distracted driver database are summarized. The experiments were designed to allow comparison between networks that use temporal information (LSTM, 3D CNN, etc) and networks that ignore it (2D CNN). The last experiments are designed to measure the improvement that is achieved by transfer learning.

For a proper comparison, the single-frame-based method chosen was a full VGG16, trained [67] from scratch on the distracted driver dataset in Keras with a learning rate of 0.001. The experiment was then repeated on a grayscale version of the distracted driver dataset with rotation, translation, and feature normalization (inputs are divided by the standard deviation of the dataset).

Experiments Utilizing Temporal Data

Once reliable baseline for what a frame-based method could accomplish had been established, experiments involving temporal information were performed. The LSTM used was the implementation from [76], implemented in Keras as ConvLSTM2D.

Network 1 was a simple 3D CNN, network 2 was the same 3D CNN, but with a LSTM step before. Network 3 was the same as network 2, except with the order reversed (3DCNN followed by LSTM). Further details on these architectures and the training methodology used are in Appendix C. All three experiments were performed on both visible and grayscale versions of the dataset. Unfortunately, these networks all quickly overfit and didn't produce satisfactory results.

This overfitting was expected due to the limited number of subjects and the highly correlated frames. Investigation of the source of the overfitting revealed that low level features (in the first few layers) were responsible.

Transfer Learning

In this experiment, a C3D was trained with random weight initialization on the distracted driver dataset. This was compared with a C3D network that had been pretrained [75] on the sports 1-M dataset.

Transfer Learning with C3D

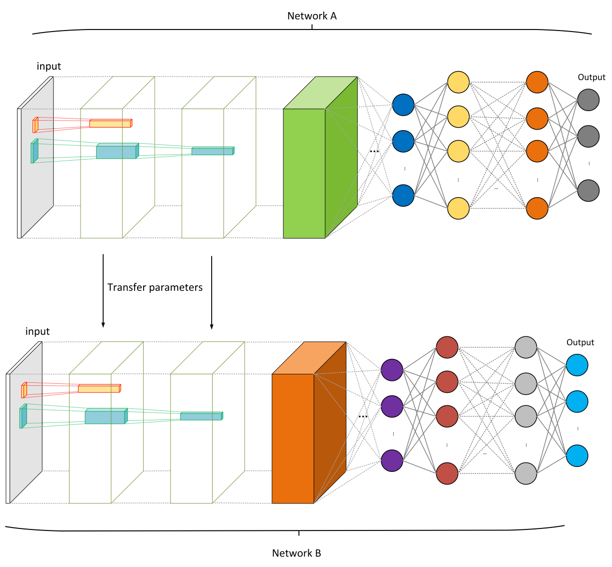


Fig. 4.1 Illustration of transfer learning concept where the first layers in network A and network B are the same.

Since other approaches to reducing the overfitting problem, were of limited success, a transfer learning approach was tried. The idea is to use pretrained weights from an existing network, trained for a more generic action recognition task, and then to tune them with the Distracted Driver training set.

In the previous experiments, the first layers were identified as being the primary source of overfitting, thus two transfer learning approaches were attempted. The first was to train the pretrained C3D network with a very low learning rate of 0.0001 without freezing any layers.

The alternate transfer learning approach wherein freezing the learning rate of the first layers was attempted. Since the first layers were identified as the cause of the greatest source

Table 4.1 Top Performing Approaches ($>=30\%$)

Approach	Accuracy
Transfer learning on 3D CNN. First 2 layers frozen	73.35%
Transfer learning on 3D CNN. First 5 layers frozen	60%
2D VGG 16 with augmentation	46%
3D CNN without augmentation	39.57 (no transfer learning)%
2D VGG 16 without augmentation	30%

Table 4.2 The Best Results on the Validation Set After Reaching 100% on the Training Set

Train loss*	Val loss	Train Accuracy	Val Accuracy
0.0173	0.8563	1	0.7335

of overfitting, this experiment was repeated again, except freezing only the first five layers, followed by freezing the first two layers.

4.1.6 Summary of Findings

In this subsection, the experiments in the previous section are summarized. Detailed results are available in appendix C. The experiments with the best results are listed in table 4.1. Since overfitting is found to be the primary cause of validation error in most experiments, details about the loss and accuracy are shown in table 4.2 before and after 100% accuracy was obtained on the training set (indicating overfitting).

In this research, it was shown that low level filters (early layers) learned by a 3D CNN can be used to greatly increase the accuracy on small datasets of drivers for the driver behavior classification task. It is not obvious that the first layers of a network trained for identifying actions in sports videos, such as basketball and swimming, could also be used to distinguish between distracted driver actions like left and right hand cell phone use or speaking with a passenger. It was also shown that temporal information could be used to increase accuracy for the driver behavior monitoring task over a network that does not use such temporal information. At a very low level, the action of moving fingers and heads may not be substantially different between different action recognition problems for convolutional neural networks. In these experiments, freezing any more than the first two layers decreased accuracy.

4.2 Semi-Parallel Deep Neural Network (SPDNN) Hybrid Architecture, First Application on Depth from Monocular Camera

SPDNN is an algorithm to merge different types of neural networks together in such a way as to benefit from each without duplication in computation.

As stated previously in this thesis, Deep Neural Networks (DNN) have been used in a range of machine learning and data-mining applications. These networks comprise sequential layers including, for example, convolutional layers or fully connected layers, typically accompanied by pooling or regularization tasks.

At times, one may encounter two or more networks that perform well at some aspects of a given task which, when taken together, perform better than they would individually. A common approach in this case is to use an ensemble or other method of merging the output of these networks. This section suggests an alternative approach called Semi-Parallel Deep Neural Network (SPDNN).

The main idea of SPDNN is to merge two or more neural networks without altering the kernel sizes or the order of layers. This is accomplished by representing each layer of each network as nodes in a graph, labeling the nodes and applying graph contraction. This will be explained in detail in the following sections. Complete details of SPDNN can be found in Appendix F.

Merging components of specialized deep neural networks was producing better results than not combining them, but some critics wondered what the difference between this approach and inception or even ensembling was.

Of course: it was not any of these methods ensambling keeps the existing network architectures and there are no inception modules in SPDNN (although inception type networks can be merged using the SPDNN technique). The major problem was that that there was no algorithm that could describe exactly what SPDNN was. A general method for generating any SPDNN-like network was elusive. At some point it became clear that this was a graph contraction problem. If each neural network layer is uniquely labeled and then represented as a graph, then it would then be possible to use graph contraction to come up with the exact same networks which had worked well in experiments. At that point, from then on, SPDNN changed from being an observation to a graph based algorithm.

SPDNN has been applied successfully to depth estimation[15] and iris segmentation [79].

The journal paper on SPDNN is included in appendix F and a detailed discussion of the graph-based algorithm is described in section 4.2.

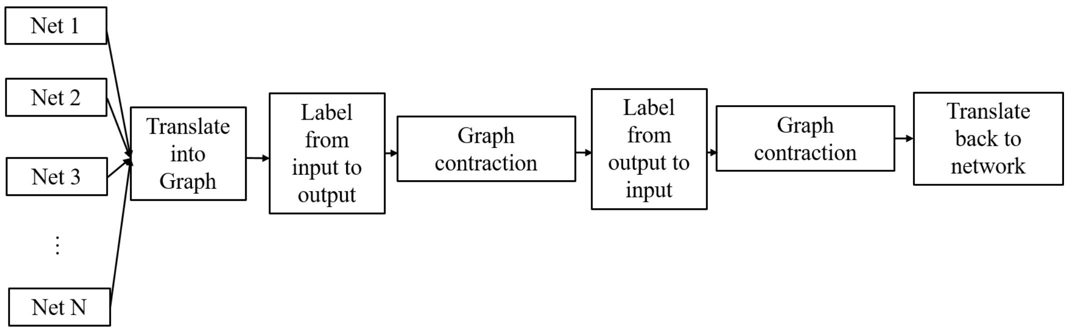

Net 1
Net 2
Net 3
Net N
Translate into Graph
Label from input to output
Graph contraction
Label from output to input
Graph contraction
Translate back to network

Fig. 4.2 High level overview showing labeling and contraction steps of SPDNN workflow.

4.2.1 How to Construct a Semi-Parallel Deep Neural Network with Graph Contraction and Labeling

As my contribution to SPDNN was entirely in the graph algorithm, I omit details on applications of SPDNN, as I was not involved in these aspects. This subsection explains how to create an SPDNN type network from multiple existing network typologies.

A high level overview showing the main steps of the SPDNN algorithm can be seen in figure 4.2. To convert two or more neural networks with the same output and input type into a graph as part of the SPDNN process, the following steps should be applied.

First, arrange each layer as a node in a graph, connecting nodes according to how they were connected in the original network. This involves labeling each node according to its properties. In the case of convolutional layers, this would be: kernel size, layer type, and distance from input. For example (5c,3) would be a convolutional layer with a 5x5 kernel at distance 3 from the input.

In the case of fully connected layers, there is no kernel size, but the number of neurons is often important so the syntax is: number of neurons, followed by the layer type and then the distance from input. For example (4f,7) would be a fully connected layer with 4 neurons at distance 7 from the input.

Pooling and unpooling layers are represented by U and P symbols with their kernel size in front. For example 2p means a 2x2 pooling layer. Pooling and unpooling operations are never represented as their own nodes in the graph but are instead appended to the previous layer. They also have a "stickiness" property, which means every node after the first keeps the pooling or unpooling property. Fully connected layers remove this property and later pooling or unpooling layers modify it. This mechanism is best illustrated by figure 4.3, where nodes labeled with C are (3c2p,2) indicating a 3x3 convolutional layer with 2x2 max pooling applied and distance 2 from the input. Also note that this property is retained until N in the

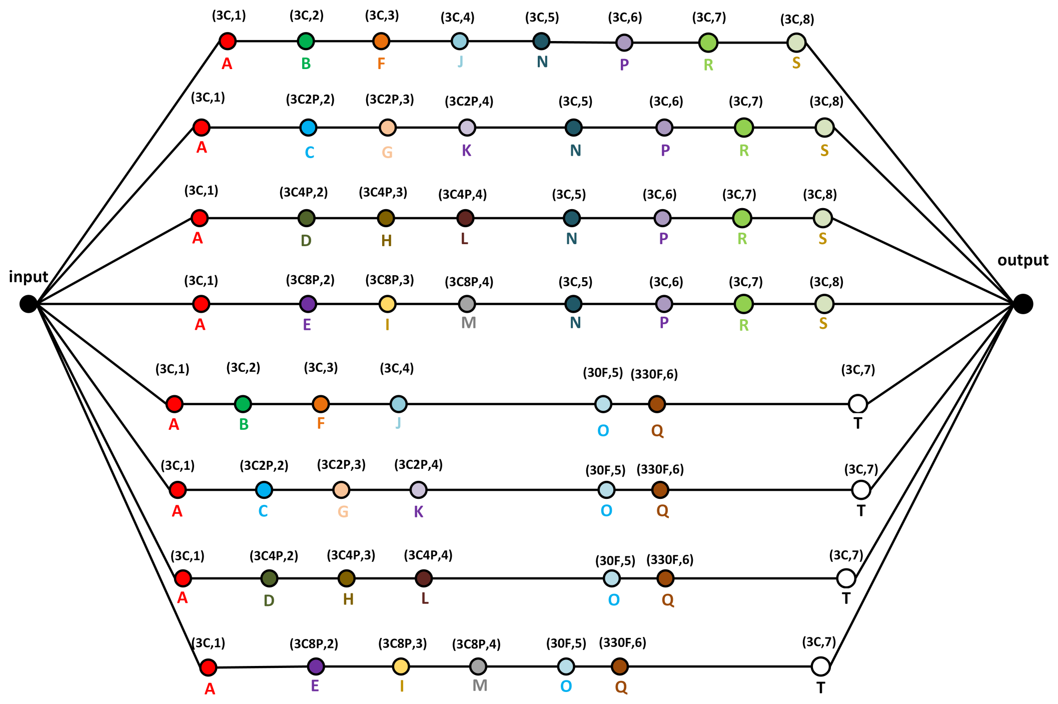


Fig. 4.3 This figure illustrates a parallelized version of 8 networks before the graph contraction step. Note that each note is labeled according to the methods described previously and there is a single input and output.

same figure. This was caused by an unpooling layer of the same kernel size removing the convolution property.

Finally, nodes with the same properties get assigned the same labels (Shown in figure 4.3). For example every (3c2p,2) will be assigned a label (perhaps C). This allows us to complete the graph contraction step shown in figure 4.4 wherein all labels that are the same are merged together.

After this point a new SPDNN type network with the same properties as the original networks can be created by following the labeling process in reverse order.

There is one final issue that remains ambiguous in the above explanations, and that is what to do with convolutional layers with kernels that are the same size and that have a different numbers of outputs. These layers should be merged but the number of outputs may be chosen according to the judgment of the engineer. In case of uncertainty or if designing an automated version of SPDNN, my suggestion is to select the number of outputs based on the maximum outputs of the merged nodes.

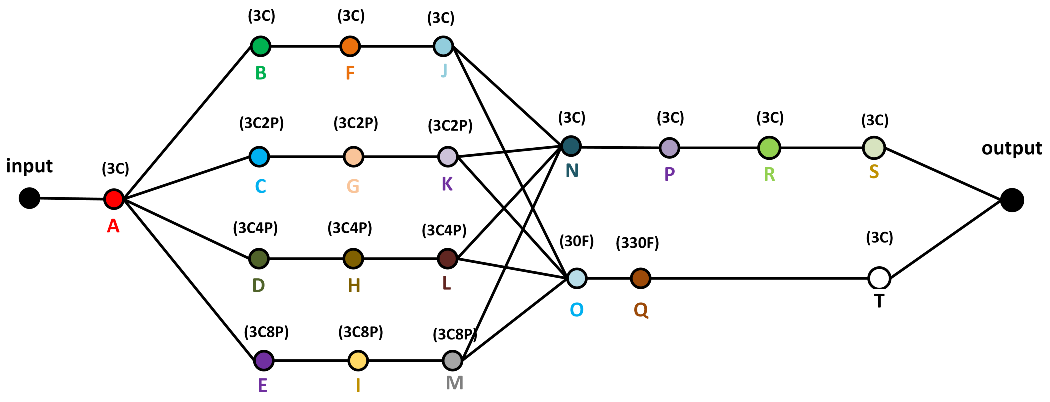


Fig. 4.4 This is the same network as shown in 4.3 after graph contraction is applied.

References

[1] D. H. Wolpert, W. G. Macready *et al.*, “No free lunch theorems for optimization,” *IEEE transactions on evolutionary computation*, vol. 1, no. 1, pp. 67–82, 1997.

[2] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[3] D. E. Rumelhart, G. E. Hintont, and R. J. Williams, “Learning representations by back-propagating errors,” *NATURE*, vol. 323, p. 9, 1986.

[4] A. M. Turing, “Intelligent machinery,” 1948.

[5] J. Lemley, S. Bazrafkan, and P. Corcoran, “Deep learning for consumer devices and services: Pushing the limits for machine learning, artificial intelligence, and computer vision.” *IEEE Consumer Electronics Magazine*, vol. 6, no. 2, pp. 48–56, 2017.

[6] J. McCarthy, M. Minsky, and N. Rochester, “A proposal for the dartmouth summer research project on artificial intelligence,” *Reprinted online at http://www-formal. stanford. edu/jmc/history/dartmouth/dartmouth. html*, 1955.

[7] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[8] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[9] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, “Backpropagation applied to handwritten zip code recognition,” *Neural Comput*, vol. 1, no. 4, pp. 541–551, 1989.

[10] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1.

[11] J. Lemley, S. Bazrafkan, and P. Corcoran, “Smart augmentation learning an optimal data augmentation strategy,” *IEEE Access*, vol. 5, pp. 5858–5869, 2017.

[12] ——, “Learning data augmentation for consumer devices and services,” in *2018 IEEE International Conference on Consumer Electronics (ICCE)*. IEEE, 2018, pp. 1–3.

[13] P. Corcoran, J. Lemley, and S. Bazrafkkan, “Getting more from your datasets: Data augmentation, annotation and generative techniques,” *Embedded Vision Summit 2018*, 2018.

[14] J. Lemley, A. Kar, A. Drimbarean, and P. Corcoran, “Convolutional neural network implementation for eye-gaze estimation on low-quality consumer imaging systems,” *IEEE Transactions on Consumer Electronics*, 2019.

[15] S. Bazrafkan, H. Javidnia, J. Lemley, and P. Corcoran, “Semiparallel deep neural network hybrid architecture: first application on depth from monocular camera,” *Journal of Electronic Imaging*, vol. 27, no. 4, p. 043041, 2018.

[16] J. Lemley, B. Shabab, and P. Corcoran, “Transfer learning of temporal information for driver action classification,” in *Proceedings of the 28th Modern Artificial Intelligence and Cognitive Science Conference 2017*, vol. 1964, 2017, pp. 123–128.

[17] J. Lemley, A. Kar, and P. Corcoran, “Eye tracking in augmented spaces: A deep learning approach,” in *2018 IEEE Games, Entertainment, Media Conference (GEM).* IEEE, 2018, pp. 1–6.

[18] A. McDonagh, J. Lemley, R. Cassidy, and P. Corcoran, “Synthesizing game audio using deep neural networks,” in *2018 IEEE Games, Entertainment, Media Conference (GEM).* IEEE, 2018, pp. 1–9.

[19] S. Bazrafkan and J. Lemley, “Method for synthesizing a neural network,” Jul. 26 2018, US Patent App. 15/413,283.

[20] ——, “Method of training a neural network,” Jul. 26 2018, US Patent App. 15/413,312.

[21] A. Brand, L. Allen, M. Altman, M. Hlava, and J. Scott, “Beyond authorship: attribution, contribution, collaboration, and credit,” *Learned Publishing*, vol. 28, no. 2, pp. 151–155, 2015.

[22] L. Sirovich and M. Kirby, “Low-dimensional procedure for the characterization of human faces,” *Josa a*, vol. 4, no. 3, pp. 519–524, 1987.

[23] A. Martinez and R. Benavente, “The ar face database,” CVC Technical Report #24, Tech. Rep., 1998.

[24] P. J. Phillips, H. Moon, S. A. Rizvi, and P. J. Rauss, “The feret evaluation methodology for face-recognition algorithms,” *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 10, pp. 1090–1104, 2000.

[25] E. Eidinger, R. Enbar, and T. Hassner, “Age and gender estimation of unfiltered faces,” *Information Forensics and Security, IEEE Transactions on*, vol. 9, no. 12, pp. 2170–2179, 2014.

[26] A. Hernández-García and P. König, “Data augmentation instead of explicit regularization,” *arXiv preprint arXiv:1806.03852*, 2018.

[27] ——, “Further advantages of data augmentation on convolutional neural networks,” in *International Conference on Artificial Neural Networks.* Springer, 2018, pp. 95–103.

[28] S. Hauberg, O. Freifeld, A. B. L. Larsen, J. Fisher, and L. Hansen, “Dreaming more data: Class-dependent distributions over diffeomorphisms for learned data augmentation,” in *Artificial Intelligence and Statistics*, 2016, pp. 342–350.

[29] L. Perez and J. Wang, “The effectiveness of data augmentation in image classification using deep learning,” *arXiv preprint arXiv:1712.04621*, 2017.

[30] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le, “Autoaugment: Learning augmentation strategies from data,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 113–123.

[31] C. Lin, M. Guo, C. Li, W. Wu, D. Lin, W. Ouyang, and J. Yan, “Online hyper-parameter learning for auto-augmentation strategy,” *arXiv preprint arXiv:1905.07373*, 2019.

[32] S. Lim, I. Kim, T. Kim, C. Kim, and S. Kim, “Fast autoaugment,” *arXiv preprint arXiv:1905.00397*, 2019.

[33] D. Ho, E. Liang, I. Stoica, P. Abbeel, and X. Chen, “Population based augmentation: Efficient learning of augmentation policy schedules,” *arXiv preprint arXiv:1905.05393*, 2019.

[34] K. Fujita, M. Kobayashi, and T. Nagao, “Data augmentation using evolutionary image processing,” in *2018 Digital Image Computing: Techniques and Applications (DICTA)*. IEEE, 2018, pp. 1–6.

[35] J. C. Acero, E. Zacur, H. Xu, R. Ariga, A. Bueno-Orovio, P. Lamata, and V. Grau, “Smod-data augmentation based on statistical models of deformation to enhance segmentation in 2d cine cardiac mri,” in *International Conference on Functional Imaging and Modeling of the Heart*. Springer, 2019, pp. 361–369.

[36] X. Peng, Z. Tang, F. Yang, R. S. Feris, and D. Metaxas, “Jointly optimize data augmentation and network training: Adversarial data augmentation in human pose estimation,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[37] A. J. Ratner, H. Ehrenberg, Z. Hussain, J. Dunnmon, and C. Ré, “Learning to compose domain-specific transformations for data augmentation,” in *Advances in neural information processing systems*, 2017, pp. 3236–3246.

[38] M. Podduturi, “Data augmentation for supervised learning with generative adversarial networks,” *Iowa State University Digital Repository GRADUATE THESES AND DISSERTATIONS*, 2018.

[39] A. Antoniou, A. Storkey, and H. Edwards, “Data augmentation generative adversarial networks,” *arXiv preprint arXiv:1711.04340*, 2017.

[40] X. Zhang, Z. Wang, D. Liu, and Q. Ling, “Dada: Deep adversarial data augmentation for extremely low data regime classification,” in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 2807–2811.

[41] T. Tran, T. Pham, G. Carneiro, L. Palmer, and I. Reid, “A bayesian data augmentation approach for learning deep models,” in *Advances in Neural Information Processing Systems*, 2017, pp. 2797–2806.

[42] J. Schlüter and T. Grill, “Exploring data augmentation for improved singing voice detection with neural networks.” in *ISMIR*, 2015, pp. 121–126.

[43] D. S. Park, W. Chan, Y. Zhang, C.-C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le, “Specaugment: A simple data augmentation method for automatic speech recognition,” *arXiv preprint arXiv:1904.08779*, 2019.

[44] I. Vatolkin and D. Stoller, “Evolutionary multi-objective training set selection of data instances and augmentations for vocal detection,” in *International Conference on Computational Intelligence in Music, Sound, Art and Design (Part of EvoStar)*. Springer, 2019, pp. 201–216.

[45] T. DeVries and G. W. Taylor, “Improved regularization of convolutional neural networks with cutout,” *arXiv preprint arXiv:1708.04552*, 2017.

[46] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, “mixup: Beyond empirical risk minimization,” *arXiv preprint arXiv:1710.09412*, 2017.

[47] H. Guo, Y. Mao, and R. Zhang, “Mixup as locally linear out-of-manifold regularization,” *arXiv preprint arXiv:1809.02499*, 2018.

[48] Y. Duan, X. Niu, and G. Nie, “Data augmentation based on interest points of feature,” in *Tenth International Conference on Digital Image Processing (ICDIP 2018)*, vol. 10806. International Society for Optics and Photonics, 2018, p. 108060B.

[49] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[50] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, “Improved techniques for training gans,” in *Advances in neural information processing systems*, 2016, pp. 2234–2242.

[51] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, “Gans trained by a two time-scale update rule converge to a local nash equilibrium,” in *Advances in Neural Information Processing Systems*, 2017, pp. 6626–6637.

[52] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, “Unpaired image-to-image translation using cycle-consistent adversarial networks,” in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2223–2232.

[53] D. Berthelot, T. Schumm, and L. Metz, “Began: Boundary equilibrium generative adversarial networks,” *arXiv preprint arXiv:1703.10717*, 2017.

[54] A. Mims, “Student neural network,” Jun. 13 2006, US Patent 7,062,476.

[55] W. Kinzel and I. Kanter, “Neural cryptography,” in *Proceedings of the 9th International Conference on Neural Information Processing, 2002. ICONIP’02.*, vol. 3. IEEE, 2002, pp. 1351–1354.

[56] Y. Liang, M. L. Reyes, and J. D. Lee, “Real-time detection of driver cognitive distraction using support vector machines,” *IEEE Trans. Intell. Transp. Syst.*, vol. 8, no. 2, pp. 340–350, June 2007.

[57] E. Wood and A. Bulling, “Eyetab: Model-based gaze estimation on unmodified tablet computers,” in *Proceedings of the Symposium on Eye Tracking Research and Applications*, ser. ETRA ’14. New York, NY, USA: ACM, 2014, pp. 207–210. [Online]. Available: http://doi.acm.org/10.1145/2578153.2578185

[58] A. Tawari and M. M. Trivedi, “Robust and continuous estimation of driver gaze zone by dynamic analysis of multiple face videos,” in *2014 IEEE Intelligent Vehicles Symposium Proceedings*, June 2014, pp. 344–349.

[59] V. Vaitukaitis and A. Bulling, “Eye gesture recognition on portable devices,” in *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, ser. UbiComp ’12. New York, NY, USA: ACM, 2012, pp. 711–714. [Online]. Available: http://doi.acm.org/10.1145/2370216.2370370

[60] A. Kar and P. Corcoran, “A review and analysis of eye-gaze estimation systems, algorithms and performance evaluation methods in consumer platforms,” *IEEE Access*, vol. 5, pp. 16 495–16 519, 2017.

[61] D. W. Hansen and Q. Ji, “In the eye of the beholder: A survey of models for eyes and gaze,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 3, pp. 478–500, 2010.

[62] F. L. Coutinho and C. H. Morimoto, “Augmenting the robustness of cross-ratio gaze tracking methods to head movement,” in *Proceedings of the Symposium on Eye Tracking Research and Applications*. ACM, 2012, pp. 59–66.

[63] S. Chen and C. Liu, “Eye detection using discriminatory haar features and a new efficient svm,” *Image Vision Comput.*, vol. 33, pp. 68–77, 2015.

[64] H. chuan Lu, C. Wang, and Y. w. Chen, “Gaze tracking by binocular vision and lbp features,” in *2008 19th International Conference on Pattern Recognition*, Dec 2008, pp. 1–4.

[65] J. Lemley, S. Bazrafkan, and P. Corcoran, “Smart augmentation learning an optimal data augmentation strategy,” *IEEE Access*, vol. 5, pp. 5858–5869, 2017. [Online]. Available: https://doi.org/10.1109/ACCESS.2017.2696121

[66] X. Zhang, Y. Sugano, M. Fritz, and A. Bulling, “Appearance-based gaze estimation in the wild,” in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015, pp. 4511–4520.

[67] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.

[68] X. Zhang, Y. Sugano, M. Fritz, and A. Bulling, “It’s written all over your face: Full-face appearance-based gaze estimation,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, July 2017, pp. 2299–2308.

[69] T. Baltrusaitis, P. Robinson, and L. P. Morency, “Openface: An open source facial behavior analysis toolkit,” in *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, March 2016, pp. 1–10.

[70] E. Wood, T. Baltrušaitis, L.-P. Morency, P. Robinson, and A. Bulling, “Learning an appearance-based gaze estimator from one million synthesised images,” in *Proceedings of the Ninth Biennial ACM Symposium on Eye Tracking Research & Applications*, ser. ETRA ’16. New York, NY, USA: ACM, 2016, pp. 131–138. [Online]. Available: http://doi.acm.org/10.1145/2857491.2857492

[71] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb, “Learning from simulated and unsupervised images through adversarial training,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017, pp. 2242–2251.

[72] S. Nie, M. Zheng, and Q. Ji, “The deep regression bayesian network and its applications: Probabilistic deep learning for computer vision,” *IEEE Signal. Proc. Mag.*, vol. 35, no. 1, pp. 101–111, Jan 2018.

[73] C. D. McMurrough, V. Metsis, J. Rich, and F. Makedon, “An eye tracking dataset for point of gaze detection,” in *Proceedings of the Symposium on Eye Tracking Research and Applications*. ACM, 2012, pp. 305–308.

[74] F. Chollet, “Xception: Deep learning with depthwise separable convolutions,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1251–1258.

[75] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, “Learning spatiotemporal features with 3d convolutional networks,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 4489–4497.

[76] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, “Convolutional lstm network: A machine learning approach for precipitation nowcasting,” in *Advances in Neural Information Processing Systems*, 2015, pp. 802–810.

[77] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, “Large-scale video classification with convolutional neural networks,” in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2014, pp. 1725–1732.

[78] K. Soomro, A. R. Zamir, and M. Shah, “Ucf101: A dataset of 101 human actions classes from videos in the wild,” *arXiv preprint arXiv:1212.0402*, 2012.

[79] S. Bazrafkan, S. Thavalengal, and P. Corcoran, “An end to end deep neural network for iris segmentation in unconstrained scenarios,” *Neural Networks*, vol. 106, pp. 79–95, 2018.

[80] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, “Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size,” *arXiv preprint arXiv:1602.07360*, 2016.

Appendix A

Deep Learning for Consumer Devices and Services

Deep Learning for Improved Consumer Devices & Services

1. INTRODUCTION TO DEEP LEARNING 1

1.1 DEEP LEARNING & CONSUMER ELECTRONICS 2
1.2 NEURAL NETWORKS - WHAT THEY ARE AND WHAT THEY ARE USED FOR 2
1.3 TRAINING THE NETWORK 2
SUPERVISED VS UNSUPERVISED LEARNING 3
1.4 INSIDE THE NEURAL NETWORK 4
1.5 A RENAISSANCE FOR NEURAL NETWORKS? 5

2. CONVOLUTIONAL NEURAL NETWORKS 6

2.1 CONVOLUTION AND ITS ROLE IN A NEURAL NETWORK: 6
2.2 DEEP NEURAL NETWORKS 9
2.2.1 TYPICAL DEEP NEURAL NETWORKS: 9
2.2.2 FULLY CONVOLUTIONAL NETWORKS (FCN): 10
2.2.2 AUTO-ENCODERS: 11
2.2.4 RECURRENT NEURAL NETWORKS (RNN): 12

3. STATE OF ART TODAY 12

3.1 MAIN ENABLING TECHNOLOGIES (OVERVIEW) 12
3.2 EXAMPLE APPLICATIONS 14

4. CONCLUSIONS 15

REFERENCES 15

1. Introduction to Deep Learning

In the last few years we have witnessed an exponential growth in research activity into the advanced training of convolutional neural networks (CNN) – a field which has become known as "Deep Learning". This has been triggered by a combination of the availability of significantly larger data-sets, thanks in part to a corresponding growth in "Big Data", and the arrival of new GPU-based hardware that enables these large data-sets to be processed in reasonable time-scales. Suddenly a myriad of long-standing problems in machine learning, artificial intelligence and computer vision have seen significant improvements, often sufficient to break through long-standing performance thresholds. Across multiple fields these achievements have inspired the development of improved tools and methodologies leading to even broader applicability of Deep Learning. The new generation of smart assistants – Alexa, Hello Google and others – have their roots and learning algorithms tied into deep learning. In this article we review the current state of Deep Learning, explaining what it is, why it has managed to improve on the longstanding techniques of conventional neural networks and, most importantly, how you can get started with adopting deep learning into your own research activities to solve both new and old problems and build better, smarter consumer devices & services.

1.1 Deep Learning & Consumer Electronics

There has perhaps never been a better time to take advantage of the power of deep learning in consumer products. In retrospect, we may consider 2016 the year that deep learning toolkits and techniques matured from tools that were mostly oriented to researchers into easily used product-enabling technology that can be used to add "intelligence" to almost any consumer device even by non-experts. We expect to see an explosion in products that take advantage of these resources in the coming years, with early adopters differentiating themselves from competitors and further refinement of technology and deep learning methods.

In this article, we hope to provide you with the tools and understanding to start using deep learning today, or to understand what consumer devices that are built using this technology can do and how they work.

1.2 Neural Networks - what they are and what they are used for

Artificial neural networks (ANN) are able to learn something about what they "see" and then "**generalize**" that knowledge to examples (or samples) that they have never seen before [1]. This is a very powerful capability that humans often take for granted because our brains do it so well automatically. You are able to understand the concept of a rock after seeing and perhaps touching very few examples of rocks. From that point on you can identify any rock, even those that are shaped differently or have different colours or textures from the rocks you've seen before. This approach can be seen as opposed to the traditional method of "teaching" or explicitly programming computers based on detailed "rules" that must cover every possible outcome.

The process of discerning the category to which a piece of data belongs is called a **classification task**; one of the more famous uses of this technique is that of training a neural network. The ability to classify unseen examples is referred to as "**generalization**".

Not surprisingly, Artificial Neural Networks are especially powerful in tasks for which the appropriate outcome cannot be determined beforehand and thus cannot use traditional pre-programmed rules [2].

1.3 Training the Network

Artificial Neural Networks (ANN) are not the only techniques that can do this, and much of the terminology we use when talking about ANNs comes from other fields such as statistics. The process of "teaching" our network is called "**training**". When we train a network, what we are formally doing is "**fitting**" a network to our **training** (data-)**set**. This language is borrowed from mathematics where we may try to find the best way to "**fit**" information to a regression line or other mathematical model. The "**training set**" is the sample information that we think is sufficiently representative that our network should be able to learn from it. The thing we want our network to learn to do is called the "**task**".

When training Artificial Neural Networks, we want the network to perform well at a given task on unseen information. When an ANN is not trained sufficiently to do this, we call it "**underfitting**". Which means that the network did not sufficiently learn the training set. The opposite of this is called **overfitting**, where the network learns the training set so well that it cannot effectively be applied to data that it has not seen before [3]. These concepts can be best understood by referring to Figure 1 which illustrates a simple 2D data-set. Note that in practice we deal with multi-dimensional data leading to far more complex fitting problems.

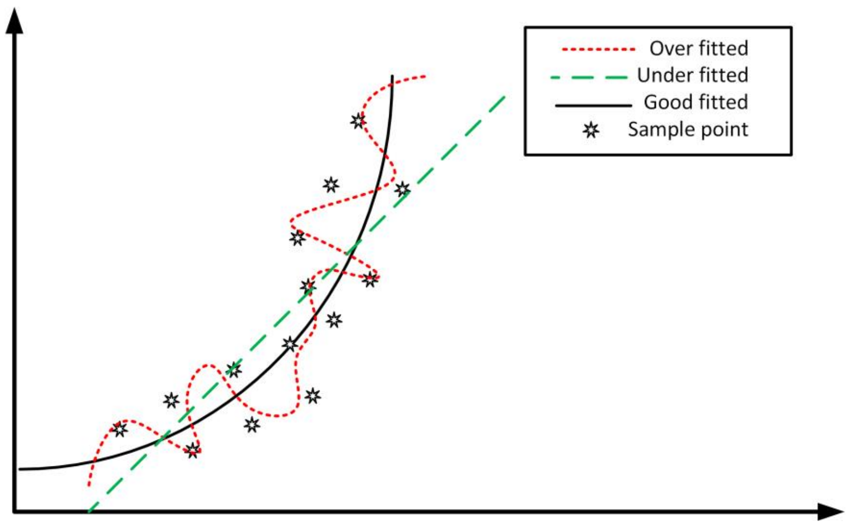


Figure 1- The red dashed line is an example of overfitting. It's not likely to work as well as the solid black line at predicting future values. The straight green dashed line is an example of under fitting. Choosing an overly complex model (such as a high degree polynomial that generates the red line leads to overfitting but picking a model that is too simple (the straight line) causes under fitting. Our goal is often to find the best model to fit the data.

If a model is underfitting, we can increase the number or size of parameters or improve the type of model. If a model is overfitting, we can reduce the size of the model or apply other techniques which we will describe later. A key challenge is how to accurately identify if the model is close to optimal fitting, and this is one of the reasons that very large data-sets are needed to achieve working solutions for these problems.

We measure the suitability of a model for a given task by withholding some of the information that we have from the training process so that we can evaluate the model during and after training. We typically separate this withheld information into two parts: the **validation set** and the **training set**. The information that we use to evaluate our model on during training is called the "**validation set**". This data is never used directly for training. It is only used to provide us with information about how well the network performs during the training process.

Supervised Vs Unsupervised Learning

There are two primary types of learning that neural networks can do: supervised learning, in which the data is labelled or annotated in some way and the task is to somehow learn to match the data with the labels, and unsupervised learning where there are no labels and the neural network learns to find relationships between data [4].

A common example of supervised learning is "classification" where we try to find a category (or label) that can successfully **discriminate** between each class using the supplied labels and data. An example of unsupervised learning is "**clustering**" where the neural network tries to separate data into "chunks" or groups without anyone giving the network any labels to apply to the groups it finds. The appeal of the latter approach is that creating "**ground truth**" labels is a time consuming and often expensive process which requires humans to create appropriate labels for the training, validation, and testing sets before a model can be trained and later placed into products in the real world with unseen (and unlabelled) data.

Despite the cost and difficulty in data preparation, much of the success of neural networks comes from supervised learning.

1.4 Inside the Neural Network

We've discussed what neural networks can do but we've not discussed the details of how they do it. We will now describe the details of how this works. Neural networks typically have an **input layer**, an **output layer**, and 1 or more so called "hidden layers".

These layers are full of nodes, often called neurons which are connected to subsequent and previous layers using a number of schemes.

Perhaps the most common connection scheme is the **fully connected** layer where every neuron in a layer is connected to every neuron in the previous and next layer.

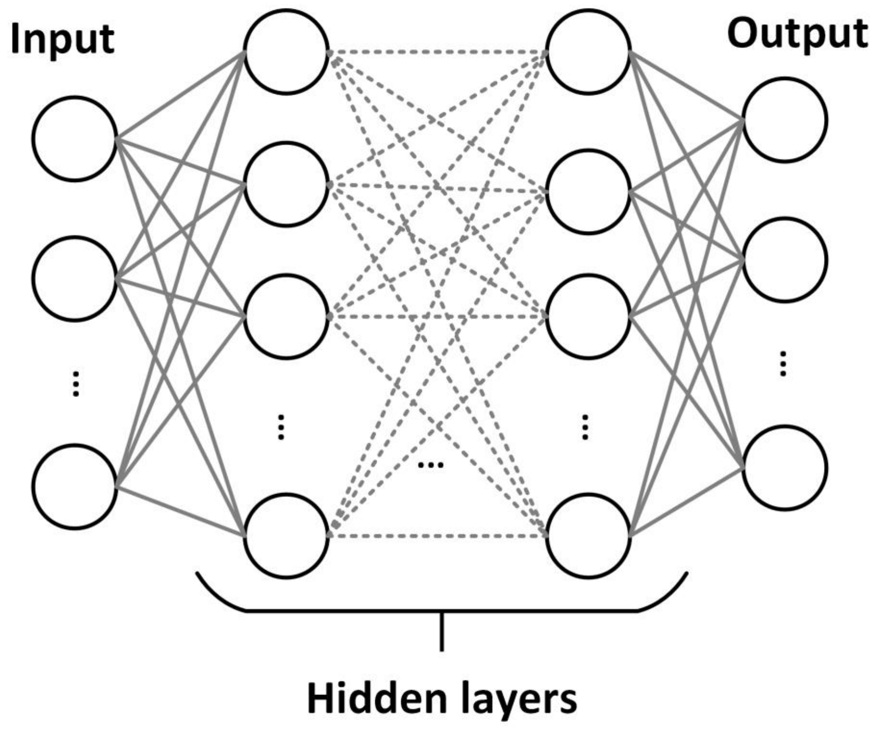
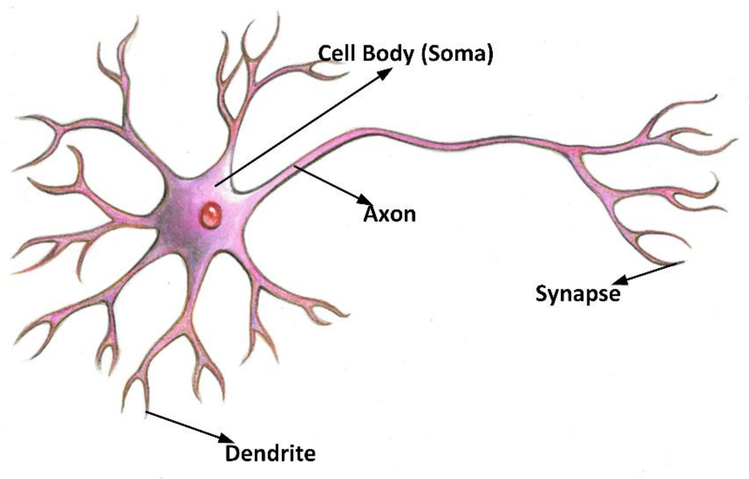


Figure 2 Typical Artificial Neural Network with a fully connected hidden layer.

This idea is inspired from biological neurons where we have axons and dendrites that connect individual neurons to each other. In the biology model axons receive input from other neurons and dendrites transmit information to other cells [5]. This corresponds to the input and output connections in a neural network. The concept of multiple connection schemes also comes from biology where we see Unipolar, Biploar, Multipolar, and Pseudounipolar connection mechanisms.

The body of a biological neuron is called the Soma which can decide to when and what to transmit based on various criteria. Artificial neurons have a similar mechanism called the "activation function".

This model of neurons was first invented in 1943 by Waren McCulloch and Walter Pitts [6]. The first popular implementation of these on computers was formalized in the idea of the Perceptron in 1957 by Frank Rosenblatt [7].



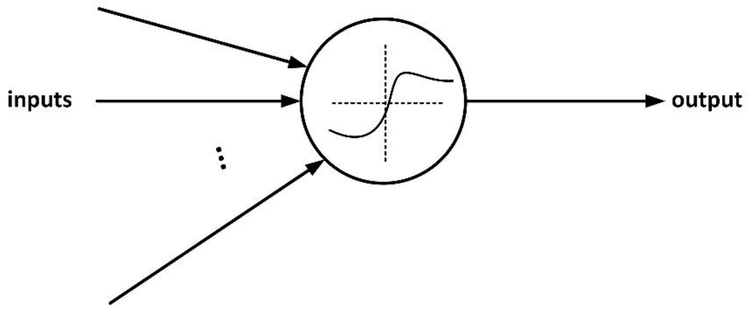


Figure 3: Diagram of biological neuron (Illustrated by Kimberly Sowell, permission obtained) above diagram of artificial neuron.

All artificial neural networks have **layers**, **weights**, **inputs**, **biases** (or thresholds), **activation functions** and some connection mechanism. But this is not enough to effectively train a neural network. When training a neural network we calculate an error (or loss). There are many ways to calculate the loss, but the simplest way is the difference between the predicted (learned) and the true outputs.

Training algorithms work by updating the weights and measuring the way that the loss changes over time. This is usually accomplished by some form of **gradient descent** optimizer. It's possible to get "stuck" or quickly converge to a non-optimal solution when strictly following the steepest gradients, and because of this, we typically use some form of stochastic gradient descent, which is simply a stochastic (or slightly randomized) form of gradient descent.

Back propagation is used to allow us to "propagate" the error information from the last layer to the first layer to modify the weights, and is often used in a way that is synonymous with training. Methods that can be used to improve a networks results are called "**learning rules**".

1.5 A Renaissance for Neural Networks?

Although these methods have been successfully used for decades, they have seen a very recent resurgence as refinements upon existing techniques, combined with newer hardware, and the growth of "Big Data", have created an AI boom that shows no signs of slowing down.

The global market for smart machines is expected to grow to +15 billion by 2019, with an average annual growth rate of nearly 20% [8]. The set of techniques that have led to this growth has been coined “deep learning”.

In the following sections we will discuss some of the techniques that enable deep learning, why they work, and how they are used to make smart machines.

2. Convolutional Neural Networks

In order store and process analog signals (for example image, voice and biological signals), on a computer, one must first convert these inputs into a digital form in a process called Analog to Digital conversion (A2D). This transforms the information from a continuous space to a discrete space, losing some information in the process. Often that information isn’t critical to understanding the underlying analog signal but in some instances we could lose critical data – e.g. high frequency information.

In digital processing we often refer to a “piece” of data, such as a picture, as a *sample*. It is convenient, but computationally expensive, to represent these samples in a high dimensional space where each unit (or pixel in the case of images), is considered as being located on a specific axis with the range of possible values (for example 0-255 in the case of an 8 bit color-channel in an image’) being the size of that axis.

An image that is 100x100 would be represented as a vector that is a point in 10,000 dimensional space. We call this space the “feature space”.

Since even the most powerful computers can have trouble with such high dimensional space, we try to reduce the number of dimensions to just those that are critical to a task or to change the way these features are represented.

In the traditional pattern recognition approach, to perform a given task we separate our process into two steps, **Feature generation** and **Feature selection**. Feature generation generates new features from the pixel space. Feature selection reduces the dimensionality of the feature space.

Examples of feature generation include morphological, fourier and wavelet transforms which create more useful features for specific tasks and feature selection includes methods like Principal Component Analysis (PCA) and Linear Fisher Discriminant (LDA) [9].

There is a newer technique based on sparse mapping where instead of trying to reduce the dimensionally, we expand it with the goal of representing more abstract features. This is inspired by models of the visual cortex of animals [10]. Convolutional layers, a key component of deep learning, make use of this sparse mapping approach.

2.1 Convolution and its Role in a Neural Network:

One of the most novel and useful aspects of convolutional neural networks is that they can learn the filters that previously had to be custom designed by the researcher (a task that would often take years of trial & error). These convolutional layers are essential to this task in modern deep neural networks.

Convolutional layers make use of the convolution operator. The Convolution operator is used on two functions. One is the signal from the sample space and the other, called the filter, is applied to the sample. On a GPU, convolutions are implemented as matrix multiplications.

This operator has a long history in image processing applications and dates back to the time when digital image processing started. The Convolution operation can be discussed in both spatial and transform space. In the spatial space the convolution operator is the equivalent operation of correlation with the reversed filter, i.e., this operator calculates the similarity of the input function with the filter. For example, the edge detection and corner detection filters are using the similarity of the input image with a pre-defined filter mimicking the edge or corner shape. Looking at the function in the transform space the convolution is performing frequency filtering. For example, low pass or high pass filters have their equivalent spatial filters which could be applied to the image using convolution operations.

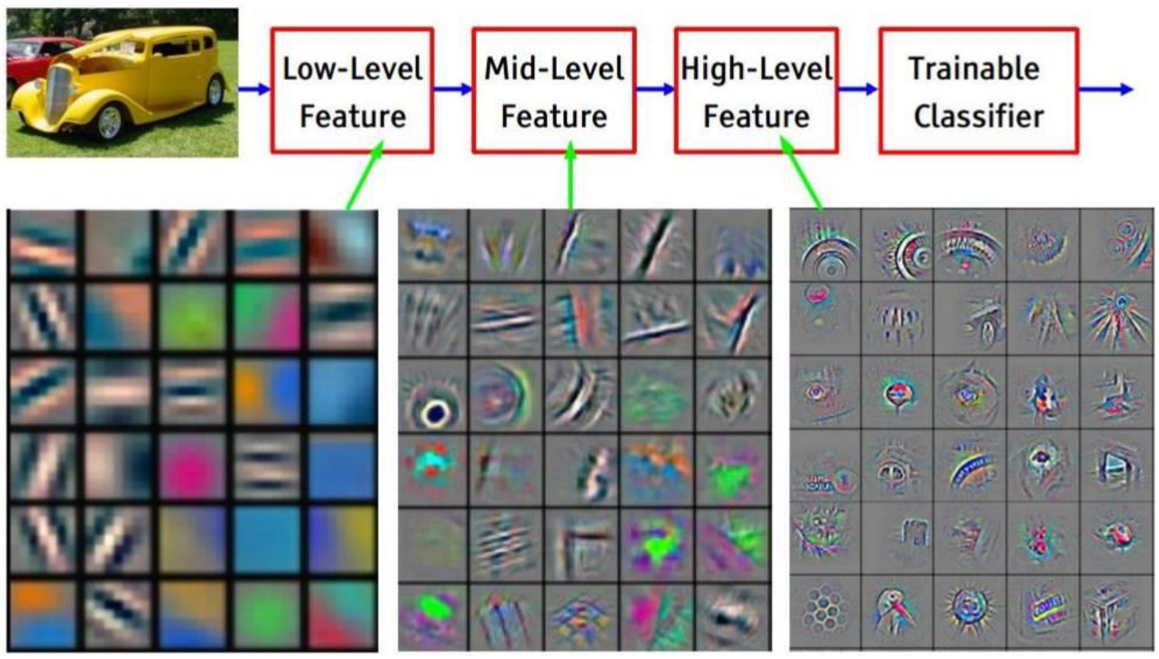


Figure 4 Visualization of the learned convolutional filters at different layers. Copyright Movidius, used with permission.

In the deep learning approach, the filter is learned and applied to the data during the training process with the hope that after training, the learned filter will be the best choice for the task. One difference between the way Convolutions are used in CNN's from more traditional uses is that the convolution operator is applied using a 4 dimensional filter. This is essentially a set of 3D filters that are stacked in the fourth dimension. We use the 4 dimensional filter to map a 3d space to another 3d space. See figure 5

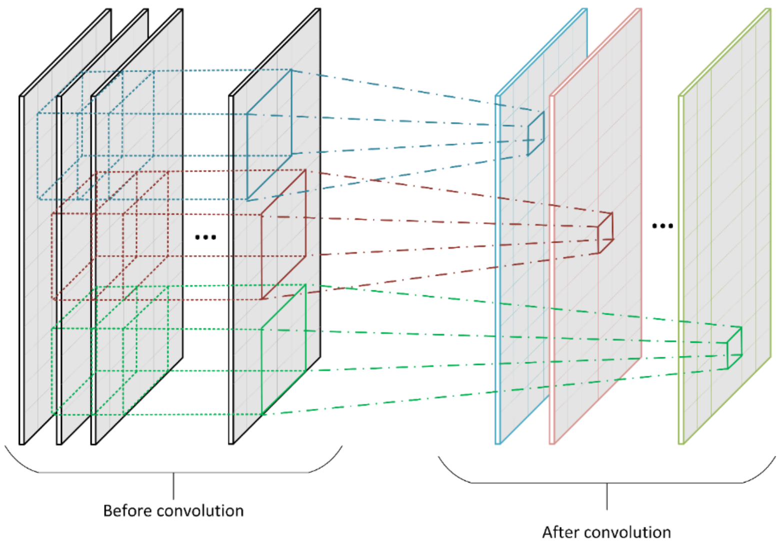


Figure 5. A 4 dimensional filter maps 3d space to another 3d space using convolutions.

Convolutional layer: in general for an n dimensional signal, the convolutional layer is an *n* or *n+1* dimensional feature space mapping with *n+1* or *n+2* dimensional kernels (filters). For example, given a 2 dimensional image, the convolutional layer would be 3 dimensional with a 4 dimensional kernel. In this case the 4 dimensions of the kernel are correspond to 1. width and 2. height of the input, 3. the number of channels of the input and 4. the number of channels of the output. In figure 5 you can see two different convolutional layers. The k channel layer on the left side is mapped to a p channel layer on the right side using a 4D kernel. This kernel is shown using different 3D kernels with different colors.

Pooling layer: pooling is an operation which accepts a pool of data values as input, and generates one value from them to be passed to the next layer. For example this operation could be mean or maximum of the input values. There are two important purposes for pooling operations. One is reducing the size of the data space to reduce overfitting and the other is transition invariance. A pooling layer is performing the pooling operation on its inputs. Figure 6 shows how a pooling operation is applied to a one channel input to reduce the dimensionality of the dataspace. In this figure we have a 3x3 pooling operation applied to a 12x6 one channel feature space. The most used pooling operation is max-pooling wherein the maximum value of the features in the pool is selected to be mapped to the next layer.

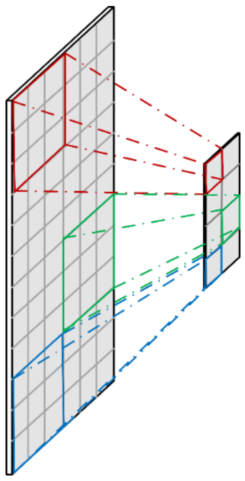


Figure 6.A pooling operation is reducing the size of the feature space.

The importance of this layer is described in the next example. In generic deep neural networks (described in 2.2.1 Generic Deep Neural Networks:)a dense, fully connected layer emerges from the convolutional layers. For example, consider a convolutional layer with 10 channels and a 100x100 feature space. Placing a fully connected layer after it would result in computing 100000 weights from this layer to each neuron in the dense layer which requires significant memory and computation resources. Using a pooling layer helps to reduce resource demands. Additionally, without a pooling layer, a network this big might suffer from over-fitting, especially if there is not enough representative data in the training set. Pooling also helps to provide transition invariance by helping each kernel to cover more space.

Nonlinearities: Each neuron in the deep neural network model is taking advantage of a non-linear activation function to calculate the output value. This would lead to one of the most advantageous properties of the deep neural networks which is their ability to describe highly non-linear systems. Being highly non-linear helps the model to be suitable for real-life problems and gives solutions in pattern recognition that cannot be achieved through more classical methods. In the early days of the neural networks the *tanh* and *sigmoid* activation functions were most popular. Realizing that most of the data tends to be concentrated around zero, newer techniques such as rectified linear units (RELU) and Exponential linear units (ELU) [11] become popular because they are nonlinear near zero. They also benefit from an infinite range.

2.2 Deep Neural Networks

The power of deep learning first made worldwide news in 2011, when a deep learning algorithm achieved better than human visual pattern recognition in an international competition. The accuracy was 6 times better than the nearest non neural network approach and twice as accurate as human experts [12].

In this section we discuss four of the better-known deep learning architectures that have made the most impact in recent research.

2.2.1 Generic Deep Neural Networks:

are usually made of one or more convolutional layers, wherein each convolutional layer is usually accompanied by a pooling (max-pooling) operation. One can also use bigger strides in the convolution layer in order to reduce the data dimensionality (the stride of a convolution operation is the number of the pixels the kernel window is sliding before calculating the convolution in each location).

In generic deep neural networks, the convolutional layers are usually followed by one or more dense fully connected layers. See figure 7. The rectified linear unit is the most common activation function used in these kind of networks. The last layer is typically taking advantage of other nonlinearities based on the task.

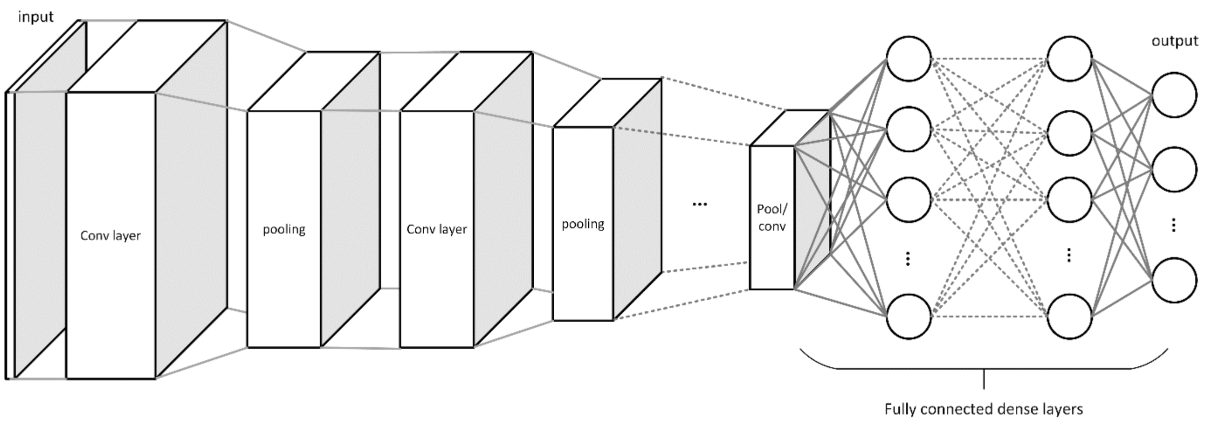


Figure 7: A generic deep neural network. Convolutional and pooling layers followed by fully conneceted dense layers.

2.2.2 Fully Convolutional Networks (FCN):

These are deep neural networks where all the layers are convolutional, pooling, and Un-pooling layers wherein the pooling and un-pooling layers are usually placed between two convolutional layers. They are similar to typical deep neural networks except they have no dense layer.

The output of a fully convolutional network is as the same type as its input. For example if the input is a k channel image the output of the network could be a p channel image but not something else entirely.

Un-pooling layers are designed to perform the inverse operation of pooling layers by increasing the size of the feature space. These layers operate on a single pixel and expand that pixel into a pool of data. There are different implementations of un-pooling layers. The most popular of which are repeating values and sparse un-pooling in DNN designs. The repeating value technique is expanding the data from one value to a pool of data with the same value (figure 8). With sparse un-pooling, the values of the original data are mapped to a larger space in sparse form. Figure 9 illustrates 2x2 sparse un-pooling. There are different methods for choosing the location where the value is mapped in sparse un-pooling. For example in [13] the indices have been memorized while applying the pooling layer and in the un-pooling layer those indices are used to map the values.

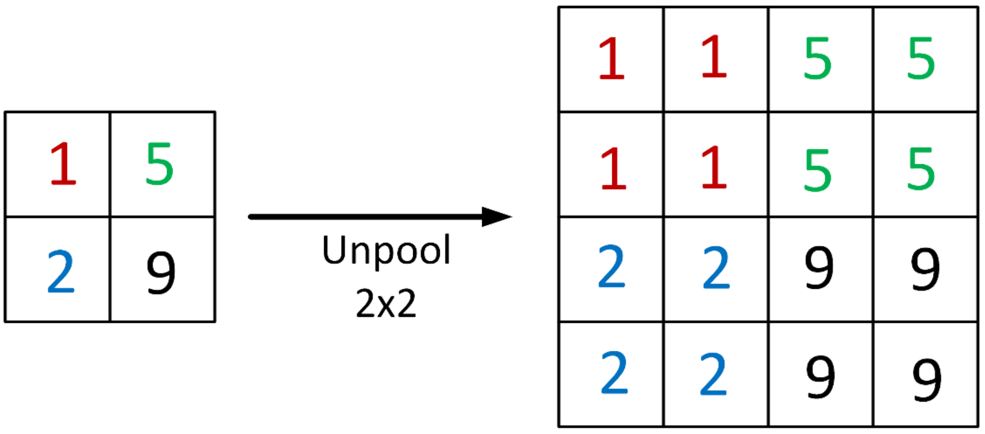


Figure 8: a 2x2 un-pooling operation with repeating values

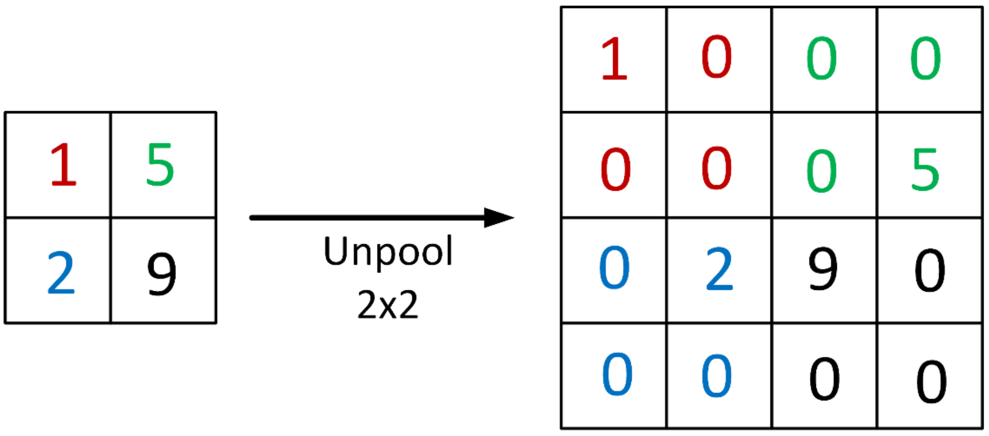


Figure 9: a 2x2 sparse un-pooling operation

2.2.2 Auto-Encoders:

Auto-Encoders are a design of a deep neural network wherein the input and output data are from the same class and also have the same data structure. For example if the input of the network is a 3 channel 128x128 image the output of the auto-encoder is also a 3 channel image with the size 128x128.

The auto-encoder network could be a fully convolutional network or it can have one or several fully connected layers in the middle of the network which is usually known as the bottleneck of the network. The idea with this kind of network is to create a compressed version of the input in the bottleneck. The data can then be reconstructed from the bottleneck. The part of the network that does the compression is called the encoder. The part that decompresses the data from the bottleneck is called the decoder.

The auto-encoder refers to the merged structure of the encoder and decoder in a model. See figure 10.

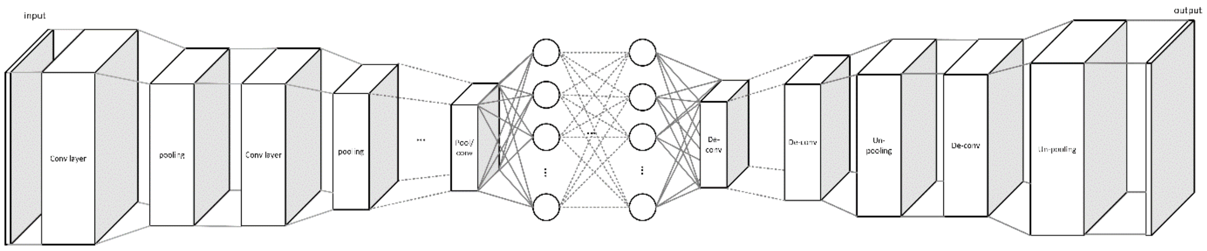


Figure 10: Auto encoder is the merged structure of the encoder and decoder in a model

2.2.4 Recurrent Neural Networks (RNN):

These networks are designed to operate on a sequence of the data as input, for example, a sequence of frames from a video. This can be considered as a system with memory that can remember the input at the previous stage and make decision for the present input based on the sequence of the data before.

The memory of an RNN networks is known as the hidden state of the network. The hidden state of the network is updated based on the current state. The input to the network and the output of the RNN is calculated based on the state of the network at each sequence. Recurrent Neural networks have had great success in speech and video scene recognition where they are often combined with convolutional layers.

3. State of Art Today

3.1 Main Enabling Technologies (overview)

GPU – how they impact training

Despite advancements in parallel pipelines, hyperthreading, and multiple cores, modern CPU's are optimized primarily for problems that are best solved sequentially. This is ideally suited for many common algorithms and tasks, but is not performant for tasks with inherent parallelism. GPU (graphics processing units) are designed for highly parallel graphical processes which can normally be reduced to a limited set of matrix or tensor operations. As GPUs became more and more powerful to cope with the increasing sophistication of 3D graphical models used in gaming, and in augmented and virtual reality, they became the preferred option for deep learning research. A typical GPU has hundreds or thousands of cores, and although each core is much slower than a typical CPU core, together they are able to train networks (especially deep neural networks) at the level of 1 or 2 orders of magnitude faster than on a CPU. This makes intuitive sense. Human neurons are significantly slower than GPU or CPU cores, but our brains are able to perform many recognition and classification tasks faster and with more accuracy than most computer models. This is because human brains have billions of individual, asynchronous neurons, and with highly parallel analog electro-chemical communication.

GPU vs CPU training benchmarks.

For some tasks, a GPU can be on the order of 100X faster while using less power and at a much lower cost. Generally, any task which requires the same operation to be performed thousands of times (as in matrix multiplication) will greatly benefit from the use of a GPU [14].

GPU models and setups:

NVIDIA's latest consumer series is the Pascal series, labelled with a 10 prefix. This series represents a significant improvement in both performance and power consumption over previous models.

	Titan X Pascal	1080	1070	1060
CUDA cores	3584	2560	2048	1280
Memory Capacity	12 GDDR5X	8 GB GDDR5X	8 GB GDDR5	Up to 6 GB GDDR5
Memory Speed	10 Gbps	10 Gbps	8 Gbps	8 Gbps

(source: [15], [16])

Multiple NVIDIA GPUs can be combined with the use of an SLI bridge, but currently, generic drivers only exist for 2 GPU SLI for the 10 series. NVIDIA also sells non consumer GPU's that are specialized for deep learning that do not have this driver limitation.

When choosing a GPU setup for deep learning, it is important to choose a device that has enough memory to fit your model and enough cores to efficiently solve the problem. One also needs to decide on multi GPU vs single GPU setups. It is also important that enough system memory exists to easily copy the model between GPU ram and system ram with room to spare for the operating system and any running applications and services.

Deep Learning on a stick

Many companies are deploying hardware customized for deep learning, often targeted at deployment in smart objects. There are significant speed and power consumption improvements that can be realized in hardware implementations of DL operations. Movidius, for example, released the "fathom neural compute stick" which uses approximately 12.5 times less power than an equivalent NVIDIA GPU in a small USB compatible device [17]. It is targeted at robots, drones and surveillance cameras and can drastically improve recognition accuracy. It is a good choice for "off the shelf" deep learning hardware when one is ready to deploy a model.

Figure 7: Movidus fathom USB stick. Copyright Movidius, used with permission.

CUDNN/CUDA

NVIDIA is the most popular consumer GPU manufacturer with Deep Learning researchers. Although their products are targeted primarily at gaming and 3D graphics processing, they also invest significant resources in deep learning research. NVIDIA provides CUDNN, a low level library of deep learning primitives that run on top of CUDA [17].

Developing and evaluating Deep Learning models.

When deploying or designing a deep neural network, it is useful to try out several variations with many parameters. Although speed and power consumption are critical to end products, people often design their networks using slightly slower but higher level frameworks. This allows us to test out ideas and "prove" products before creating highly optimized implementations in hardware or software. Although deep learning can be implemented in nearly any programing language, most resources and community support available is intended for python. Although most deep learning tools are now compatible with python 3X, the deep learning community still primarily uses python

2.7.

The most popular high level frameworks for this are Caffe, Theano, and Tensorflow.

Caffe [18] **–** Created at the Berkely Vision and Learning center. Models are defined in a configuration file instead of being coded directly. They claim to have the fastest convolutional neural network implementation available.

Theano [19]**–** Initially designed for fast, stable symbolic operations (including symbolic differentiation). It dynamically generates optimized C code which can be transparently executed on the GPU up to 140 times faster than an equivalent CPU **implementation.**

Tensorflow [20]– The newest of the three, tensorflow was designed by Google and is currently experiencing the fastest growth in usage. As of this writing, tensorflow performs slightly slower than the other two on benchmarks but is easier to deploy on multiple GPUs.

There are a number of frameworks built on top of these platforms, such as Lasagne and Keras that allow further abstraction and thus ease and speed of development. Keras is a good choice, especially for beginners since it can use either Tensorflow and or Theano as a back end and it provides a simpler model for development.

3.2 Example applications

Recently, Google provided a mobile Tensorflow API which allows developers to deploy deep learning models directly to smart phones [21]. Google uses it for “Translate's instant visual translation” and recommend it for any application where processing needs to be done on the device

FotoNation (www.fotonation.com), a company whose software runs in over 2.7 Billion smartphones worldwide, has released a new Face Recognition product based on the latest deep learning techniques to achieve rapid and accurate face recognition.

Nuance uses deep learning for their “Dragon Naturally Speaking” line of voice recognition applications [22]. Deep learning is used in both initial product development and later for custom refinement on consumer electronics.

Prisma, with 27 million users, is a popular cell phone app that uses deep learning to transform photos into paintings in the style of famous artists [23].

Soundhound, inc is collaborating with NVIDIA using deep learning to create “to add a smart, voice-enabled, conversational interface to every technology that humans interact with” based on their platform called “Houndafiy” [24].

At the WWDC 2016 keynote, Apple’s John Gruber revealed that they use deep learning in their photo app, first in their own datacenter where they pre-train a model based on a vast base of labeled photos, and later on the iPhone to annotate images as they are taken. They also analyze all photos on a phone when the device is plugged in and not in use so as to avoid draining battery power. Google takes a different approach, avoiding “on device” classification, and instead processing users’ photos with their cloud infrastructure [25].

Neurala released an API that uses deep learning to perform real time object tracking, recognition, and other tasks. As of September 2016 they are the largest supplier of deep learning software for Consumer Drones. Their deep learning software allows drones to operate autonomously, enabling tasks such as “follow this person” or “find this car”. Good performance on such tasks were the realm of sci-fi only a few years ago [26].

4. Thoughts & Conclusions

Today deep learning is being used in our cell phones, in our cars, in tablets and computers. It has pushed the boundaries of what is possible for tasks such as Image segmentation [13], object detection [27], face recognition [28], voice analyzing [29], emotion detection [30] and gender recognition [31].

Why has Deep Learning suddenly catalyzed research across so many fields? Well it is a combination of many factors: the recent emergence of highly affordable high-density, GPU-based computational hardware has provided the engines to process very large datasets and implement the advanced training methodologies required to develop accurate CNNs; the widespread availability of GPUs in today’s devices, coupled with cloud-based data processing services provides the means to apply these CNN architectures to everyday applications such as voice or image processing. Big data provides the fuel to drive research activity and refine results to the point where Deep Learning solutions typically outperform even the best of human-designed pattern recognition tools.

Today we are at a point where many new problems can be tackled through the use of Deep Learning techniques – many of these are long-standing problems such as voice recognition which was never quite good enough to make its way out of the laboratory and into everyday use. But this year we have seen several launches of ‘smart speakers’ that can control your home – and behind these new devices lies a web of deep learning technologies – both analyzing your needs, translating your voice requests and marshalling the necessary logistics to deliver everything to your doorstep or onto your TV set.

To get started with solving your own problem you’ll need a state-of-art GPU – the same technology going into the latest gaming PCs – and most of the core software is freely available on the Internet. There are several software packages trending in the Deep Learning field, including but not limited to Theano (on python), Lasagne (on Theano), Tensorflow (on python and C++), Caffe (on python and MATLAB), and MatConvNet (on MATLAB).

And the good news is that all of these tools are relatively inexpensive and readily available!

Finally you’ll need a large dataset – for your particular application we’ll assume you have your own specialized data sources, but there are many public sources of suitable data. And from here its is deep oceans of learning ahoy!

References

[1] B. Yegnanarayana, *Artificial neural networks*. PHI Learning Pvt. Ltd., 2009.

[2] S. Lawrence, C. L. Giles, and A. C. Tsoi, “What Size Neural Network Gives Optimal Generalization ? Convergence Properties of Backpropagation,” *Networks*, no. UMIACS-TR-96-22 and CS-TR-3617, pp. 1–37, 1996.

[3] Y. Chauvin, “Generalization performance of overtrained back-propagation networks,” in *Neural Networks*, Springer, 1990, pp. 45–55.

[4] S. Theodoridis and K. Koutroumbas, “Pattern recognition and neural networks,” in *Machine Learning and Its Applications*, Springer, 2001, pp. 169–195.

[5] P. Lewis, “Analogy between human and artificial neural nets.” [Online]. Available: http://users.ecs.soton.ac.uk/phl/ctit/nn/node2.html.

[6] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *Bull. Math. Biophys.*, vol. 5, no. 4, pp. 115–133, 1943.

[7] F. Rosenblatt, “The Perceptron - A Perceiving and Recognizing Automaton,” *Report 85, Cornell Aeronautical Laboratory*. pp. 460–1, 1957.

[8] A. McWilliams, “Smart machines: Technologies and global markets,” *BCC Res.*, no. May, 2014.

[9] G. Chandrashekar and F. Sahin, “A survey on feature selection methods,” *Comput. Electr. Eng.*, vol. 40, no. 1, pp. 16–28, 2014.

[10] “Convolutional Neural Networks (LeNet).” [Online]. Available: http://deeplearning.net/tutorial/lenet.html.

[11] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, “Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs),” *CoRR*, vol. abs/1511.0, 2015.

[12] J. Schmidhuber, “Who Invented Backpropagation?,” 2014. [Online]. Available: http://people.idsia.ch/~juergen/who-invented-backpropagation.html.

[13] V. Badrinarayanan, A. Kendall, and R. Cipolla, “SegNet: {A} Deep Convolutional Encoder-Decoder Architecture for Image Segmentation,” *CoRR*, vol. abs/1511.0, 2015.

[14] K. Krewell, “What’s the Difference Between a CPU and a GPU,” 2009. [Online]. Available: https://blogs.nvidia.com/blog/2009/12/16/whats-the-difference-between-a-cpu-and-a-gpu/.

[15] “GEFORCE GTX 10-SERIES NOTEBOOKS.” [Online]. Available: http://www.geforce.com/hardware/10series/notebook.

[16] “NVIDIA TitanX Graphics Card with Pascal.” [Online]. Available: http://www.geforce.com/hardware/10series/titan-x-pascal.

[17] “NVIDIA cuDNN.” [Online]. Available: https://developer.nvidia.com/cudnn.

[18] “Caffe.” [Online]. Available: http://caffe.berkeleyvision.org/.

[19] “Theano.” [Online]. Available: http://deeplearning.net/software/theano/.

[20] “TensorFlow is an Open Source Software Library for Machine Intelligence.” [Online]. Available: https://www.tensorflow.org/.

[21] “Mobile TensorFlow.” [Online]. Available: https://www.tensorflow.org/mobile.html.

[22] N. Lenke, “Why we’re using Deep Learning for our Dragon speech recognition engine,” 2016. [Online]. Available: http://whatsnext.nuance.com/in-the-labs/dragon-deep-learning-speech-recognition/.

[23] “The Technology Behind the Viral Prisma Photo App,” 2016. [Online]. Available: https://news.developer.nvidia.com/the-technology-behind-the-viral-prisma-photo-app/.

[24] “SoundHound Inc. Collaborates With NVIDIA to Bring Deep Learning-Based Natural Language Understanding to Cars,” 2016. [Online]. Available: http://www.businesswire.com/news/home/20160105006247/en/SoundHound-Collaborates-NVIDIA-Bring-Deep-Learning-Based-Natural.

[25] “The Technology Behind Apple Photos And The Future Of Deep Learning And Privacy,” 2016. [Online]. Available: http://highscalability.com/blog/2016/6/20/the-technology-behind-apple-

photos-and-the-future-of-deep-le.html.

[26] “Neurala Becomes Largest Supplier od Deep Learning Software Running in Real Time on a Drone,” 2016. [Online]. Available: http://www.neurala.com/top-deep-learning-software/.

[27] C. Szegedy *et al.*, “Going Deeper with Convolutions,” 2014.

[28] Y. Sun, Y. Chen, X. Wang, and X. Tang, “Deep learning face representation by joint identification-verification,” in *Advances in Neural Information Processing Systems*, 2014, pp. 1988–1996.

[29] X.-L. Zhang and J. Wu, “Deep belief networks based voice activity detection,” *IEEE Trans. Audio. Speech. Lang. Processing*, vol. 21, no. 4, pp. 697–710, 2013.

[30] S. Bazrafkan, T. Nedelcu, P. Filipczuk, and P. Corcoran, “Deep Learning for Facial Expression Recognition: A step closer to a SmartPhone that Knows your Moods,” in *IEEE International Conference on Consumer Electronics (ICCE)*, 2017.

[31] J. Lemley, S. Abdul-Wahid, D. Banik, and R. Andonie, “Comparison of Recent Machine Learning Techniques for Gender Recognition from Facial Images,” in *Modern Artificial Intelligence and Cognitive Science (MAICS 2016)*, 2016.

Appendix B

Smart Augmentation - Learning an Optimal Data Augmentation Strategy

Smart Augmentation
Learning an optimal data augmentation strategy

Joseph Lemley
Collage of Engineering and Informatics
National University of Ireland Galway
Galway Ireland
Email: j.lemley2@nuigalway.ie

Shabab Bazrafkan
Collage of Engineering and Informatics
National University of Ireland Galway
Galway Ireland
Email: s.bazrafkan1@nuigalway.ie

Peter Corcoran
Collage of Engineering and Informatics
National University of Ireland Galway
Galway Ireland
Email: peter.corcoran@nuigalway.ie

***Abstract*—A recurring problem faced when training neural networks is that there is typically not enough data to maximize the generalization capability of deep neural networks(DNN). There are many techniques to address this, including data augmentation, dropout, regularization, and transfer learning. In this paper, we introduce an additional method which we call Smart Augmentation and we show how to use it to increase the accuracy and reduce overfitting on a target network. Smart Augmentation works by creating a network that learns how to generate augmented data during the training process of the target network in a way that reduces the loss of the target neural network. This allows us to learn augmentations that minimize the error of that network.**

I. Introduction

In order to train a deep neural network, the first and probably most important task is to have access to enough labeled samples of data. Not having enough quality labeled data will generate overfitting, which means that the network is highly biased to the data it has seen in the training set and, therefore will not be able to generalize the learned model to any other samples. In [1] there is a discussion about how much the diversity in training data and mixing different datasets can affect the model generalization. Mixing several datasets might be a good solution, but it is not always feasible due to lack of accessibility. One of the other approaches to solve this problem is using different regularization techniques. In recent years different regularization approaches have been proposed and successfully tested on deep neural network models. The drop-out technique [2] and batch normalization [3] are two well-know regularization methods used to avoid overfitting when training deep models.

Another technique for addressing this problem is called augmentation. Data augmentation is the process of supplementing a dataset with similar data that is created from the information in that dataset. The use of augmentation in deep learning is ubiquitous, and when dealing with images, often includes application of rotation, translation, blurring and other modifications to existing images that allow a network to better generalize [4].

Augmentation serves as a type of regularization, reducing the chance of overfitting by extracting more general information from the database and passing it to the network. One can classify the augmentation methods into two different types. The first is unsupervised augmentation. In this type of augmentation the data expansion task is done regardless of the label of the sample. For example adding different kind of noise, rotating or flipping the data. These kinds of data augmentations are usually not difficult to implement.

Next there is Supervised augmentation. One of the most challenging kind of data expansion is mixing different samples with the same label in feature space in order to generate a new sample with the same label. The generated sample has to be recognizable as a valid data sample, and also as a sample representative of that specific class. Since the label of the data is used to generate the new sample, this kind of augmentation is named supervised augmentation.

Many deep learning frameworks can generate augmented data. For example, Keras [5] has a built in method to randomly flip, rotate, and scale images during training but not all of these methods will improve performance and should not be used blindly. For example, on MNIST (The famous hand written number dataset), if one adds rotation, the network will be unable to distinguish properly between hand written 6 and 9 digits. Likewise a system that uses deep learning to classify or interpret road signs may become incapable of discerning left and right arrows if the training set was augmented with by indiscriminate flipping of images.

More sophisticated types of augmentation, such as selectively blending images or adding directional lighting rely on expert knowledge. Besides intuition and experience, there is no universal method that can determine if any specific augmentation strategy will improve results until after training. Since training deep neural nets is a time consuming process, this means only a limited number of augmentation strategies will likely be attempted before deployment of a model.

Blending several samples in the dataset in order to highlight their mutual information is not a trivial task in practice. Which samples should be mixed together how many of them and how they mixed is a big problem in data augmentation using blending techniques.

Augmentation is typically performed by trial and error, and the types of augmentation performed are limited to the imagination, time, and experience of the researcher. Often, the choice of augmentation strategy can be more important than the type of network architecture used [6]. Before Convolutional Neural Networks (CNN) became the norm for computer vision research, features were "hand crafted". Hand crafting features went out of style after it was shown that Convolutional Neural Networks could learn the best features for a given task. We suggest that since the CNN can generate the best features for some specific pattern recognition tasks, it might be able to give the best feature space in order to merge several samples in a specific class and generate a new sample with the same label. Our idea is to generate the merged data in a way that produces the best results for a specific target network through intelligent blending of features between 2 or more samples.

II. Related Work

Manual augmentation techniques such as rotating, flipping and adding different kinds of noise to the data samples, are described in depth in [4] and [7] which attempt to measure the performance gain given by specific augmentation techniques. They also provide a list of recommended data augmentation methods.

In 2014, Srivastava et al. introduced the dropout technique [2] aiming to reduce overfitting, especially in cases where there is not enough data. Dropout works by temporarily removing a unit (or artificial neuron) from the Artificial Neural Network and any connections to or from that unit.

Konda et al. Showed that dropout can be used for data augmentation by "projecting the the dropout noise within a network back into the input space". [8]

Jaderberg et al. devised an image blending strategy as part of their paper "Synthetic Data and Artificial Neural Networks for Natural Scene Text Recognition" [9]. They used what they call "natural data blending" where each of the image layers are blended with a randomly sampled crop of an image from a training dataset. They note a significant (+44%) increase in accuracy using such synthetic images when image layers are blended together via a random process.

Another related technique is training on adversarial examples. Goodfellow et al. notes that, although augmentation is usually done with the goal of creating images that are as similar as possible to the natural images one expects in the testing set, this does not need to be the case. They further demonstrate that training with adversarial examples can increase the generalization capacity of a network, helping to expose and overcome flaws in the decision function [10].

The use of Generative Adversarial Neural Networks [11] is a very powerful unsupervised learning technique that uses a min-max strategy wherein a 'counterfeiter' network attempts to generate images that look enough like images within a dataset to 'fool' a second network while the second network learns to detect counterfeits. This process continues until the synthetic data is nearly indistinguishable from what one would expect real data to look like. Generative Adversarial Neural Networks can also be used to generate images that augment datasets, as in the strategy employed by Shrivastav et al. [12]

Another method of increasing the generalization capacity of a neural network is called "transfer learning". In transfer learning we want to take knowledge learned from one network, and transfer it to another [13]. In the case of Convolutional Neural Networks, when used as a technique to reduce over-fitting due to small datasets, it is common to use the trained weights from a large network that was trained for a specific task and to use it as a starting point for training the network to perform well on another task.

Batch normalization, introduced in 2015, is another powerful technique. It was discovered upon the realization that normalization need not not just be performed on the input layer, but can also be achieved on intermediate layers. [3]

Like the above regularization methods, Smart Augmentation attempts to address the issue of limited training data to improve regularization and reduce overfitting. As with [10], our method does not attempt to produce augmentations that appear "natural". Instead our network learns to combine images in ways that improve regularization. Unlike [4] and [7], we do not address manual augmentation, nor does our network attempt to learn simple transformations. Unlike the approach of image blending in [9], we do not arbitrarily or randomly blend images. Smart augmentation can be used in conjunction with other regularization techniques, including dropout and traditional augmentation.

III. Smart Augmentation

Smart Augmentation is the process of learning suitable augmentations when training deep neural networks.

The goal of Smart Augmentation is to learn the best augmentation strategy for a given class of input data. It does this by learning to merge two or more samples in one class. This merged sample is then used to train a target network. The loss of the target network is used to inform the augmenter at the same time. This has the result of generating more data for use by the target network. This process often includes letting the network come up with unusual or unexpected but highly performant augmentation strategies.

During the training phase, we have two networks: Network A, which generates data; and network B, which is the network that will perform a desired task (such as classification). The main goal is to train network B to do some specific task while there are not enough representative samples in the given dataset. To do so, we use another network A to generate new samples. This network accepts several inputs from the same class (the sample selection could be random, or it could use some form of clustering, either in the pixel space or in the feature space) and generates an output which approximates data from that class. This is done by minimizing the loss function LA which accepts out1 and image i as input. Where out1 is the output of network A and mage i is a selected sample from the same class as the input. The only constraint on the network A is that the input and output of this network should be the same shape and type. For example, if N samples of a P channel image are fed to network A, the output will be a single P channel image.

The loss function can be further parameterized by the inclusion of α and β as $f(\alpha * L_A, \beta * L_B)$. In the experiments and results sections of this paper we examine how these can impact final accuracy.

Network A can either be implemented as a single network (figure 2), or as multiple networks, as in figure1. Using more than one network A has the advantage that the networks can learn class-specific augmentations that may not be suitable for other classes, but which work well for the given class.

Network A is a neural network, such as a generative model, with the difference that network A is being influenced by network B in the back propagation step, and network A accepts multiple samples as input simultaneously instead of just one at a time. This causes the data generated by network A to converge to the best choices to train network B for that specific task, and at the same time it is controlled by loss function LA in a way that ensures that the outputs are similar to other members of its class. The overall loss function during training is $f(LA, LB)$ where f is a function whose output is a transformation of LA and LB. This function could be an epoch-dependent function i.e. the function could change with the epoch number. In the training process, the error back-propagates from network B to network A. This tunes network A to generate the best augmentations for network B. After training is finished, Network A is cut out of the model and network B is used in the test process. The joint information between data samples is exploited to both reduce overfitting, and to increase the accuracy of the target network during training.

The proposed method uses a network (network A) to learn the best sample blending for the specific problem. The output of network A is the used for the input of network B. The idea is to use network A to learn the best data augmentation to train network B. Network A accepts several samples from the same class in the dataset, and generates a new sample from that class, and this new sample should reduce the training loss for network B. In figure 3 we see an output of network A designed to do the gender classification. The image on the left is a merged image of the other two. This image represents a sample from the class "male" that does not appear in the dataset, but still has the identifying features of its class.

Notice that in figure 3, an image was created with an open mouth and open eyes from two images. The quality of the face image produced by network A does not matter. Only its ability to help network B better generalize. Our approach is most applicable to classification tasks but may also have applications in any approach where selective blending of sample features improves performance. Our observations show that this approach can reduce overfitting and increase accuracy. In the following sections we evaluate several implementations of our smart augmentation technique on various datasets to show how it can improve accuracy and prevent overfitting. We also show that with smart augmentation, we can train a very small network to perform as well as (or better than) a much larger network that produces state of the art results.

IV. Methods

Experiments were conducted on NVIDIA Titan X GPU's running a pascal architecture with python 2.7, using the Theano [14] and Lasange frameworks.

A. Data Preparation

To evaluate our method, we chose 4 datasets with characteristics that would allow us to examine the performance of the algorithm on specific types of data. Since the goal of our paper is to measure the impact of the proposed technique, we do not attempt to provide a comparison of techniques that work well on these databases. For such a comparison we refer to [15] for gender datasets or [16] for the places dataset.

1) Highly constrained faces dataset (db1): Our first dataset, db1 was composed from the AR faces database [17] with a total of 4,000 frontal faces of male and female subjects. The data was split in to subject exclusive training, validation, and testing sets, with 70% for training, 20% for validation, and 10% for testing. All face images were reduced to 96X96 grayscale with pixel values normalized between 0 and 1.

2) Augmented, highly constrained faces dataset (db1a): To compare traditional augmentation with smart augmentation and to examine the effect of traditional augmentation on smart augmentation, we created an augmented version of db1 with every combination of flipping, blurring, and rotation (-5,-2,0,2,5 degrees with the axis of rotation at the center of the image). This resulted in a larger training set of 48360 images. The test and validation sets were unaltered from db1. The data was split in to a subject exclusive training, validation, and testing sets with 70% for training, 20% for validation, and 10% for testing. All face images were reduced to 96X96 with pixel values normalized between 0 and 1.

3) FERET: Our second dataset, db2, was the FERET dataset. We converted FERET to grayscale and reduced the size of each image to 100X100 with pixel values normalized between 0 and 1. The data was split in to subject exclusive training, validation, and testing sets, with 70% for training, 20% for validation and 10% for testing.

Color FERET [18] Version 2 was collected between December 1993 and August 1996 and made freely available with the intent of promoting the development of face recognition algorithms. The images are labeled with gender, pose and name.

Although FERET contains a large number of high quality images in different poses and with varying face obstructions (beards, glasses, etc), they all have certain similarities in quality, background, pose, and lighting that make them very easy for modern machine learning methods to correctly classify. In our experiments, we use all images in FERET for which gender labels exist.

4) Adience: Our third dataset, db3, was Adience. We converted Adience to grayscale images with size 100x100 and normalized the pixel values between 0 and 1. The data was split in to subject exclusive training, validation, and testing sets, with 70% for training, 20% for validation and 10% for testing.

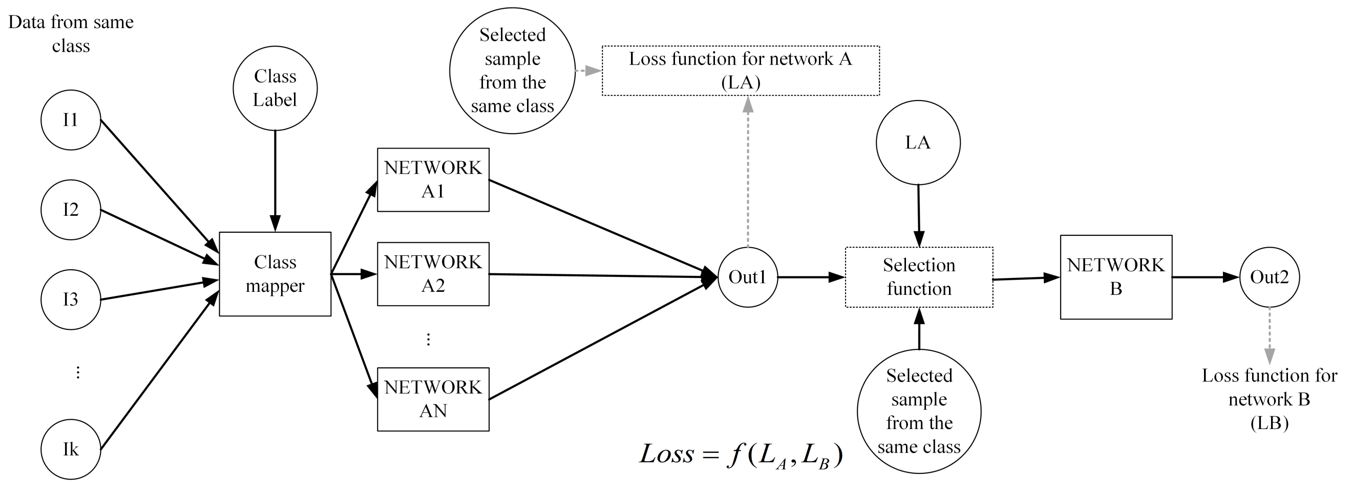


Fig. 1: Smart augmentation with more than one network A

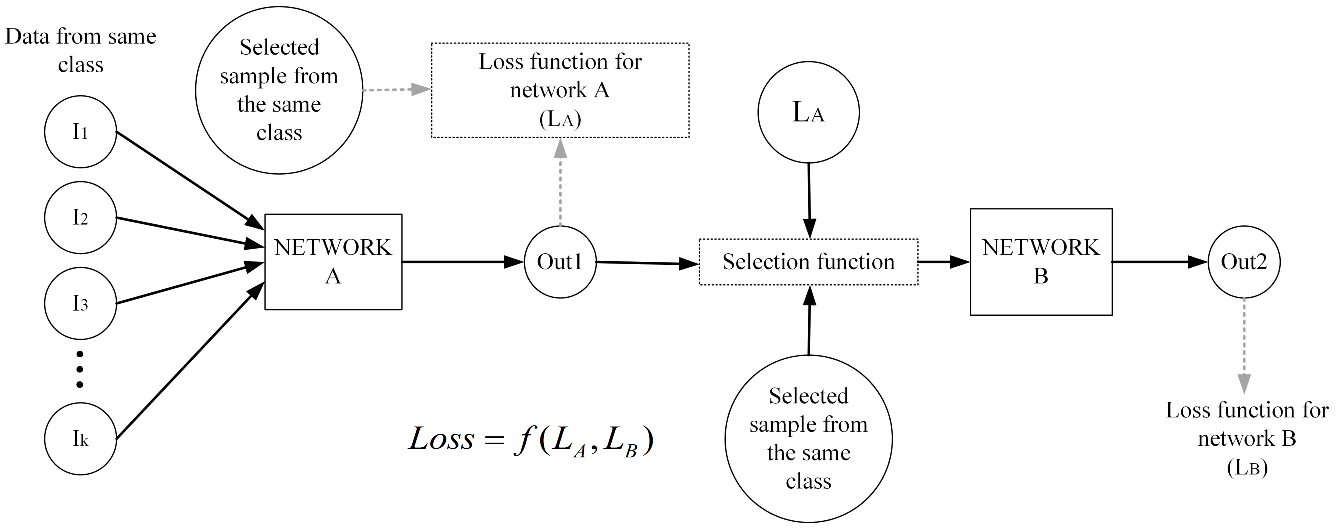

Data from same class
Selected sample from the same class
Loss function for network A (LA)
I1
I2
I3
Ik
NETWORK A
Out1
LA
Selection function
NETWORK B
Out2
Selected sample from the same class
Loss function for network B (LB)
$Loss = f(L_A, L_B)$

Fig. 2: Diagram illustrating the reduced smart augmentation concept with just one network A

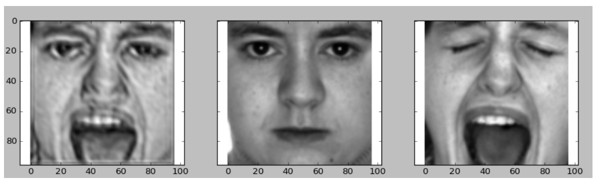


Fig. 3: The image on the left is created by a learned combination of the two images on the right. This type of image transformation helped increase the accuracy of network B. The image was not produced to be an ideal approximation of a face but instead contains features that helped network B better generalize the concept of gender which is the task it was trained for.

Fig. 4: Arbitrarily selected images from FERET demonstrate similarities in lighting, pose, subject, background, and other photographic conditions.

5) DB4: Our fourth dataset, db4, was the MIT places dataset [16]. The MIT PLACES dataset is a machine learning database containing has 205 scene categories and 2.5 million labeled images.

The Places Dataset is unconstrained and includes complex scenery in a variety of lighting conditions and environments.

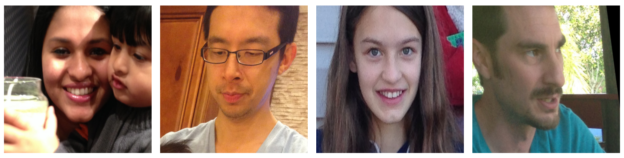


Fig. 5: Arbitrarily selected images from the Adience show significant variations in lighting, pose, subject, background, and other photographic conditions.

We restricted ourselves to just the first two classes in the dataset (Abbey and Airport). Pixel values were normalized between 0 and 1. The "small dataset," which had been rescaled to 256x256 with 3 color channels, was used for all experiments without modification except for normalization of the pixel values between 0 and 1.

Fig. 6: Example images from the MIT places dataset showing two examples from each of the two classes (abbey and airport) used in our experiments.

V. Experiments

In these experiments, we call network B the network that is being trained for a specific task (such as classification). We call network A the network that learns augmentations that help train network B.

All experiments are run for 1000 epochs. The test accuracy reported is for the network that got the highest score on the validation set during those 1000 epochs.

To analyze the effectiveness of Smart Augmentation, we performed 30 experiments using 4 datasets with different parameters. A brief overview of the experiments can be seen in Table I. The experiments were conducted with the motivation of answering the following questions:

1) Is there any difference in accuracy between using smart augmentation and not using it? (Is smart augmentation effective?)
2) If smart augmentation is effective, is it effective on a variety of datasets?
3) As the datasets become increasingly unconstrained, does smart augmentation perform better or worse?
4) What is the effect of increasing the number of channels in the smart augmentation method?
5) Can smart augmentation improve accuracy over traditional augmentation?
6) If smart augmentation and traditional augmentation are combined, are the results better or worse than not combining them?
7) Does altering the α and β parameters change the results?
8) Does Smart Augmentation increase or decrease overfitting as measured by train/test loss ratios?
9) If smart augmentation decreases overfitting, can we use it to replace a large complex network with a simpler one without losing accuracy?
10) What is the effect of the number of network A's on the accuracy? Does training separate networks for each class improve the results?

As listed below, we used three neural network architectures with varied parameters and connection mechanisms. In our experiments, these architectures were combined in various ways as specified in table I.

- Network B_1 is a simple, small Convolutional neural network, trained as a classifier, that takes an image as input, and outputs class labels with a softmax layer. This network is illustrated in figure 7.
- Network B_2 is a unmodified implementation of VGG16 as described in [19]. Network B_2 is a large network that takes an image as input, and outputs class labels with a softmax layer.
- Network A is a Convolutional neural network that takes one or more images as input and outputs a modified image. The details of this network can be seen in figure [insert figure number here]

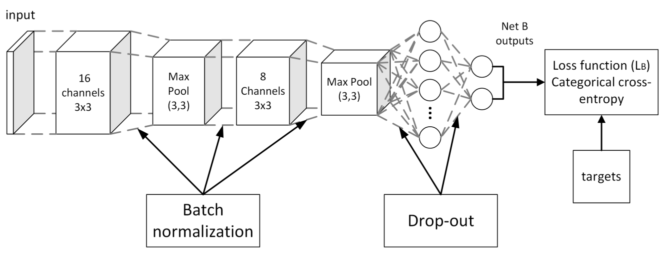


Fig. 7: Illustration of network B_1

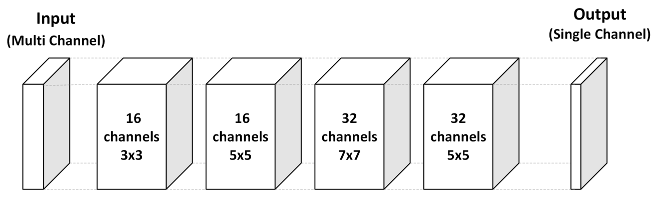
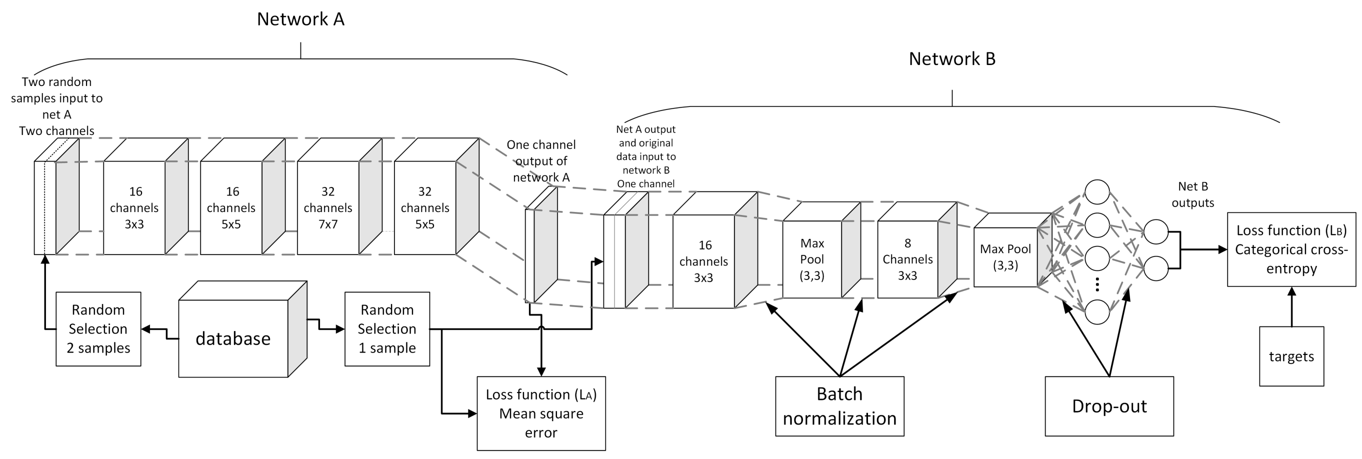


Fig. 8: Illustration of network A

A. Smart Augmentation with one network A on the gender classification task

Experiments 1-8, 19,22, and 24 as seen in table I were trained for gender classification using the same technique as illustrated in figure 9. In these experiments, we use smart augmentation to train a network (network B) for gender classification using the specified database.



Network A
Network B
Two random samples input to net A Two channels
16 channels 3x3
16 channels 5x5
32 channels 7x7
32 channels 5x5
One channel output of network A
Net A output and original data input to network B One channel
16 channels 3x3
Max Pool (3,3)
8 Channels 3x3
Max Pool (3,3)
Net B outputs
Loss function (LB) Categorical cross-entropy
targets
Random Selection 2 samples
database
Random Selection 1 sample
Loss function (LA) Mean square error
Batch normalization
Drop-out

$$Loss = \alpha L_A + \beta L_B$$

Fig. 9: Diagram of simplified implementation of Smart Augmentation showing network A and network B

TABLE I: Full listing of experiments.

Exp	DB	Net A	A ch	Net B	α	β	LR	Momentum
1	db1	1	1	B_1	0.3	0.7	0.01	0.9
2	db1	1	2	B_1	0.3	0.7	0.01	0.9
3	db1	1	3	B_1	0.3	0.7	0.01	0.9
4	db1	1	4	B_1	0.3	0.7	0.01	0.9
5	db1	1	5	B_1	0.3	0.7	0.01	0.9
6	db1	1	6	B_1	0.3	0.7	0.01	0.9
7	db1	1	7	B_1	0.3	0.7	0.01	0.9
8	db1	1	8	B_1	0.3	0.7	0.01	0.9
9	db1	2	1	B_1	0.3	0.7	0.005	0.9
10	db1	2	2	B_1	0.3	0.7	0.005	0.9
11	db1	2	3	B_1	0.3	0.7	0.005	0.9
12	db1	2	4	B_1	0.3	0.7	0.005	0.9
13	db1	2	5	B_1	0.3	0.7	0.005	0.9
14	db1	2	6	B_1	0.3	0.7	0.005	0.9
15	db1	2	7	B_1	0.3	0.7	0.005	0.9
16	db1	2	8	B_1	0.3	0.7	0.005	0.9
17	db1	NA	NA	B_1	NA	NA	0.01	0.9
18	db1a	NA	NA	B_1	NA	NA	0.01	0.9
19	db1a	1	2	B_1	0.3	0.7	0.01	0.9
20	db1a	2	2	B_1	0.3	0.7	0.005	0.9
21	db2	NA	NA	B_1	NA	NA	0.01	0.9
22	db2	1	2	B_1	0.3	0.7	0.01	0.9
23	db3	NA	NA	B_1	NA	NA	0.01	0.9
24	db3	1	2	B_1	0.3	0.7	0.01	0.9
25	db4	NA	NA	B_2	NA	NA	0.005	0.9
26	db4	NA	NA	B_1	NA	NA	0.005	0.9
27	db4	1	2	B_1	0.3	0.7	0.01	0.9
28	db4	1	2	B_1	0.7	0.3	0.01	0.9
29	db4	2	2	B_1	0.7	0.3	0.005	0.9
30	db4	2	2	B_1	0.3	0.7	0.005	0.9

The first, k images are randomly selected from the same class (male or female) in the dataset. These k samples are merged into k channels of a single sample. The grayscale values of the first image, img_0, are mapped to channel 0 and the grayscale values of the second image im_1 are mapped to channel 1 and so on until we reach the number of channels specified in the experiments table. This new k channel image is fed into the network A. Network A is a fully convolutional neural network (See figure 8) which accepts images as the input and gives the images with the same size at the output in single channel.

An additional grayscale image is then randomly selected from the same class in the dataset (this image should not be any of those images selected in step 1). The loss function for this network A is calculated as the mean squared error between this randomly selected image and the output of network A. The output of network A, and the target image are then fed into network B as separate inputs. Network B is a typical deep neural network with two convolutional layers followed by batch normalization and max-pooling steps after each convolutional layer. Two fully connected layers are placed at the end of the network. The first of these layers has 1024 units and the second dense layer is made of two units as the output of network B using softmax. Each dense layer takes advantage of the drop-out technique in order to avoid over-fitting. The loss function of network B is calculated as the categorical cross-entropy between the outputs and the targets.

The total loss of the whole model is a linear combination of the loss functions of two networks. This approach is designed to train a network A that generates samples which reduces the error for network B. The validation loss was calculated only for network B, without considering network. This allows us to compare validation loss with and without smart augmentation.

Our models were trained using Stochastic Gradient Descent with Nestrov Momentum [20], learning rate 0.01 and momentum 0.9. The lasagne library used to train the network in python.

In these experiments, we varied the number of input channels and datasets used. Specifically, we trained a network B from scratch with 1-8 input channels with a single network A on db1, 2 channels on network A for db2 and 3, and 2 channels on network db1a as shown in the table of experiments.

B. Smart Augmentation with two network A's on the gender classification task

In experiments 9-16 and 20 we evaluate a different implementation of smart augmentation, containing a separate network A for each class. As before, the first k images are randomly selected from the same class (male or female) in the dataset. These k samples are merged into k channels of a single sample.The grayscale values of the first image, img_0, are mapped to channel 0 and the grayscale values of the second image, im_1, are mapped to channel 1, and so on until we reach the number of channels specified in the experiments table just as before. Since we now have two network A's, it is important to separate out the loss functions for each network as illustrated in figure 9.

All other loss functions are calculated the same way as before.

One very important difference is the updated learning rate (0.005). While performing initial experiments we noticed that using a learning rate above 0.005 led to the dying relu problem and stopped effective learning within the first two epochs. This network is also more sensitive to variations in batch size.

The goal of these experiments was to examine how using multiple network As impacts accuracy and over fitting compared to just using one network A. We also wanted to know if there were any differences when trained on a manually augmented database (experiment 20).

C. Training without smart augmentation on the gender classification task

In these experiments we train a network (network B) to perform gender classification without applying network A during the training stage. These experiments (23, 21, 18, and 17) are intended to serve as a baseline comparison of what network B can learn without smart augmentation on a specific dataset (db3 ,db2, db1a, and db1 respectively). In this way we measure any improvement given by smart augmentation. A full implementation of Network B is shown in figure 7.

This network has the same architecture as the network network B presented in the previous experiment except that it does not utilize a network A.

As before, two fully connected layers are placed at the end of the network. The first of these layers has 1024 units, and the second dense layer has two units (one for each class). Each dense layer takes advantage of the drop-out technique in order to avoid over-fitting.

All loss functions (training, validation, and testing loss) were calculated as the categorical cross-entropy between the outputs and the targets.

As before, models were trained using Stochastic Gradient Descent with Nestrov Momentum [20], learning rate 0.01 and momentum 0.9. The lasagne library was used to train the network in python.

D. Experiments on the places dataset

In the previous experiments in this section, we used 3 different face datasets. In experiments 25 - 30 we examine the suitability of Smart Augmentation with color scenes from around the world from the MIT Places dataset to evaluate our method on data of a completely different topic. We varied the α and β parameter in our global loss function so that we could identify how they influence results. Unlike in previous experiments, we also retained color information.

Experiment 25 utilized a VGG16 trained from scratch as a classifier, chosen because VGG16 models have performed very well on the places dataset in public competitions [16]. The input to network A was 256x256 RGB images and the output was determined by a 2 class softmax classifier.

In experiment 26 we use a network B, identical in all respects to the one used in the previous subsection, except that we use the lower learning rate specified in the experiments table and take in color images about places instead of gender.

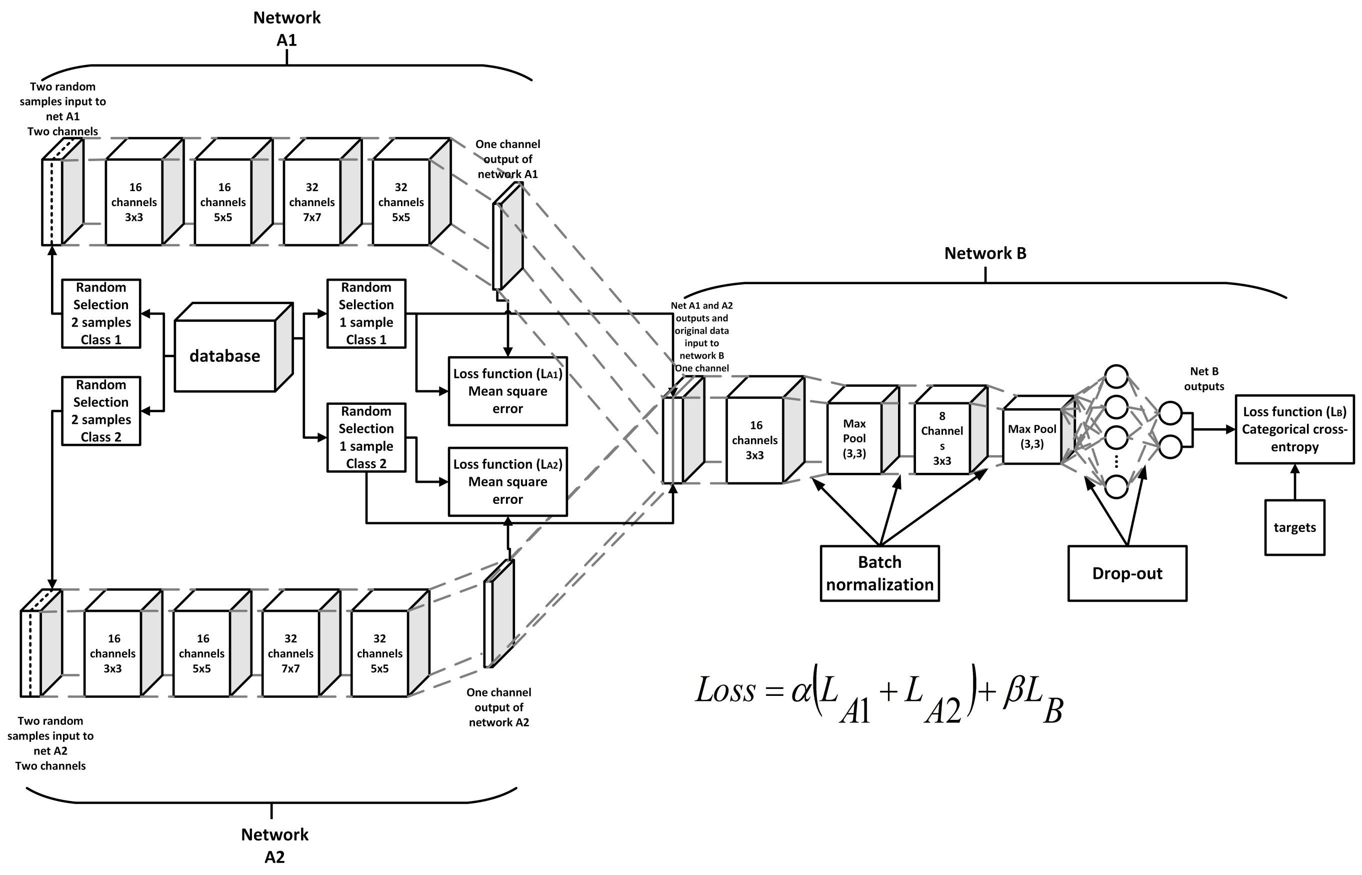
These two experiments (25,26) involved simple classifiers to establish a baseline against which other experiments on the same dataset could be evaluated.

In experiments 27-28, k images were randomly selected from the same class (abbey or airport) in the dataset. These k samples are merged into $k * 3$ channels of a single sample. The values of the first three channels of image img_0 are mapped to channel 0-2, and the first three channels of the second image im_1 are mapped to channels 3-5, and so on, until we reach the number of channels specified in the experiments table multiplied by the number of color channels in the source images. This new $k * 3$ channel image is fed into the network A. Network A is a fully convolutional neural network) which accepts images as the input, and outputs a single color image.

An additional image is then randomly selected from the same class in the dataset. The loss function for network A is calculated as the mean squared error between the randomly selected image and the output of network A. The output of network A, and the target image are then fed into network B as separate inputs. Network B is a typical deep neural network with two convolutional layers followed by batch normalization and max-pooling steps after each convolutional layer. Two fully connected layers are placed at the end of the network. The first of these layers has 1024 units and the second dense layer is made of two units as the output of network B using softmax. Each dense layer takes advantage of the drop-out technique in order to avoid over-fitting. The loss function of network B is calculated as the categorical cross-entropy between the outputs and the targets.

The total loss of the whole model is a linear combination of the loss functions of two networks. This approach is designed to train a network A that generates samples that reduce the error for network B. The validation loss was calculated only for network B, without considering network A. This allows us to compare validation loss with and without smart augmentation.

Our models were trained using Stochastic Gradient Descent with Nestrov Momentum [20], learning rate 0.005 and momentum 0.9. The lasagne library used to train the network in python.


Network A1
Two random samples input to net A1 Two channels
16 channels 3x3
16 channels 5x5
32 channels 7x7
32 channels 5x5
One channel output of network A1
Random Selection 2 samples Class 1
database
Random Selection 2 samples Class 2
Random Selection 1 sample Class 1
Random Selection 1 sample Class 2
Loss function (LA1) Mean square error
Loss function (LA2) Mean square error
Network B
Net A1 and A2 outputs and original data input to network B One channel
16 channels 3x3
Max Pool (3,3)
8 Channels 3x3
Max Pool (3,3)
Net B outputs
Loss function (LB) Categorical cross-entropy
targets
Batch normalization
Drop-out
16 channels 3x3
16 channels 5x5
32 channels 7x7
32 channels 5x5
One channel output of network A2
Two random samples input to net A2 Two channels
Network A2

$$Loss = \alpha\left(L_{A1} + L_{A2}\right) + \beta L_B$$

Fig. 10: Diagram of our implementation of Smart Augmentation with one network A for each class

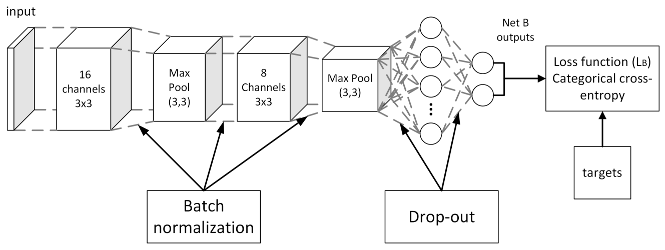


Fig. 11: Diagram of implementation of network B without Smart Augmentation

In these experiments, we varied the number of input channels and datasets used. Specifically, we trained a network B from scratch with 1-8 input channels on network A on db1, 2 channels on network A for db2 and 3, and 2 channels on network db1a as shown in the table of experiments.

In experiments 29-30, k images are randomly selected from the same class (abbey or airport) in the dataset. These k samples are merged into $k * 3$ channels of a single sample. The values of the first three channels in image img_0 are mapped to channel 0-2 and the first three channels of the second image im_1 are mapped to channels 3-5 and so on until we reach the number of channels specified in the experiments table multiplied by the number of color channels in the source images. This new $k * 3$ channel image is fed into the network A. Network A is a fully convolutional neural network which accepts images as the input and outputs a single color image.

An additional image is then randomly selected from the same class in the dataset. The loss function for each network A is calculated as the mean squared error between the randomly selected image and the output of network A. The output of network A, and the target image are then fed into network B as separate inputs. Network B is a typical deep neural network with two convolutional layers followed by batch normalization and max-pooling steps after each convolutional layer. Two fully connected layers are placed at the end of the network. The first of these layers has 1024 units, and the second dense layer is made of two units as the output of network B using softmax. Each dense layer takes advantage of the drop-out technique in order to avoid over-fitting. The loss function of network B is calculated as the categorical cross-entropy between the outputs and the targets.

TABLE II: Results of experiments on Face Datasets

Experiments on Face Datasets				
Dataset	**#Net As**	**Input Channels**	**Augmented**	**Test Accuracy**
AR Faces	1	1	no	0.927746
AR Faces	1	2	no	0.924855
AR Faces	1	3	no	0.950867
AR Faces	1	4	no	0.916185
AR Faces	1	5	no	0.910405
AR Faces	1	6	no	0.933526
AR Faces	1	7	no	0.916185
AR Faces	1	8	no	0.953757
AR Faces	2	1	no	0.869942188
AR Faces	2	2	no	0.956647396
AR Faces	2	3	no	0.942196548
AR Faces	2	4	no	0.942196548
AR Faces	2	5	no	0.907514453
AR Faces	2	6	no	0.933526039
AR Faces	2	7	no	0.916184962
AR Faces	2	8	no	0.924855471
AR Faces	0	NA	no	0.881502867
AR Faces	0	NA	yes	0.890173435
AR Faces	1	2	yes	0.956647396
AR Faces	2	2	yes	0.956647396
Adience	0	NA	no	0.700206399
Adience	1	2	no	0.760577917
FERET	0	NA	no	0.835242271
FERET	1	2	no	0.884581506

TABLE III: Results of experiments on Place Dataset

Experiments on MIT places dataset				
#Net As	**Target Network**	**Test Accuracy**	**A**	**B**
0	VGG16	98.5	NA	NA
0	Small net B	96.5	NA	NA
1	Small net B	98.75	0.3	0.7
1	Small net B	99%	0.7	0.3
2	Small net B	99%	0.7	0.3
2	Small net B	97.87%	0.3	0.7

The total loss of the whole model is a linear combination of the loss functions of the two networks. This approach is designed to train a network A that generates samples that reduce the error for network B. The validation loss was calculated only for network B, without considering network A. This allows us to compare validation loss with and without smart augmentation.

Our models were trained using Stochastic Gradient Descent with Nestrov Momentum [20], learning rate 0.005 and momentum 0.9. The lasagne library was used to train the network in python.

In these experiments, we varied the number of input channels and datasets used. Specifically, we trained a network B from scratch with 1-8 input channels on network A on db1, 2 channels on network A for db2 and 3, and 2 channels on network db1a as shown in the table of experiments.

VI. Results

The results of experiments 1-30 as shown in Table I are listed in tables II and III.

A. Smart Augmentation with one network A on the gender classification task

In figure 12, we show the training and validation loss for experiments 1 and 17. As can be observed , the rate of overfitting was greatly reduced when smart augmentation was used compared to when it was not used.

One can see how the smart augmentation technique could prevent network B from overfitting in the training stage. The smaller difference between training loss and validation loss caused by the smart augmentation technique shows how this approach helps the network B to learn more general features for this task. Network B also had higher accuracy on the test set when trained with smart augmentation (92% compared to 88 percent without).

In figures 13 and 14 we show examples of the kinds of images network A learned to generate. In these figures, the image on the left side is the blended image of the other two images produced by network A.

We observe an improvement in accuracy from 83.52% to 88.46% from smart augmentation on Feret with 2 inputs and an increase from 70.02% to 76.06% on the adience dataset.

We see that there is no noticeable pattern when we vary the number of inputs for network A. Despite lack of a pattern, a significant difference was observed with 8 and 3 channels providing the best results at 95.38% and 95.09% respectively. At the lower end, 7, 5, and 4 channels performed the worst, with accuracies of 91.62%, 91.04%, and 91.04%.

For comparison, the accuracy without network A was: 88.15%. We suspect that much of the variation in accuracy reported above may be due to chance. Since in this particular experiment, images are chosen randomly there may be times when 2 or more images with very helpful mutual information are present by chance and the opposite is also possible. It is interesting that when 3 and 8 channels were used for network A, the accuracy was over 95%.

B. Smart augmentation and Traditional Augmentation

We note that traditional augmentation improved the accuracy from 88.15% to 89.08% without smart augmentation on the gender classification task. When we add smart augmentation we realize an improvement in accuracy to 95.66%

The results of the same experiment when we used 2 networks A's was also 95.66 % which seems to indicate that both configurations may have found the same optima when smart augmentation was combined with traditional augmentation.

This demonstrates that smart augmentation can be used with traditional augmentation to further improve accuracy. It is clear that in all cases examined so far, smart augmentation performed better than traditional augmentation. However, since there are no practical limits on the types of traditional augmentation that can be performed, there is no way to guarantee that manual augmentation could not find a better augmentation strategy. This is not a major concern since we do not claim that smart augmentation should replace traditional augmentation. We only claim that smart augmentation can help with regularization.

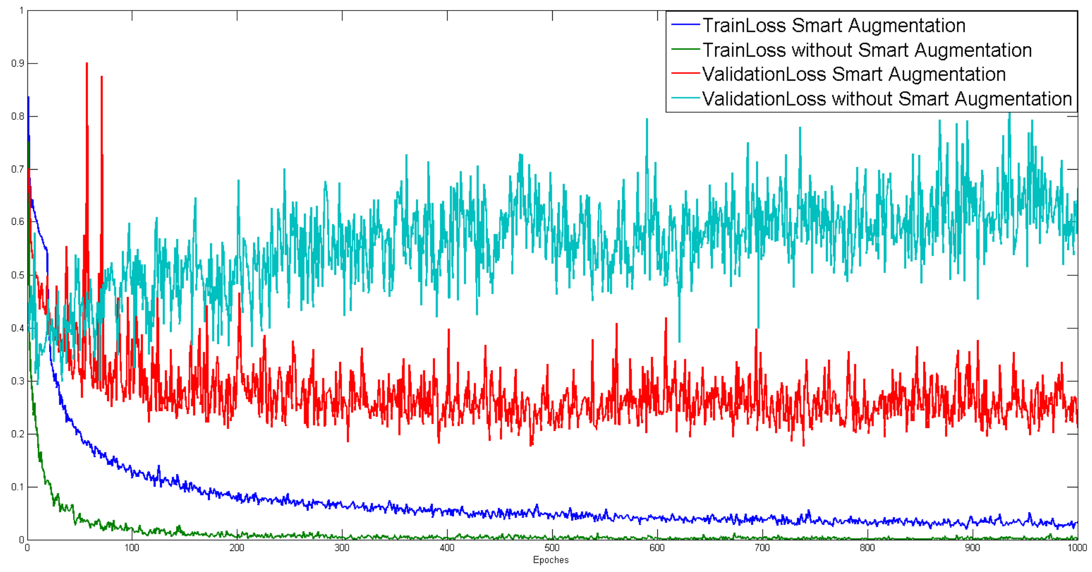


Fig. 12: Training and validation losses for experiments 1 and 17, showing reductions in overfitting by using Smart Augmentation

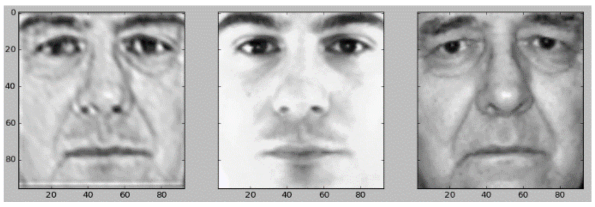


Fig. 13: The image on the left is a learned combination of the two images on the right as produced by network A

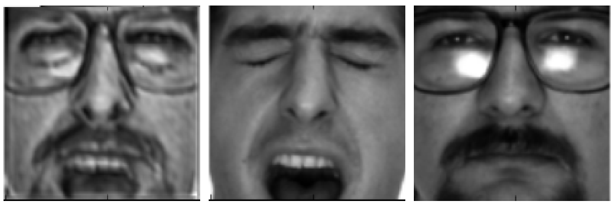


Fig. 14: The image on the left is a learned combination of the two images on the right as produced by network A

C. Smart Augmentation with two network A's on the gender classification task

In this subsection we discuss the results of our two network architecture when trained on the gender classification set.

These experiments show that approaches which use a distinct network A for each class, tend to slightly outperform networks with just 1 network A. This seems to provide support for our initial idea that one network A should be used for each class so that class-specific augmentations could be more efficiently learned. If the networks with just 1 and 0 input channels are excluded, we see an average increase in accuracy from 92.94% to 93.19% when smart augmentation is used, with the median accuracy going from 92.49% to 93.35%.

There is only one experiment where smart augmentation performed worse than not using smart augmentation. This can be seen in the 9th row of table II where we use only one channel which caused the accuracy to dip to 86.99%, contrasted with 88.15% when no smart augmentation is used. This is expected because when only one channel is used, mutual information can not be effectively utilized. This experiment shows the importance of always using at least 2 channels.

D. Experiments on the places dataset

As with previously discussed results, when the places dataset is used, networks with multiple network A's performed slightly better. We also notice that when α is higher than β an increase in accuracy is realized.

The most significant results of this set of experiments is the comparison between smart augmentation, VGG 16, and network B trained alone. Note that a small network B trained alone (no smart augmentation) had an accuracy of 96.5% compared to VGG 16 (no smart augmentation) at 98.5. When the same small network B was trained with smart augmentation we see accuracies ranging from 98.75% to 99% which indicates that smart augmentation, in some cases, can allow a much smaller network to replace a larger network.

VII. Discussion and Conclusion

In this paper we discussed a new regularization approach, called "Smart Augmentation" to automatically learn suitable augmentations during the process of training a deep neural network. We focus on learning augmentations that take advantage of the mutual information within a class. The proposed solution was tested on progressively more difficult datasets starting with a highly constrained face database and ending with a highly complex and unconstrained database of places. This indicates that our method is appropriate for a wide range of tasks and demonstrates that it is not biased to any particular type of image data.

Our experiments showed that the augmentation process can be automated, specifically in non trivial cases where two or more samples of a certain class are merged in non linear ways resulting in improved generalization of a target network. Our results indicate that a deep neural network can be used to learn the augmentation task at the same time the task is being learned. We have demonstrated that smart augmentation can be used to reduce overfitting during the training process and reduce the error during testing.

No linear correlation between the number of samples mixed by network A and accuracy was found so long as at least 2 samples are used.

We found that smart augmentation is effective at reducing error and decreasing overfitting and that this is true regardless of how unconstrained the database is. In our experiments we were able to achieve better accuracy with smart augmentation than with traditional augmentation alone. We found that altering the α and β parameters of the loss function slightly impacts results but more experiments are needed to identify if optimal parameters can be found.

Finally, we found that Smart Augmentation on a small network allowed us to achieve better results than those obtained by a much larger network (VGG 16).

Future work may include expanding Smart Augmentation to learn other types of augmentation strategies and performing experiments on larger datasets with significantly more classes. A statistical study to identify the number of channels that give the highest probability of obtaining good results could also be useful.

VIII. Acknowledgements

This research is funded under the SFI Strategic Partnership Program by Science Foundation Ireland (SFI) and FotoNation Ltd. Project ID: 13/SPP/I2868 on Next Generation Imaging for Smartphone and Embedded Platforms. This work is also supported by an Irish Research Council Employment Based Programme Award.

We gratefully acknowledge the support of NVIDIA Corporation with the donation of a Titan X GPU used for this research.

References

[1] S. Bazrafkan, T. Nedelcu, P. Filipczuk, and P. Corcoran, "Deep learning for facial expression recognition: A step closer to a smartphone that knows your moods," in *IEEE International Conference on Consumer Electronics (ICCE)*, 2017.

[2] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting." *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[3] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[4] P. Y. Simard, D. Steinkraus, and J. C. Platt, "Best practices for convolutional neural networks applied to visual document analysis." in *ICDAR*, vol. 3, 2003, pp. 958–962.

[5] F. Chollet, "Keras," https://github.com/fchollet/keras, 2015.

[6] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[7] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman, "Return of the devil in the details: Delving deep into convolutional nets," *CoRR*, vol. abs/1405.3531, 2014. [Online]. Available: http://arxiv.org/abs/1405.3531

[8] K. Konda, X. Bouthillier, R. Memisevic, and P. Vincent, "Dropout as data augmentation," *arXiv preprint arXiv:1506.08700*, 2015.

[9] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman, "Synthetic data and artificial neural networks for natural scene text recognition," *CoRR*, vol. abs/1406.2227, 2014. [Online]. Available: http://arxiv.org/abs/1406.2227

[10] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[11] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, 2014, pp. 2672–2680.

[12] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb, "Learning from simulated and unsupervised images through adversarial training," *arXiv preprint arXiv:1612.07828*, 2016.

[13] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[14] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.02688, May 2016. [Online]. Available: http://arxiv.org/abs/1605.02688

[15] J. Lemley, S. Abdul-Wahid, D. Banik, and R. Andonie, "Comparison of recent machine learning techniques for gender recognition from facial images," in *27th Modern Artificial Intelligence and Cognitive Science Conference*, 2016.

[16] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva, "Learning deep features for scene recognition using places database," in *Advances in neural information processing systems*, 2014, pp. 487–495.

[17] A. Martinez and R. Benavente, "The ar face database," CVC Technical Report #24, Tech. Rep., 1998.

[18] P. J. Phillips, H. Moon, S. A. Rizvi, and P. J. Rauss, "The feret evaluation methodology for face-recognition algorithms," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 10, pp. 1090–1104, 2000.

[19] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[20] I. Sutskever, J. Martens, G. E. Dahl, and G. E. Hinton, "On the importance of initialization and momentum in deep learning." *ICML (3)*, vol. 28, pp. 1139–1147, 2013.

Appendix C

Transfer Learning of Temporal Information for Driver Action Classification

Transfer Learning of Temporal Information for Driver Action Classification

Joseph Lemley
National University of Ireland Galway
College of Engineering and Informatics
Galway, Ireland
j.lemley2@nuigalway.ie

Shabab Bazrafkan
National University of Ireland Galway
College of Engineering and Informatics
Galway, Ireland
S.Bazrafkan1@nuigalway.ie

Peter Corcoran
National University of Ireland Galway
College of Engineering and Informatics
Galway, Ireland
peter.corcoran@nuigalway.ie

ABSTRACT

Correct classification of image data can depend on features learned in multiple sequential frames. We focus on the problem of learning action from video data with an emphasis on driver behavior monitoring. An insufficient quantity of high quality labeled data is a major problem in machine learning research. This is especially true when deep neural networks are used. Although some sufficiently large, general purpose image databases exist for action recognition, most of these are limited to single frames. This kind of data requires that the action recognition task is applied regardless of the temporal information (information from previous and next frames of a video sequence). In this paper, we show that temporal information is useful for accurate classification of video and that the temporal information in lower layers of a convolutional neural network can successfully be transferred from one network to another to greatly improve performance on the driver behavior monitoring task.

KEYWORDS

Deep Learning, Transfer Learning, Action Recognition

1 INTRODUCTION

In recent years, deep learning has become ubiquitous for image classification, with some results exceeding human accuracy for certain tasks. With few exceptions, these impressive results have been limited to a set of tasks for which a single picture provides all the information required to easily distinguish between classes and essentially ignore any time element for the recognition task.

As we approach the limits of frame-based methods, there is a desire to further improve deep learning algorithms by utilizing temporal information, which is information between multiple frames taken sequentially to give a more complete idea of what is happening. Using a single frame, it is trivial to train a classifier to determine if a person is holding a glass, but difficult or impossible to train a classifier to understand if the glass is being picked up or put down. Even distinguishing jogging from walking can be difficult without a time component.

For this more refined level of visual understanding, we need machine learning models that can learn temporal information and information about frame content at the same time. We also need quality labeled data of sufficient size to prevent overfitting. Much work has been done recently to address these issues (described in the related work section below), but there is still a lack of quality data for use in many practical applications. For example, there are no publicly available databases for driver monitoring that are of sufficient size to train a deep neural network from scratch.

MAICS'2017, Fort Wayne Indiana

The largest database for driver behavior monitoring that we could find is the "Distracted Driver Dataset", provided as part of a Kaggle challenge in mid-2016. Although this database is intended for single frame classification, it is possible to identify the original frame sequences from which movies can be created. These movies can then be used for learning a limited amount of temporal information.

In this paper, we address two questions:

"Can temporal information be used to improve accurate classification of driver actions?" and "Can low-level information about temporal information from an unrelated problem be successfully used to better understand driver actions in videos?"

In the next session, we provide brief background information on Recurrent Neural networks and 3D CNN's for the interested reader. In section three, we present related work. In sections 4, 5, and 6, we present the methods, experiments, and results.

2 BACKGROUND

In this section, we provide a brief overview of the key neural network architectures used in this work, including recurrent neural networks, long short term neural networks, convolutional neural networks and 3D convolutional neural networks. We also briefly describe transfer learning. Readers with a knowledge of these topics are encouraged to skip to the Related Work section below.

2.1 RNN (Recurrent Neural Network)

Recurrent neural networks(RNN) [19] are a highly flexible network architecture frequently used to model time series data, such as a sequence of frames from a video. RNNs remember the input at a previous stage and make a decision for the present input based on the sequence of the previous data. The hidden state of an RNN is calculated based on the state of the network at each sequence. The hidden state can be considered the memory. RNNs have had great success in speech and video recognition and are sometimes combined with convolutional layers [11].

2.2 LSTM (Long Short Term Memory)

LSTMs [5] are a type of RNN that are designed with both long and short term memory which permits the modeling of more complex time series. LSTMs include at least 3 gates which control the way information flows.

2.3 CNN(Convolutional Neural Networks)

Convolutional Neural networks (CNNs) were popularized by lecun et al. [9], who used them successfully for handwritten digit classification. These networks are inspired by the organization of the visual cortex and allow spatial information to be more efficiently learned. Convolutional Neural Networks can be used on input of any number of dimensions but due to their success in pictures, are most popularly implemented for 2D input plus color channels. Other popular types of CNN's include 1D CNNs which are commonly used for time series and 3D CNNs which can be used for volumetric data or time series data where the third dimension represents either spatial frames or temporal frames [11].

2.4 Transfer Learning

Transfer learning is the process of transferring knowledge that has already been learned by one neural network into another one. This is often accomplished by copying the learned weights and biases from one or more layers of a fully trained network to a different network. Transfer learning can be used to overcome overfitting issues and to speed up the training process for a related task.

3 RELATED WORK

In 2008, before the recent wave of deep learning, Klaser et al. [8] used 3D HOG (Histogram of Oriented Gradients) descriptors and showed that 3D gradients were able to assist in understanding action from videos. More modern approaches to action classification from videos are primarily based on 3D convolutional neural networks (CNN) or recurrent neural networks (RNN) with some convolutional component or a combination of the two. 3D CNNs have been shown to capture short-term temporal information.

For example, [18] makes a compelling case for the use of 3D CNNs for understanding video data. Their method, which they named C3D, compared favorably to other published results on 5 of 6 generic action datasets used. They also showed that their network learns information about both motion and appearance, first learning appearance and then motion. One problem with this design is that only relatively short action sequences (16 frames) can be learned.

Addressing this problem, Lei, et al. [10] present an interesting approach to action recognition, they combine a 3D convolutional neural network with a hidden Markov model (HMM) and show that their method compares favorably with other methods. This paper adds support to the idea that 3D convolutions are important to understanding short-term temporal information about movement and the need for another mechanic (RNN, HMM, etc) to understand long-term context (more than a few frames)

Keeping on the theme of combining 3D CNN with a network that is able to learn long-term information, Molchanov et al. [12] combined a 3D convolutional neural network with a recurrent neural network to classify short video clips of hand gestures, reporting excellent results.

Donahue et al. introduced a new architecture called LRCN (Long-term Recurrent Convolutional Networks) that combines RNNs with CNNs [2]. Specifically, they show how to train and optimize a long-term RNN model that can account for temporal information in video data. They evaluate their method on the UCF - 101, a database that contains over 12,000 videos with 101 human action classes [16] .

Another new architecture that combines LSTM with convolutions is introduced in "A Machine Learning Approach for Precipitation Nowcasting" [20]. Although the focus of their paper is forecasting precipitation, their method is generally applicable to the task of gathering long and short term time information from video sequences.

In [21], convolutional temporal pooling is used with a long short term memory (LSTM) network to produce state of the art results. In the same year, [2] used a convolutional LSTM to narrate videos.

Because convolutions on 3D data are time intensive, there have been attempts to combine 2D networks while retaining temporal information. For example, in [3], Feichtenhofer et al. train separate CNNs for motion and appearance and report very good results.

Addressing the problem of insufficient data to train a neural network, [7] introduced a large, automatically generated, database gathered from YouTube clips called the sport 1M dataset. They showed that a transfer learning approach is effective at gaining accuracy on UCF 101 when a network is first trained on sports 1M.

Object proposal networks, specifically fast/faster R-CNNs (region-based convolutional neural networks) have been used successfully as in [14],and [22], for action recognition, and seem to work especially well in cases where there is not enough data to train a full network. Hoang, et al. used this method for detecting cell phone usage and hands on steering wheel detection in [4].

Just as there are attempts to classify behavior without exploiting temporal information, there are also ways of addressing the problem of driver behavior monitoring that do not depend on spatial information but instead use only temporal information. For example, [6] compared LSTM, RNN, logistic regression, and deep neural networks, for detecting driver confusion. Rather than using images, that paper uses multimodal sensor data to assess driver confusion. They found that LSTM outperforms the other models, likely because some long-term information is important for this task. Similarly, [13] uses human motion, as given by cell phone sensors, as a biometric, and an RNN was utilized to process sensor signals.

4 METHODS

Experiments were conducted on NVIDIA Titan X GPU's running a pascal architecture with python 2.7, using the Theano [17] and Keras [1].

4.1 Data Preparation

4.1.1 Driver Monitoring Dataset. The Distracted Driver Dataset was provided as part of a Kaggle Competition in 2016. The dataset was created by filming actors on a closed driving course engaging in various distracted and undistracted behaviors. It should be noted that these images were obtained in a controlled environment the car was not actually being driven. It was being pulled by a truck instead. The objective of the competition was to correctly classify still images into 10 categories.

The training set of the distracted driver database contains frames of 26 subjects displaying several of the following behaviors/actions:

(1) c0: safe driving

(2) c1: texting - right
(3) c2: talking on the phone - right
(4) c3: texting - left
(5) c4: talking on the phone - left
(6) c5: operating the radio
(7) c6: drinking
(8) c7: reaching behind
(9) c8: hair and makeup
(10) c9: talking to passenger

Although all the images in the supplied training set are still images, it is possible to reconstruct the original "movies" based on their order in the CSV file supplied with ground truth annotations.

While classifying frames for driver monitoring is an interesting problem, we wanted to see if we could learn anything from the temporal information in the movies. Instead of using individual frames as required for the competition, we created short movie clips, as we describe later. It is not possible to create such movies from the supplied testing set because sequence information was intentionally left out of it by the competition organizers.

First, we arranged the dataset into distinct subject, action segments ordered by time. The generated movies varied between 38 and 135 frames in length (average is about 101) for a total of 100 clips. Because the 3D convolutional neural network we were planning to experiment with requires a fixed frame size, we further processed the videos into fixed sized frames using the sliding time window method. For each of these segments, we create a series of 5, 10, 16, and 30 frame clips. Each clip is made using a sliding window starting with the first (subject, action) with a step equal to approximately 70% of the clip length to allow sufficient frame overlap. Appropriate action labels are applied to each clip.

In all experiments, 20 drivers are used for the training set and the remaining 6 for the validation set. Due to the small size of this database, we did not feel that further dividing the data into a smaller test set would be reasonable.

In our experiments, we used 4 variations of these:

(1) Grayscale video: All videos of the 26 drivers were reduced to 60×80 Grayscale images.
(2) Color video: All videos of the 26 drivers were reduced to 112×112 full color.
(3) Grayscale frames: All frames of the 26 drivers were reduced to 60×80 Grayscale images.
(4) Color Frames: All frames of the 26 drivers were reduced $112 \times' 112$ Grayscale images.

Due to copyright restrictions, we are unable to include examples of these images in this paper, but the interested reader may view them by visiting the relevant competition at:

https://www.kaggle.com/c/state-farm-distracted-driver-detection

4.2 Augmentation

In experiments where augmentation was applied, the ImageDataGenerator class within Keras was used. This class is used to dynamically create augmented images during training given a set of parameters. Since the standard implementation of ImageDataGenerator only supports 2D data, we subclassed it to properly apply the transformations to video data. This modification involved ensuring that the same transformation was applied to every frame of a clip instead of treating each frame as an individual image with a potentially different transformation.

5 EXPERIMENTS

In this section, we describe 10 experiments on the distracted driver database. The experiments were designed to allow comparison between networks that use temporal information (LSTM, 3D CNN, etc) and networks that ignore it (2D CNN). The last experiments are designed to measure the improvement that is achieved by transfer learning.

5.1 2D experiments

We train an implementation of VGG-16 [15] from scratch on the distracted driver dataset in Keras with a learning rate of 0.001. Our loss function was categorical cross entropy and our output class predictions were obtained from a final softmax layer.

We then repeated our experiments on our grayscale dataset with rotation, translation, and feature normalization. By feature normalization, we mean that the inputs are divided by the standard deviation of the dataset.

5.2 Very Small CNNs and LSTMs

In this set of experiments, we utilized several small (one or two layer) configurations of CNN+LSTM, 3DCNN +LSTM, and 3DCNN.

For these experiments, the LSTM used is the one described by [20], implemented in Keras as ConvLSTM2D.

Three networks were evaluated:

Network 1: 3DCNN followed by Softmax (Num_Classes)

One 3D convolution with 16 filters and a kernel size of 3,3,3 followed by a softmax layer.

Network 2: LSTM followed by 3DCNN $\longrightarrow$ Softmax (Num_Classes)

One LSTM with time length 16, and a 3x3x3 convolution kernel followed by one 3D convolutional layer with 16 filters and a kernel size of 3,3,3 ending with a softmax layer.

Network 3: 3DCNN followed by LSTM $\longrightarrow$ Softmax (Num_Classes)

One 3D convolutional layer with 16 filters and a kernel size of 3,3,3 followed by one LSTM with time length 16, and a 3x3x3 convolution ending with a softmax layer.

All three experiments used a learning rate of 0.001 with nestrov momentum as 0.95. Categorical crossentropy was our loss function and stochastic gradient descent as the training method.

These experiments were then repeated with the grayscale video database.

5.3 Experiments with small 3DCNN architectures

Since the distracted driver dataset has a strong tendency to overfit due to the limited number of subjects, we designed a network and training strategy that would delay overfitting for as long as possible and prevent fast convergence. This involved using a high learning rate and large batch size, resulting in a training method that would allow the network to jump over minima. We also observed that the overfitting came from very low level information, so we restricted the first layer to only 8 features. The network architecture was designed as follows:

(1) input layer with 15 frames, a width and height of 112.

(2) 3D Convolutional Layer(RELU). 8 filters with size: 3, 3, 3
(3) 3D Max pooling operator with size: (2,2,2)
(4) 3D Convolutional Layer(RELU). 16 filters with size: 3, 3, 3
(5) 3D Max pooling operator with size: (2,2,2)
(6) 3D Convolutional Layer(RELU). 32 filters with size: 3, 3, 3
(7) 3D Max pooling operator with size: (2,2,2)
(8) 3D Convolutional Layer(RELU). 64 filters with size: 3, 3, 3
(9) 3D Convolutional Layer(RELU). 128 filters with size: 3, 3, 3
(10) 3D Convolutional Layer(RELU). 256 filters with size: 3, 3, 3
(11) Fully connected Layer (RELU) with 4096 units.
(12) Dropout with a probability of 0.5
(13) Fully connected Layer (RELU) with 4096 units.
(14) Dropout with a probability of 0.5
(15) Softmax with 10 units (one for each of the 10 action classes in the Distracted Driver Dataset)

A learning rate of 0.001 with nestrov momentum as 0.95 and categorical crossentropy as the loss function was used. Stochastic gradient descent was used as the optimizer.

Variations with additional dropout and Batch Normalization layers were also evaluated, but only increased the overfitting and decreased validation accuracy.

These experiments were repeated with rotation, translation, and featurewise normalization.

5.4 C3D trained from scratch

In this experiment, we trained a C3D [18] from scratch on the distracted driver dataset. Since C3D was designed to use 16 frame color video clips of size 112 X 112 we used the color video dataset described previously. As with previous experiments, a learning rate of 0.001 with nestrov momentum as 0.95 was used with categorical crosentropy and gradient descent.

5.5 Transfer Learning with C3D

Since other approaches to reducing the overfitting problem were of limited success, we tried a transfer learning approach. The idea is to use pretrained weights from an existing network, trained for a more generic action recognition task, and then to tune them with the Distracted Driver training set.

We used a pretrained 3CD on the sports 1-M dataset, included the architecture, and trained weights for The Sports-1M dataset in a GitHub repository as a 3D CNN. They reported very good results at capturing temporal information using their 3D CNN, so it seemed like a natural place to start.

In this experiment, we investigate the use of transfer learning to overcome the overfitting problem identified in the previous cases. In the previous experiments, the first layers were identified as being the primary source of overfitting, so we wanted to try two transfer learning approaches.

The first transfer learning approach we tried was to train the C3D network with a very low learning rate of 0.0001 without freezing any layers with these pretrained weights.

We then used an alternate transfer learning approach where we trained only on the final softmax layer, freezing the learning rate of the first layers.

Since we previously identified the first layers as being the greatest source of overfitting, we repeated this experiment freezing only the first five layers, and then repeated it again with only the first two layers frozen.

6 RESULTS

In this section, we report the results of the experiments in the previous section. The experiments with the best results are listed in table 1. Since overfitting is found to be the primary cause of validation error in most experiments, we also show details about the loss and accuracy in tables 2 and 3 before and after we reach 100% on the training set.

6.1 2D experiments

The best 2D accuracy we obtain without augmentation is 30% using a modified VGG 16. [15]

Interestingly, randomly augmenting the 2D dataset with shifts of up to 10% in the horizontal or vertical direction is enough to increase the accuracy from 30% to 46.3%. Adding rotation (5 or 15 degrees) lowered accuracy and allowing shifts of greater than 10% also decreased accuracy.

6.2 Very Small CNNs and LSTMs

In this set of experiments, we used 3D convolutional LSTM of 3 different configurations that have been widely cited in the literature as being effective for this task.

Despite widely different architectures, these networks all quickly resulted in an overfit network with 100% accuracy on the training set and roughly 10% accuracy on the validation set, often within the first 10 epochs.

When trained on reduced size grayscale data the results were similar but slightly better with the best accuracy on the validation set at around 16%.

The goal of this set of experiments was to identify the network architecture which would be most promising for measuring the impact of temporal information for the driver monitoring task, but due to the overfitting issue, we can not reach any conclusions from these 3 experiments.

6.3 Experiments with small 3DCNN architectures

Here we report the results of our experiment with small 3DCNN, which was designed to reduce overfitting by forcing the network to concentrate on more general features and by limiting the number of low level features.

This network achieved an accuracy of 39.57% on the validation set which is better than the 2D VGG 16 inspired network, which had been shown to be effective at this task in the Distracted Driver Competition without manual augmentation, but not better than the 2D VGG when augmented data was provided.

Variations with additional dropout and Batch Normalization layers were also evaluated, but only increased the overfitting and decreased validation accuracy.

None of the augmentation strategies that were found to improve accuracy for the 2D network (translation, rotation, featurewise normalization) increased accuracy for this small 3DCNN architecture.

Table 1: Top performing approaches (>= 30%)

Approach	Accuracy
Transfer learning on 3D CNN. First 2 layers frozen	73.35%
Transfer learning on 3D CNN. First 5 layers frozen	60%
2D VGG 16 with augmentation	46%
3D CNN without augmentation	39.57%
2D VGG 16 without augmentation	30%

Table 2: The best result on the validation set before reaching 100% on the training set was:

Train loss*	Val loss	Train Accuracy	Val Accuracy
0.0359	0.9004	0.9984	0.7273

Table 3: The best results on the validation set after reaching 100% on the training set was

Train loss*	Val loss	Train Accuracy	Val Accuracy
0.0173	0.8563	1	0.7335

6.4 C3D trained from scratch

When we trained the C3D from scratch on the distracted driver dataset, the results were: 100% accuracy on the training set. 12% accuracy on the validation set.

6.5 Transfer Learning with C3D

The first experiment was to train the pretrained 3CD network at a very low learning rate using the driver monitoring dataset. This still resulted in overfitting and unsatisfactory results on the validation set (13%).

The next and most successful transfer learning approach we used was to train only on the last few layers, freezing the learning rate of the first layers. When the first 5 layers were frozen, we achieved 60% on the validation set, which is a significant improvement over the previous results. Further refinement (freezing just the first 2 layers) allowed us to achieve over 72% accuracy on the validation set.

*in all cases loss is the categorical cross-entropy calculated on an entire batch.

7 DISCUSSION

When training a CNN, Information flows in a cone-shaped manner (see figure 1). This is a property of the convolution and pooling operations, which merge and mix the information of several pixels (a window of pixels) into a single value. In the early layers of the convolutional network, each row of the layer corresponds to a small spatial portion of the input image. This means that in these early layers, the network is restricted to the details and low-level features of the input data.

This is also the case for temporal information where time is the depth dimension. In the early stages of the 3D network, the kernels are able to observe a very short part of the input time sequence data. i.e., the beginning layers of the network are processing a short time sequence from a small spatial portion of the input data.

This could correspond to small finger movement, or eye blinking for example. These movements are common among a wide variety of human activities like swimming or holding a cell phone. This is the main reason a transfer learning approach that involved freezing the first two layers gave significantly better results than other methods (See figure 2). In our presented approach, the network trained on the sports 1m dataset (a database of YouTube videos focused on sports) was tuned for the driver monitoring task, but the first two layers were frozen during the tuning procedure. The success of these features indicates that the early stage features learned from sports activity classification are generic enough to be used for other human activities.

These features represent the most detailed activities, which are the same for a wide variety of human behaviors. Figure 1 shows that by going deeper in network layers, each row of a convolutional layer is observing a wider and larger temporal and spatial region of the input data.

This suggests that the kernels in these later layers are dealing with coarser features of the input data, which includes specific movements or behaviors of the subject in the input movie stream. This explains why tuning later layers for our specific task converged to a reasonable approximation of driver activities. In figure 2, network A is the C3D network trained on the sports 1M dataset and the network B is the presented network for driver activity classification. As you can see, the parameters from the first two layers of the network A were transferred to network B and the rest of the network was tuned for our task.

This is our explanation of the results obtained by a transfer learning approach.

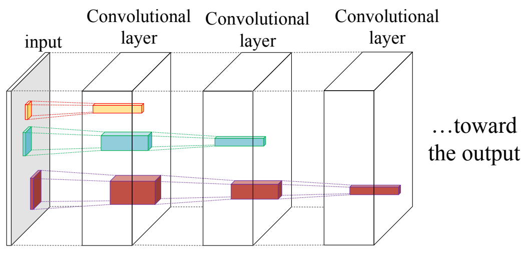


Figure 1: Illustration of cone shape of information flow in convolutional neural networks.

8 CONCLUSIONS

We have shown that the lower level filters learned by a 3D CNN can be used to greatly increase the accuracy on small datasets of drivers for the driver behavior classification task. It is not obvious that the first layers of a network trained for identifying actions in sports videos, such as basketball and swimming, could also be used to distinguish between distracted driver actions like left and right hand cell phone use or speaking with a passenger.

We have also shown that temporal information can be used to increase accuracy for the driver behavior monitoring task over a network that does not use such temporal information.

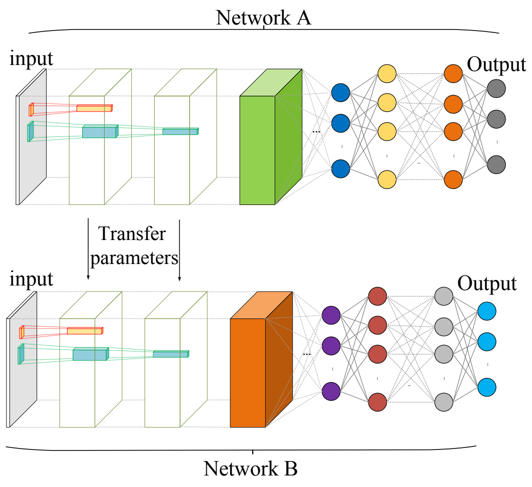


Figure 2: Illustration of transfer learning concept where the first layers in network A and network B are the same.

At a very low level, the action of Moving fingers and heads may not be substantially different between different action recognition problems for convolutional neural networks. In our experiments, using any more than the first two layers decreased accuracy.

Given the shortage of well-labeled task-specific datasets for action recognition, this is an encouraging result.

ACKNOWLEDGMENTS

This research is funded under the SFI Strategic Partnership Program by Science Foundation Ireland (SFI) and FotoNation Ltd. Project ID: 13/SPP/I2868 on Next Generation Imaging for Smartphone and Embedded Platforms. This work is also supported by an Irish Research Council Employment Based Programme Award. Project ID: EBPPG/2016/280.

We gratefully acknowledge the support of NVIDIA Corporation for the donation of a Titan X GPU used for this research.

REFERENCES

[1] François Chollet. 2015. Keras. https://github.com/fchollet/keras. (2015).
[2] Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. 2015. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2625–2634.
[3] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. 2016. Convolutional two-stream network fusion for video action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 1933–1941.
[4] T Hoang Ngan Le, Yutong Zheng, Chenchen Zhu, Khoa Luu, and Marios Savvides. 2016. Multiple Scale Faster-RCNN Approach to Driver's Cell-Phone Usage and Hands on Steering Wheel Detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops.* 46–53.
[5] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.
[6] Chiori Hori, Shinji Watanabe, Takaaki Hori, Bret A Harsham, JohnR Hershey, Yusuke Koji, Yoichi Fujii, and Yuki Furumoto. 2016. Driver confusion status detection using recurrent neural networks. In *Multimedia and Expo (ICME), 2016 IEEE International Conference on.* IEEE, 1–6.
[7] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. 2014. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition.* 1725–1732.
[8] Alexander Klaser, Marcin Marsza lek, and Cordelia Schmid. 2008. A spatio-temporal descriptor based on 3d-gradients. In *BMVC 2008-19th British Machine Vision Conference.* British Machine Vision Association, 275–1.
[9] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. 1989. Backpropagation applied to handwritten zip code recognition. *Neural computation* 1, 4 (1989), 541–551.
[10] Jun Lei, Guohui Li, Jun Zhang, Qiang Guo, and Dan Tu. 2016. Continuous action segmentation and recognition using hybrid convolutional neural network-hidden Markov model model. *IET Computer Vision* 10, 6 (2016), 537–544.
[11] Joe Lemley, Shabab Bazrafkan, and Peter Corcoran. 2017. Deep Learning for Consumer Devices and Services: Pushing the limits for machine learning, artificial intelligence, and computer vision. *IEEE Consumer Electronics Magazine* 6, 2 (2017), 48–56.
[12] Pavlo Molchanov, Xiaodong Yang, Shalini Gupta, Kihwan Kim, Stephen Tyree, and Jan Kautz. 2016. Online detection and classification of dynamic hand gestures with recurrent 3d convolutional neural network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 4207–4215.
[13] Natalia Neverova, Christian Wolf, Griffin Lacey, Lex Fridman, Deepak Chandra, Brandon Barbello, and Graham Taylor. 2016. Learning human identity from motion patterns. *IEEE Access* 4 (2016), 1810–1820.
[14] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems.* 91–99.
[15] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).
[16] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. 2012. UCF101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402* (2012).
[17] Theano Development Team. 2016. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints* abs/1605.02688 (May 2016). http://arxiv.org/abs/1605.02688
[18] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. 2015. Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision.* 4489–4497.
[19] DRGHR Williams and GE Hinton. 1986. Learning representations by back-propagating errors. *Nature* 323, 6088 (1986), 533–538.
[20] SHI Xingjian, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. 2015. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In *Advances in Neural Information Processing Systems.* 802–810.
[21] Joe Yue-Hei Ng, Matthew Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. 2015. Beyond short snippets: Deep networks for video classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition.* 4694–4702.
[22] Liliang Zhang, Liang Lin, Xiaodan Liang, and Kaiming He. 2016. Is Faster R-CNN Doing Well for Pedestrian Detection?. In *European Conference on Computer Vision.* Springer, 443–457.

Appendix D

Convolutional Neural Network Implementation for Eye-Gaze Estimation on Low-Quality Consumer Imaging Systems

Efficient CNN Implementation for Eye-Gaze Estimation on Low-Power/Low-Quality Consumer Imaging Systems

Joseph Lemley, *Student Member, IEEE,* Anuradha Kar, *Student Member, IEEE,* Alexandru Drimbarean, *Member, IEEE,* and Peter Corcoran, *Fellow, IEEE*

***Abstract*—Accurate and efficient eye gaze estimation is important for emerging consumer electronic systems such as driver monitoring systems and novel user interfaces. Such systems are required to operate reliably in difficult, unconstrained environments with low power consumption and at minimal cost. In this paper a new hardware friendly, convolutional neural network model with minimal computational requirements is introduced and assessed for efficient appearance-based gaze estimation. The model is tested and compared against existing appearance based CNN approaches, achieving better eye gaze accuracy with significantly fewer computational requirements. A brief updated literature review is also provided.**

***Index Terms*—Eye gaze, Neural Networks, Deep Learning**

I. Introduction

The potential of eye gaze tracking and gaze-based human computer interactions in modern consumer devices is currently an active topic for exploration. Eye gaze has been used to derive human behavioral cues, as an input modality and for achieving immersive user experiences in virtual and augmented reality systems. However, applications of gaze in consumer devices operating in real world conditions face tough challenges in terms of accuracy and reliability

A. Gaze Tracking in Consumer Devices

After decades of research on desktop-based gaze estimation techniques, the focus has recently shifted to building eye gaze applications for dynamic platforms such as driver monitoring systems [1] and handheld devices [2]. For an automobile driver, eye based cues such as levels of gaze variation, speed of eyelid movements and eye closure can be indicative of a driver's cognitive state. These can be useful inputs for intelligent vehicles to understand driver attentiveness levels, lane change intent, and vehicle control in the presence of obstacles to avoid accidents [3].

Handheld devices like smartphones and tablets form unique platforms for gaze tracking applications wherein gaze may be used as an input modality for device control, activating safety features and novel UI designs [4]. The most challenging aspect of these modern gaze applications includes operation under dynamic user conditions and unconstrained environments. Further requirements for implementing a consumer-grade gaze-tracking system include real-time high-accuracy operation, minimal or no calibration, and robustness to user head movements and varied lighting conditions. Therefore accurate and reliable gaze tracking typically demands high quality cameras and special equipment like narrow angle lenses, external illumination, and stereo setups [5] for capturing eye region features with sufficient details. As a result, gaze estimation systems frequently become costly with complicated setups, which are unsuitable for generic and consumer applications.

Therefore a major challenge of gaze based consumer electronics design involves maximizing system performance while reducing costs and system complexities.

B. Deep Learning for Eye Gaze

In this paper, we introduce a calibration-free method for appearance-based gaze estimation that is suitable for consumer applications and low cost hardware with real time requirements, using a Convolutional Neural Network (CNN).

Convolutional Neural networks (CNNs) were popularized by Lecun et al. [6], who used them successfully for handwritten digit classification. These networks are inspired by the organization of the visual cortex and allow spatial information to be more efficiently learned. Convolutional Neural Networks can be used on input with any number of dimensions, but due to their success in pictures, are most popularly implemented for 2D input plus color channels. Other popular types of CNN's include 1D CNNs, which are commonly used for time series, and 3D CNNs, which can be used for volumetric data or time series data where the third dimension represents either spatial frames or temporal frames [7]. Although CNNs have become ubiquitous for most computer vision tasks, they have yet to become popular for eye gaze estimation.

C. Contributions of this Work

From the perspective of developing a deep learning model for gaze estimation, the task can either be considered as a regression task or a classification task. Although both are useful, regression provides the greatest predictive flexibility and thus this paper treats the eye gaze estimation task as a regression problem with the goal of finding a gaze angle (ϕ,θ) that corresponds with a low resolution eye image such as one

Joseph Lemley, Anurada Kar, and Peter Corcoran are with the Department of Electrical and Electronic Engineering, National University of Ireland Galway, Galway, Ireland e-mail: J.lemley2@nuigalway.ie Alexandru Drimbarean is with Xperi Corporation, Galway Ireland,

taken from a distance with a simple RGB webcam mounted on a dashboard.

In this paper, a hardware optimized network is implemented with demonstrated suitability for deployment on such consumer devices in terms of memory requirements and speed. This network achieves superior accuracy using a dual channel input technique when compared to other state-of-the-art CNN-based gaze tracking methods for unconstrained, low resolution eye tracking.

II. Related work

In this section a review of conventional gaze tracking techniques, studies on using low resolution data, and the application of deep learning in gaze estimation are discussed. The development and usage of important databases for gaze research are also presented.

A. Contemporary Methods for Eye Gaze Estimation

Gaze-tracking algorithms can be broadly classified into two types: model-based methods and appearance-based methods. [8] Appearance-based methods operate directly on the eye images.

Examples of model-based methods include 2D and 3D models that use NIR illumination to create corneal reflections and track them with respect to the pupil center to estimate the gaze vector. These require polynomial or geometric approximations of the human eye to obtain the gaze direction or the point of gaze. Appearance-based methods use eye region images to extract content information such as local features, shape, and texture of eye regions to estimate gaze direction. Some key works in each of these classes of methods are summarized below.

1) Measures of Accuracy for Eye Gaze Tracking Methods: Contemporary research on gaze tracking measures accuracy in a wide variety of ways [9]. For example, commonly used measures include angular resolution in degrees [10], gaze recognition rates in percentage [11], and shifts in number of pixels or distance in cm/mm between gaze [12] and target locations. Unfortunately, these 4 methods are not correlated and not inter-comparable. It is the view of the authors that angular resolution is most reliable as it describes the performance of an algorithm irrespective of other system variables like user distance from tracker, pixel size of screen etc. Therefore, in this work, the angular resolution in degrees is estimated and used as the metric of accuracy for the proposed algorithm. Results from this work are only directly compared to other papers which express their performance in angles.

2) 2D Models: 2D models utilize polynomial transformation functions for mapping the gaze vector (vector between pupil center and corneal glint) to corresponding gaze coordinates on the screen. In [13], it is shown that calibration targets and components of the mapping function are significant in determining overall accuracy of a regression-based tracker. Artificial Neural Network (ANN) based mapping methods are presented in [14], [15], [16] . In [16] a three layer ANN achieves better accuracy than regression-based approaches. Methods that are robust to head pose are presented in [17] and [18]. Multiple geometrical transformation based mapping for handling variable user distances and head motion are used in [17]. A highly accurate calibration-free algorithm with tolerance for natural head movements is discussed in [18] using a support vector machine (SVM). Another high resolution, head-pose-invariant tracking method that does not require geometric models is presented by [19]. The typical accuracy of such models is between two and four degrees.

3) 3D Models: 3D model-based methods typically use a geometrical model of the human eye to estimate the center of the cornea, and the optical and visual axes of the eye. Gaze coordinates are estimated as points of intersection of the visual axes with the scene. These methods achieve high accuracy (1 degree) but require elaborate system setups and knowledge about geometric relations between system components like LEDs, monitors and cameras. [20] presents a mathematical model to estimate the optical and visual axes of the users' eyes from the center of the pupil and glint, considering single and multiple cameras and light sources. Methods achieving high accuracy and head pose robustness are reported in [5], [21], [22], [23] which require multiple cameras in their setup. [22] also uses a dynamic head compensation model for updating the mapping function to track gaze under natural head movement. Calibration-free gaze estimation techniques are proposed by [24], [25] in which cameras, light sources and a spherical model of the cornea are used. Recent developments in 3D gaze tracking include usage of depth sensors along with RGB cameras such as proposed in [26]. In this, 3D gaze coordinates can be tracked in real time with a Kinect device, which provides 3D coordinates of eye features while eye parameters like eyeball and pupil center are derived from user calibration. The Kinect sensor is used in [27] for gaze estimation using free head motion along with the iris center localization method and geometric constraints-based eyeball center estimation.

4) Appearance-Based Methods: Appearance based methods utilize cropped eye images of a subject gazing at known locations to generate gaze point coordinates. The eye images are then used as training data for various machine learning models. For example, in [28], coordinates of eye contours, iris size, location, and pupil positions are estimated using an Active Appearance Model (AAM) and then used as input to a support vector machine (SVM) for gaze estimation. In [12], texture features are obtained using Local-Binary-Pattern (LBP) and used with an SVM along with space coordinates of the eyes for head pose free gaze tracking. A comparative evaluation of different classification methods, such as SVM, neural networks, and k-Nearest Neighbor (k-NN)s is presented in [29] where Local Binary Patterns Histograms (LBPH) and Principle Component Analysis (PCA) are used to extract eye appearance features. The use of Haar features is reported in [30], [31] for real time gaze tracking. In [32] a neural network with a skin colour model to detect face and eye regions is used. Head-pose tolerant tracking is achieved in [33] using a neural network. Recently, appearance-based methods implemented using deep learning (DL) and convolutional neural network (CNN) approaches have gained momentum. These are described in detail in section C.

B. Gaze Estimation from Low Resolution Images

To facilitate gaze tracking in everyday settings, the use of cheap, compact and easy-to-integrate webcams is commonly preferred. Unfortunately, webcams offer low resolution images (typically 640x480 pixels) resulting in very poor gaze estimation accuracy. Low resolution images have strong noise effects [34], and distortions in the eye region contours and eye features become indistinguishable under varying illumination levels, user distance, and movements. Therefore, several approaches have been developed to achieve high gaze accuracy from low quality images and are discussed in this subsection.

An early ANN approach was used to map gaze coordinates to low quality cropped eye images in [33]. The ANN used back propagation and 50 output units each for X and Y coordinate. It was trained with 2000 image/gaze position pairs. A hybrid approach is adopted in [35] in which the iris centres are determined first using circular a Hough transform, followed by refinement using a gradient-aware random sample consensus (RANSAC) algorithm and ellipse fitting. Eye corners are estimated using Gabor jets [36] and tracked using optical flow with normalized cross-correlation. Finally, the point of gaze (POG) is estimated from the iris center and eye corners using regression. A similar method is proposed in [34], where the problem arising due to the small size of cropped eye regions from low resolution images is overcome using 2D bilinear interpolation for reconstructing the eye image to a larger size for accurate tracing of the corneal reflection vector.

In [37], multiple miniature low resolution cameras positioned around a head-mounted setup are used. The gaze-mapping function is learned from multiple cameras using a 512 unit ANN, and trained on a large dataset of eye images. In [38], the eyeball and its movement direction are detected using a deformable angular integral search (DAISMI) method followed by a Deformable template-based 2D gaze estimation (DTBGE) algorithm used as a noise filter. [39] trains an appearance model using Singular Value Decomposition (SVD) and a set of eye region images expanded artificially by adding positioning errors. Then a third order tensor is estimated from the training images as the gaze direction and positioning vectors of these images. The SVD is trained and tested by extracting the gaze vector from the test images and comparing with that obtained from the training images.

C. Eye Gaze Estimation Using CNN's

Deep learning (DL) techniques have been successfully used in challenging conditions such as those with variable illumination, unconstrained backgrounds and free head motion. For example, [40] describes a calibration-free real-time CNN-based framework for gaze classification. Two CNNs, for the left and right eyes, are then trained independently to classify the gaze in seven directions. In [41], CNN-based gaze tracker for Augmented and Virtual reality devices achieves foveated rendering and gaze-contingent focus. The deep learning model is built to be robust to variations like skin and eye colour, illumination and occlusion. In [42], deep features are obtained from eye images using multi-scale convolutions and pooling for predicting gaze direction. This method uses minimized cross-entropy loss, coupled with Random Forest regression as a clustering algorithm. It classifies areas on a device screen according to gaze locations, and operates under natural illumination and head poses. [43] describes a novel, appearance-based gaze estimation method in which a CNN utilizes the full face image as input with spatial weights on the feature maps to suppress or enhance information in different facial regions. It achieves high accuracy and robust performance under varied illumination and extreme head poses. [44] achieves free head pose, 3D gaze tracking using two separate head pose and eye movement models with two CNNs, connected via a gaze transform layer. Finally, in [45] a CNN is built to learn the mapping between 2D head angle, eye image and gaze angle (output) using a small Lenet-inspired CNN. For testing, an extensive database is built with more than 200,000 images under variable illumination levels and eye appearances. This database (called MPII Gaze) is also used in this work and its further details are provided in the next section.

Several of the CNN-based works are specifically targeted towards gaze tracking in consumer/handheld devices such as [46], [47]. In [46], a CNN-based real time, calibration-free gaze estimation algorithm is presented. It is trained using a large and diverse dataset of eye images taken under variable lighting, head pose, and backgrounds captured from users through a smartphone app. Inputs to their CNN model include eye and face images. The location of the faces in the images are obtained through a face grid, which is used to infer relative eye and head poses. [47] presents a calibration-free method using Deep Belief Networks which classifiy gaze into a grid of nine gaze locations under various head-poses and viewing directions. In [48], a nine directional CNN-based gaze classifier is developed for a screen typing application, robust to false detections, blinks, and saccades (rapid, abrupt changes in fixation).

An overview of selected deep learning methods for gaze estimation can be seen in table I.

D. Related Work Utilizing the MPII Gaze Dataset

The MPII Gaze dataset[45] is large and challenging, containing images collected under a wide range of realistic scenarios, such as varied illumination levels, eye appearances and head poses. Use of MPII gaze dataset for training and testing gaze estimation algorithms can be found in their own paper [45], as also in [43], [49], [57], [58]. [45], which introduces the MPII Gaze dataset, uses a multimodal CNN for gaze estimation and reports a cross dataset test accuracy of 6 degrees. [43] uses full face (instead of eye only and multi-region) images with a CNN and achieves a person independent accuracy of 4.8 degrees on MPII Gaze while being robust to illumination variations and extreme head poses. In [49], gaze location over a block of screen area is tracked using a CNN, and the cross-subject performance is tested with MPII Gaze and that authors' own dataset. On MPII Gaze, the classification accuracy is poor (75.6%) compared to that on authors' dataset (92.5%). A Deep Regression Bayesian Network described in [58] achieves an accuracy of 7.1 degrees when tested on this dataset. In [59], [60], [61], the MPII Gaze dataset is used for

TABLE I
SELECTED CNN MODELS FOR EYE GAZE ESTIMATION

Citation	Dataset used	Network model used	Image Resolution	Accuracy	Special features
[41]	CAVE and Own dataset (cropped images of the eye and their respective gaze pixel coordinates)	LeNet (two convolutional layers and two pooling layers followed by a fully connected layer at the end). 21 classes for CAVE , 1829 classes for captured dataset.	28x28	6.7 degrees tested on CAVE as well as own collected dataset	Near eye tracking, operating under lighting changes and occlusions
[43]	MPII Gaze, UT Multiview	AlexNet. five convolutional layers, two fully connected layers. Additional linear regression layer on top of last fully connected layer.	448 x 448 (full face)	4.8 degrees on MPII Gaze, 6 degrees on UT dataset. Person-independent evaluation	Tolerates various illumination, gaze direction,
[42]	Own dataset with 107,681 images under different lighting, head movement, glasses.	Three convolutional layers with max- pooling layer, one single max-pooling layer, one hidden layer and one soft-max layer	40x70	5-7 degrees within dataset evaluation, but training and test set have different gaze angles.	Works under natural light with free head motion
[44]	Own dataset 200-subjects different head poses, eyeball movements, lighting conditions, glasses, occlusions, reflections.	AlexNet, BN-Inception network. Two CNNs to model head pose and eyeball motion. A gaze transform layer to aggregate them into gaze prediction	62x62	4.3 degrees cross subject evaluation	Allows free head motion
[45]	MPII Gaze, Eyediap, UT Multiview	LeNet architecture. A linear regression layer trained on top of fully connected layer. Multimodal CNN model to use eye image and head pose information	60x36	∼6 degrees Cross Dataset Evaluation	Variable appearance, illumination, head pose
[46]	Gaze Capture dataset	AlexNet + SVR	80x80	2.58cm (within dataset evaluation)	Tolerant pose, appearance, and lighting
[49]	Own dataset. 56 groups of eye videos, 181440 eye images from 22 subjects	3 convolutional layers followed by max-pooling layers, 2 fully connected and a soft-max layer for classification. 6 & 54 classes	40x72	6.375 deg, 69.3% Cross dataset evaluation	—-

comparing synthetic datasets and facial models. [59] presents a method for synthesizing a large set of variable eye region images with a generative 3D eye region model. Then a gaze estimation method using the k-Nearest-Neighbour algorithm) is tested on the synthetic data and the MPII Gaze dataset to achieve an accuracy of 9.95 and 9.58 degrees respectively. Another method for synthetic, labelled photo-realistic eye region image creation is described in [60] using head scan geometry. The generated dataset, along with MPII gaze, is used to test and compare the accuracy of a CNN based gaze estimation method. In [61], a facial behaviour analysis tool is developed that is capable of tracking gaze vectors using a Conditional Local Neural Fields (CLNF) framework by detecting eye region features like eyelids, irises and pupils. The tool is tested on MPII Gaze to achieve 9.96 degrees of accuracy in gaze estimation. In [62] a semi-supervised learning method is developed for improving the realism of simulated data and used to create refined training images for gaze estimation using CNNs. The CNNs are then tested on the MPII gaze dataset to achieve an error of 7.8 degrees. Apart from the above, the MPII dataset has been used for training a CNN for gaze estimation coupled with gaze target discovery in [63] and a cascaded-regressor-based eye center detector [64].

III. IMAGE AND VIDEO DATASETS FOR LOW RESOLUTION AND UNCONSTRAINED GAZE ESTIMATION

A survey of relevant publicly available gaze databases is summarized in Table II. In this survey, only the databases built for training and testing gaze estimation algorithms are listed, while other gaze databases, e.g. for studying saliency models, are not included as they are out of scope for this work.

IV. METHODS

The CNN-based gaze estimation methods in this work were evaluated on NVIDIA 1080 TI GPUs using python 2.7 and caffe 1.0 with accuracy and euclidean loss layers modified to calculate angle difference in radians. Person-exclusive, leave-one-out cross-validation was used in all experiments.

In eye gaze tracking literature it is common to use the word “accuracy” and “error” interchangeably and this can sometimes cause confusion to the reader. For this reason we use the word “error” in any case where the meaning could be unclear. All angles are reported in degrees. In this paper, error was determined as the average euclidean distance between the ground truth and predicted angles on the left-out, person-exclusive test set.

Multiple deep neural networks were compared for eye gaze estimation using deep neural networks. The publicly available MPII Gaze dataset was used for all experiments except for the first where the UT Multiview dataset is also used.

A. *MPII Gaze Dataset Details*

The MPII gaze dataset is a large collection of 213659 images captured under unconstrained conditions from 15 subjects over several days. The images are collected under multiple illumination conditions. Some of the subjects wear spectacles and some do not. The images were captured at various gaze

TABLE II
PUBLICLY AVAILABLE DATASETS FOR GAZE ESTIMATION

Name	**Per-sons**	**Items**	**Conditions**	**Resolution**	**Purpose**
MPII Gaze [45]	15	213,659 images.	Software running on subjects' laptops ask participants to look at a random 20 on-screen positions and confirm	unknown	Appearance-based gaze estimation in the wild.
UT Multiview [50]	50	64000 eye images	160 gaze directions per person were acquired using 8 cameras (views)	SXGA resolution (1280x1024)	Training and test data for appearance-based gaze estimation methods.
EYEDIAP [51]	16	94 sessions	Diversity of participants, head poses, gaze targets and sensing conditions. Screen or 3D objects. Data collection with Kinect for RGB and depth video streams	640x480 at 30 fps	Training and evaluation of gaze estimation approaches with robustness to pose, person.
Gaze Capture [46]	1474	2445504 images	Data captured with iphone/ipad using app. Large variation in pose, appearance, lighting. Variation in relative distance and orientation of the mobile device	unknown	Training CNNs for high accuracy calibration-free eye tracking on handheld devices under variable conditions.
TabletGaze [52]	51	100000 images	Video sequences recorded with tablet front-facing camera while subjects look at a dot on tablet screen. Unrestricted subject motion, each subject performed 4 body postures: standing, sitting, slouching, and lying.	1280x720	Mobile gaze dataset for studying unconstrained mobile gaze estimation
Weiden-bacher [53]	20	2220 images	Manual landmarks on pupils, eye corners, nose tip, mouth corners. Horizontal head rotations (0 to 90 in steps of 10), vertical head poses 0, 30, and 60 azimuth (-20, +20) elevation. For each head pose, nine different gaze conditions	1600x1200	Evaluating computational methods for head pose and eye gaze estimation
McMurrough [54]	20	120 sessions	Videos recorded eye motion as subjects look at, or follow a set of predefined points on a computer screen. Head position in 3D captured using a Vicon Motion Tracking System	768x480 pixels at a frame rate of 29.97 Hz	To be used as a benchmark for Point of Gaze (PoG) detection algorithms
OMEG [55]	50	40000 images	Eye images captured under multiple head poses, three fixed poses, 0 and 30 degree , and a free pose style.	1280 x 1024	Evaluating and comparing gaze tracking algorithms
HPEG [56]	10	20 videos	Subject faces camera frontally, free to move, background covers a big part of the image, with intense human action.	1280x960 pixels, 30 fps	Head pose and gaze estimation algorithm testing

angles, recorded by software running on the participant's laptops. In each session, the subjects were asked to look at random sequences of 20 onscreen positions and to confirm their attentiveness, the subjects were asked to press the space bar once the onscreen target was disappearing.

The dataset contains eye and head features and target (gaze angle) values for every participant. To use MPII Gaze, the authors suggest mapping their reported vector to angles using a Rodrigious transformation, and this has been done for all reported experiments.

V. Experiments and Results

In this section, multiple experiments are described to provide insight on 4 primary research questions. These are tested on multiple CNN architectures and are discussed in this section. One of the first research goals was to achieve state of the art test error on a network that could perform inference within 3-15 ms on a typical single proprietary low power consumer embedded device.

The specific research questions are:

1) How does an architecture that uses both eyes compare to one that uses one eye in terms of accuracy?
2) How does simulated camera distance impact eye gaze accuracy for the proposed model?
3) Can augmentation be used to reduce any negative impacts?
4) Can the proposed hardware-friendly architecture perform with sufficient accuracy and speed?

First, the intra-dataset, person-exclusive experiments from [45] were duplicated. The same procedure to estimate accuracy was used except the altered accuracy layers were modified to eliminate NaNs by replacing undefined values of the arc cosine function with the largest or smallest valid values as appropriate.

A. Approach 1: Analysis of Eye Flipping

In [45], one network is used for both eyes and one of the eyes is flipped so that the gaze angle is roughly correct. An experiment was designed to see if this flipping had an impact on model accuracy. Six experiments were performed using the UT and MPII-Gaze datasets to see if training on both eyes or just one eye impacted accuracy. By doing experiments with combined and non combined datasets, it was also possible to determine if they had similar distributions, and thus, if combining the two would be helpful for future experiments.

As shown in table III, the method of individually classifying eye images and simply adjusting the right eye and angles as used in [45] is a limiting factor in accuracy for that method. In both datasets, the performance was increased exclusively using

TABLE III
RESULTS OF APPROACH 1

Training set	Error
MPII Left eye only	5.9 degrees
MPII+UT Left eye only	5.6 degrees
MPII Both eyes	7.4 degrees
MPII+UT Both eyes	6.5 degrees
MPII Right eye only	5.3 degrees
MPII+UT Right eye only	6.1 degrees

left eyes or right eyes. This suggests that simply flipping the eye as suggested by [45] may be a source of error in their model.

These results also indicated that the distribution of MPII Gaze and UT-Multiview are sufficiently different that combining the two for training gives no or very little improvement. Because of this, it was decided to use only MPII Gaze for the remaining experiments in this section as UT-Multiview had no significant influence on error.

B. A New Approach: Dual Eye Channels

Given the problems identified in the previous subsection with flipping one of the eyes, and not wanting to use two different networks for reasons of efficiency, a new approach involving using both the left and right eyes in separate input channels was investigated. Specifically, the left eye and right eye images are passed to the network in channels 0 and 1 respectively, and the gaze and pose information are averaged between the left and right eye images to create a single gaze and pose vector. Due to the results in the previous section, which indicated that data from UT did not significantly impact the results, only the MPII-Gaze dataset was used.

This modified, two channel architecture resulted in a significant increase in accuracy, averaging 4.63 degrees of error between the target and predicted values on unseen individuals. A diagram of this network can be seen in Figure 1.

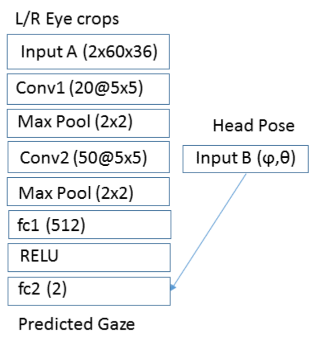


Fig. 1. Diagram of hardware CNN for initial eye gaze estimation tasks, heavily based on network from [45]

C. Can We Reduce the Number of Parameters?

Deep neural networks can often be made more efficient by reducing the number of parameters but this can sometimes come at the cost of accuracy. To see if reducing the number of parameters was possible without harming accuracy, an experiment was performed to halve the size of all output parameters. This experiment was not allowed to run for the full duration because the exact angle accuracy did not matter, only evidence that the network complexity could be reduced to a point where it would be small enough if necessary. This resulted in an average error of 4.980% on an unseen individual from the MPII Gaze dataset and indicates that reducing the number of parameters had little impact on accuracy.

TABLE IV
RESULT OF DISTANCE SIMULATION EXPERIMENTS

Resolution	Error
60 x 36	4.63 degrees
52 x 31	9.90 degrees
26 x 16	10.10 degrees

D. Multi Resolution Experiments

Eye gaze systems in consumer devices must be able to maintain accuracy at a large range of distances. Although MPII Gaze has some variability in distance from the camera, the distances are not realistic for the conditions expected in, for example, a driver monitoring system or a distant cell phone camera. Specifically, it was desired to accommodate realistic distances between the camera and the subject in situations that would be typical in commercial eye trackers that utilize low cost, low resolution cameras. To simulate the loss of information caused by distance, down-sampling was performed on the eye images in MPII-Gaze as follows:

- Input image 60 x 36 ->Downscale to 52 x 31 ->Upscale to 60 x 36 ->CNN Eye gaze angle
- Item Input image 60 x 36 ->Downscale to 26 x 16 ->Upscale to 60 x 36 ->CNN Eye gaze angle.

As can be seen in table IV, the network learned a narrow range of distances, and performance deteriorates when the subject is further from the camera than those in the training set. As a sanity check, an experiment was done to see if the Downscaling algorithm was at fault for the poor results, so in addition to nearest, we also tried bicubic, linear, and LANCZOS from using OpenCV. The experiment showed that the downscaling algorithm used had no influence on the results.

This demonstrates that the model is sensitive to changes in distance. In the next section, an experiment is performed to see if data augmentation can be used to improve upon this.

E. Impact of Random Resizing as Augmentation

Data augmentation has been shown in many studies to have a large impact on model performance.

To further improve accuracy, the dataset was augmented with multiple randomly chosen resolutions to match the full range of desired distances. To help reduce the chance that the network would learn the specific interpolation method used, Nearest is used in the training set, but Lanczos filtering is used in the testing set.

- Original resolution: 4.918 degrees error
- 60 x 36 ->52 x 31 ->60 x 36 : 4.94 degrees error

TABLE V
ERROR OF PROPOSED MODEL FOR VARIOUS RESOLUTIONS (DEGREES)

Resolutions	36 x 60	31 x 52	26 x 16
Best	3.650	4.1690	4.240
Second best	4.100	4.324	4.366
Benchmark	4.917	4.940	4.970

TABLE VI
COMPARISON OF PROPOSED MODEL WITH OTHER PUBLISHED WORKS ON THE MPII-GAZE DATABASE

Citation	Error (degrees)
[61]	9.96
[59]	9.58
[62]	7.8
[58]	7.1
[45]	6
[43]	4.8
Proposed	3.64

- 60 x 36 ->26 x 16 ->60 x 36: 4.97 degrees error

These results indicate that augmenting the images with distances that are likely to be encountered in real world usage situations is an effective way to increase accuracy and succeeds in achieving some invariance to subject distance.

F. A Quest for Hardware Efficiency and Even Better Accuracy

It was shown in [65] that two stacked layers of 3x3 convolutions has the same receptive field as a single 5x5 layer, with fewer multiplications. Due to this, one of the requirements was that kernel sizes be 3x3, this required retraining and slight redesign of the network. Several experiments involving architectures with stacked 3x3 kernels were performed using different parameters. The best two architectures were further evaluated, and the best models from each of them were chosen and evaluated on multiple resolutions as shown in table V. A diagram of the final architecture used can be seen in figure 2, and a comparison with other published works can be seen in table VI.

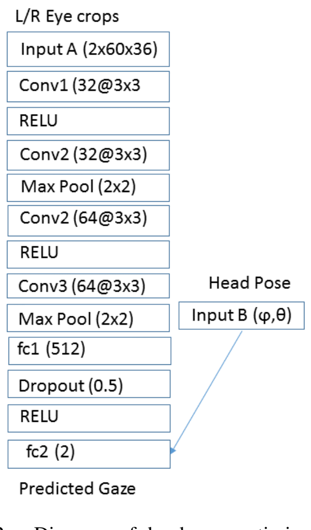


Fig. 2. Diagram of hardware optimized CNN for eye gaze estimation corresponding to the accuracies in table V.

VI. CONCLUSION

Our results show that using information from both eyes in the neural network can increase accuracy. This is demonstrated in section V, where adding additional eye information from the opposite eye enabled improved results over individual eyes, helping the network make sense of low quality images with ambiguous gaze. As expected, in all cases, the deeper network had the best performance. This research demonstrated the sensitivity of such models to variations in distance and how data augmentation can be used to overcome this. Most importantly, a new compact hardware-friendly architecture designed for use in small consumer electronics has been introduced and evaluated on the eye gaze task.

When evaluated on MPII Gaze, the proposed model performs favorably (see table VI even when compared with much larger networks in the literature.

Since augmentation resulted in a significant improvement in accuracy, it may be fruitful to try other types of augmentation such as Generative Adversarial Networks (GANS) with landmarks, [66] and Smart Augmentation (SA) [67] in future work. This will either require modifying such methods to work on regression problems or translating the eye gaze problem into a classification task for the purpose of generating augmented data[68] and then back to a regression task. Additionally, there are plans to investigate whether temporal information[69] can be used to further increase the accuracy without sacrificing the need for performance as it has been shown to increase performance in DMS systems.

ACKNOWLEDGMENT

This research is funded under the SFI Strategic Partnership Program by Science Foundation Ireland (SFI) and FotoNation Ltd. Project ID: 13/SPP/I2868 on Next Generation Imaging for Smartphone and Embedded Platforms. This work is also supported by an Irish Research Council Employment Based Programme Award. Project ID: EBPPG/2016/280.

REFERENCES

[1] Y. Liang, M. L. Reyes, and J. D. Lee, "Real-time detection of driver cognitive distraction using support vector machines," *IEEE Transactions on Intelligent Transportation Systems*, vol. 8, no. 2, pp. 340–350, June 2007.

[2] E. Wood and A. Bulling, "Eyetab: Model-based gaze estimation on unmodified tablet computers," in *Proceedings of the Symposium on Eye Tracking Research and Applications*, ser. ETRA '14. New York, NY, USA: ACM, 2014, pp. 207–210. [Online]. Available: http://doi.acm.org/10.1145/2578153.2578185

[3] A. Tawari and M. M. Trivedi, "Robust and continuous estimation of driver gaze zone by dynamic analysis of multiple face videos," in *2014 IEEE Intelligent Vehicles Symposium Proceedings*, June 2014, pp. 344–349.

[4] V. Vaitukaitis and A. Bulling, "Eye gesture recognition on portable devices," in *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, ser. UbiComp '12. New York, NY, USA: ACM, 2012, pp. 711–714. [Online]. Available: http://doi.acm.org/10.1145/2370216.2370370

[5] S.-W. Shih and J. Liu, "A novel approach to 3-d gaze tracking using stereo cameras," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 34, no. 1, pp. 234–245, Feb 2004.

[6] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[7] J. Lemley, S. Bazrafkan, and P. Corcoran, "Deep learning for consumer devices and services: Pushing the limits for machine learning, artificial intelligence, and computer vision." *IEEE Consumer Electronics Magazine*, vol. 6, no. 2, pp. 48–56, 2017.

[8] D. W. Hansen and Q. Ji, "In the eye of the beholder: A survey of models for eyes and gaze," *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 3, pp. 478–500, 2010.

[9] A. Kar and P. Corcoran, "A review and analysis of eye-gaze estimation systems, algorithms and performance evaluation methods in consumer platforms," *IEEE Access*, vol. 5, pp. 16 495–16 519, 2017.

[10] F. L. Coutinho and C. H. Morimoto, "Augmenting the robustness of cross-ratio gaze tracking methods to head movement," in *Proceedings of the Symposium on Eye Tracking Research and Applications*. ACM, 2012, pp. 59–66.

[11] S. Chen and C. Liu, "Eye detection using discriminatory haar features and a new efficient svm," *Image and Vision Computing*, vol. 33, pp. 68–77, 2015.

[12] H. chuan Lu, C. Wang, and Y. w. Chen, "Gaze tracking by binocular vision and lbp features," in *2008 19th International Conference on Pattern Recognition*, Dec 2008, pp. 1–4.

[13] P. Blignaut, "Mapping the pupil-glint vector to gaze coordinates in a simple video-based eye tracker," *Journal of Eye Movement Research*, vol. 7, no. 1, 2013. [Online]. Available: https://bop.unibe.ch/index.php/JEMR/article/view/2373

[14] Z. Zhu and Q. Ji, "Eye and gaze tracking for interactive graphic display," *Machine Vision and Applications*, vol. 15, no. 3, pp. 139–148, Jul 2004. [Online]. Available: https://doi.org/10.1007/s00138-004-0139-4

[15] C. Jian-nan, Z. Chuang, Y. Yan-tao, L. Yang, and Z. Han, "Eye gaze calculation based on nonlinear polynomial and generalized regression neural network," in *2009 Fifth International Conference on Natural Computation*, vol. 3, Aug 2009, pp. 617–623.

[16] J. Wang, G. Zhang, and J. Shi, "2d gaze estimation based on pupil-glint vector using an artificial neural network," *Applied Sciences*, vol. 6, no. 6, 2016. [Online]. Available: http://www.mdpi.com/2076-3417/6/6/174

[17] Chunfei, K.-A. Ma, B.-D. Choi, and S.-J. K. Choi, "Robust remote gaze estimation method based on multiple geometric transforms," *Optical Engineering*, vol. 54, pp. 54 – 54 – 7, 2015. [Online]. Available: https://doi.org/10.1117/1.OE.54.8.083103

[18] Z. Zhu, Q. Ji, and K. P. Bennett, "Nonlinear eye gaze mapping function estimation via support vector regression," in *18th International Conference on Pattern Recognition (ICPR'06)*, vol. 1, 2006, pp. 1132–1135.

[19] J. Zhu and J. Yang, "Subpixel eye gaze tracking," in *Proceedings of Fifth IEEE International Conference on Automatic Face Gesture Recognition*, May 2002, pp. 124–129.

[20] E. D. Guestrin and M. Eizenman, "General theory of remote gaze estimation using the pupil center and corneal reflections," *IEEE Transactions on Biomedical Engineering*, vol. 53, no. 6, pp. 1124–1133, June 2006.

[21] T. Ohno and N. Mukawa, "A free-head, simple calibration, gaze tracking system that enables gaze-based interaction," in *Proceedings of the 2004 Symposium on Eye Tracking Research & Applications*, ser. ETRA '04. New York, NY, USA: ACM, 2004, pp. 115–122. [Online]. Available: http://doi.acm.org/10.1145/968363.968387

[22] Z. Zhu and Q. Ji, "Novel eye gaze tracking techniques under natural head movement," *IEEE Transactions on Biomedical Engineering*, vol. 54, no. 12, pp. 2246–2260, Dec 2007.

[23] D. Beymer and M. Flickner, "Eye gaze tracking using an active stereo head," in *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, vol. 2, June 2003, pp. II–451–8 vol.2.

[24] T. Nagamatsu, J. Kamahara, and N. Tanaka, "Calibration-free gaze tracking using a binocular 3d eye model," in *CHI '09 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '09. New York, NY, USA: ACM, 2009, pp. 3613–3618. [Online]. Available: http://doi.acm.org/10.1145/1520340.1520543

[25] D. Model and M. Eizenman, "User-calibration-free remote eye-gaze tracking system with extended tracking range," in *2011 24th Canadian Conference on Electrical and Computer Engineering(CCECE)*, May 2011, pp. 001 268–001 271.

[26] K. Wang and Q. Ji, "Real time eye gaze tracking with kinect," in *2016 23rd International Conference on Pattern Recognition (ICPR)*, Dec 2016, pp. 2752–2757.

[27] X. Zhou, H. Cai, Z. Shao, H. Yu, and H. Liu, "3d eye model-based gaze estimation from a depth sensor," in *2016 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Dec 2016, pp. 369–374.

[28] Y.-L. Wu, C.-T. Yeh, W.-C. Hung, and C.-Y. Tang, "Gaze direction estimation using support vector machine with active appearance model," *Multimedia Tools and Applications*, vol. 70, no. 3, pp. 2037–2062, Jun 2014. [Online]. Available: https://doi.org/10.1007/s11042-012-1220-z

[29] C. M. Yilmaz and C. Kose, "Local binary pattern histogram features for on-screen eye-gaze direction estimation and a comparison of appearance based methods," in *2016 39th International Conference on Telecommunications and Signal Processing (TSP)*, June 2016, pp. 693–696.

[30] S. Chen and C. Liu, "Eye detection using discriminatory haar features and a new efficient svm," *Image Vision Comput.*, vol. 33, no. C, pp. 68–77, Jan. 2015. [Online]. Available: http://dx.doi.org/10.1016/j.imavis.2014.10.007

[31] Y. Li, X. Xu, N. Mu, and L. Chen, "Eye-gaze tracking system by haar cascade classifier," in *2016 IEEE 11th Conference on Industrial Electronics and Applications (ICIEA)*, June 2016, pp. 564–567.

[32] T. Schneider, B. Schauerte, and R. Stiefelhagen, "Manifold alignment for person independent appearance-based gaze estimation," in *Proceedings of the 2014 22Nd International Conference on Pattern Recognition*, ser. ICPR '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 1167–1172. [Online]. Available: http://dx.doi.org/10.1109/ICPR.2014.210

[33] S. Baluja and D. Pomerleau, "Non-intrusive gaze tracking using artificial neural networks," in *Advances in Neural Information Processing Systems*, 1994, pp. 753–760.

[34] Y. Fu, W. P. Zhu, and D. Massicotte, "A gaze tracking scheme with low resolution image," in *2013 IEEE 11th International New Circuits and Systems Conference (NEWCAS)*, June 2013, pp. 1–4.

[35] A. George and A. Routray, "Fast and accurate algorithm for eye localisation for gaze tracking in low-resolution images," *IET Computer Vision*, vol. 10, no. 7, pp. 660–669, 2016.

[36] D. Gonzalez-Jimenez and J. L. Alba-Castro, "Shape-driven gabor jets for face description and authentication," *IEEE Transactions on Information Forensics and Security*, vol. 2, no. 4, pp. 769–780, Dec 2007.

[37] M. Tonsen, J. Steil, Y. Sugano, and A. Bulling, "Invisibleeye: Mobile eye tracking using multiple low-resolution cameras and learning-based gaze estimation," *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 1, no. 3, pp. 106:1–106:21, Sep. 2017. [Online]. Available: http://doi.acm.org/10.1145/3130971

[38] Y.-T. Lin, R.-Y. Lin, Y.-C. Lin, and G. C. Lee, "Real-time eye-gaze estimation using a low-resolution webcam," *Multimedia Tools Appl.*, vol. 65, no. 3, pp. 543–568, Aug. 2013. [Online]. Available: http://dx.doi.org/10.1007/s11042-012-1202-1

[39] Y. Ono, T. Okabe, and Y. Sato, "Gaze estimation from low resolution images," in *Advances in Image and Video Technology*, L.-W. Chang and W.-N. Lie, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 178–188.

[40] A. George and A. Routray, "Real-time eye gaze direction classification using convolutional neural network," in *2016 International Conference on Signal Processing and Communications (SPCOM)*, June 2016, pp. 1–5.

[41] R. Konrad, S. Shrestha, and P. Varma, "Near-eye display gaze tracking via convolutional neural networks."

[42] Y. Wang, T. Shen, G. Yuan, J. Bian, and X. Fu, "Appearance-based gaze estimation using deep features and random forest regression," *Know.-Based Syst.*, vol. 110, no. C, pp. 293–301, Oct. 2016. [Online]. Available: https://doi.org/10.1016/j.knosys.2016.07.038

[43] X. Zhang, Y. Sugano, M. Fritz, and A. Bulling, "It's written all over your face: Full-face appearance-based gaze estimation," in *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, July 2017, pp. 2299–2308.

[44] H. Deng and W. Zhu, "Monocular free-head 3d gaze tracking with deep learning and geometry constraints," in *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct 2017, pp. 3162–3171.

[45] X. Zhang, Y. Sugano, M. Fritz, and A. Bulling, "Appearance-based gaze estimation in the wild," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015, pp. 4511–4520.

[46] K. Krafka, A. Khosla, P. Kellnhofer, H. Kannan, S. Bhandarkar, W. Matusik, and A. Torralba, "Eye tracking for everyone," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 2176–2184.

[47] H. Park and D. Kim, "Gaze classification on a mobile device by using deep belief networks," in *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, Nov 2015, pp. 685–689.

[48] C. Zhang, R. Yao, and J. Cai, "Efficient eye typing with 9-direction gaze estimation," *Multimedia Tools and Applications*, Nov 2017. [Online]. Available: https://doi.org/10.1007/s11042-017-5426-y

[49] X. Wu, J. Li, Q. Wu, and J. Sun, "Appearance-based gaze block estimation via cnn classification," in *2017 IEEE 19th International Workshop on Multimedia Signal Processing (MMSP)*, Oct 2017, pp. 1–5.

[50] Y. Sugano, Y. Matsushita, and Y. Sato, "Learning-by-synthesis for appearance-based 3d gaze estimation," in *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, ser. CVPR '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 1821–1828. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2014.235

[51] K. A. Funes Mora, F. Monay, and J.-M. Odobez, "Eyediap: A database for the development and evaluation of gaze estimation algorithms from rgb and rgb-d cameras," in *Proceedings of the Symposium on Eye Tracking Research and Applications*, ser. ETRA '14. New York, NY, USA: ACM, 2014, pp. 255–258. [Online]. Available: http://doi.acm.org/10.1145/2578153.2578190

[52] Q. Huang, A. Veeraraghavan, and A. Sabharwal, "Tabletgaze: dataset and analysis for unconstrained appearance-based gaze estimation in mobile tablets," *Machine Vision and Applications*, vol. 28, no. 5, pp. 445–461, Aug 2017. [Online]. Available: https://doi.org/10.1007/s00138-017-0852-4

[53] U. Weidenbacher, G. Layher, P. M. Strauss, and H. Neumann, "A comprehensive head pose and gaze database," in *2007 3rd IET International Conference on Intelligent Environments*, Sept 2007, pp. 455–458.

[54] C. D. McMurrough, V. Metsis, J. Rich, and F. Makedon, "An eye tracking dataset for point of gaze detection," in *Proceedings of the Symposium on Eye Tracking Research and Applications*, ser. ETRA '12. New York, NY, USA: ACM, 2012, pp. 305–308. [Online]. Available: http://doi.acm.org/10.1145/2168556.2168622

[55] Q. He, X. Hong, X. Chai, J. Holappa, G. Zhao, X. Chen, and M. Pietikäinen, "Omeg: Oulu multi-pose eye gaze dataset," in *Image Analysis*, R. R. Paulsen and K. S. Pedersen, Eds. Cham: Springer International Publishing, 2015, pp. 418–427.

[56] S. Asteriadis, D. Soufleros, K. Karpouzis, and S. Kollias, "A natural head pose and eye gaze dataset," in *Proceedings of the International Workshop on Affective-Aware Virtual Agents and Social Robots*, ser. AFFINE '09. New York, NY, USA: ACM, 2009, pp. 1:1–1:4. [Online]. Available: http://doi.acm.org/10.1145/1655260.1655261

[57] S. D. Iyer and H. Ramasangu, "Hybrid lasso and neural network estimator for gaze estimation," in *2016 IEEE Region 10 Conference (TENCON)*, Nov 2016, pp. 2579–2582.

[58] S. Nie, M. Zheng, and Q. Ji, "The deep regression bayesian network and its applications: Probabilistic deep learning for computer vision," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 101–111, Jan 2018.

[59] E. Wood, T. Baltrušaitis, L.-P. Morency, P. Robinson, and A. Bulling, "Learning an appearance-based gaze estimator from one million synthesised images," in *Proceedings of the Ninth Biennial ACM Symposium on Eye Tracking Research & Applications*, ser. ETRA '16. New York, NY, USA: ACM, 2016, pp. 131–138. [Online]. Available: http://doi.acm.org/10.1145/2857491.2857492

[60] E. Wood, T. Baltruaitis, X. Zhang, Y. Sugano, P. Robinson, and A. Bulling, "Rendering of eyes for eye-shape registration and gaze estimation," in *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ser. ICCV '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 3756–3764. [Online]. Available: http://dx.doi.org/10.1109/ICCV.2015.428

[61] T. Baltrusaitis, P. Robinson, and L. P. Morency, "Openface: An open source facial behavior analysis toolkit," in *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, March 2016, pp. 1–10.

[62] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb, "Learning from simulated and unsupervised images through adversarial training," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017, pp. 2242–2251.

[63] X. Zhang, Y. Sugano, and A. Bulling, "Everyday eye contact detection using unsupervised gaze target discovery," in *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, ser. UIST 2017. New York, NY, USA: ACM, 2017, pp. 193–203. [Online]. Available: http://doi.acm.org/10.1145/3126594.3126614

[64] S. J. Baek, K. A. Choi, C. Ma, Y. H. Kim, and S. J. Ko, "Eyeball model-based iris center localization for visible image-based eye-gaze tracking systems," *IEEE Transactions on Consumer Electronics*, vol. 59, no. 2, pp. 415–421, May 2013.

[65] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[66] S. Bazrafkan, H. Javidnia, and P. Corcoran, "Face synthesis with landmark points from generative adversarial networks and inverse latent space mapping," *arXiv preprint arXiv:1802.00390*, 2018.

[67] J. Lemley, S. Bazrafkan, and P. Corcoran, "Smart augmentation learning an optimal data augmentation strategy," *IEEE Access*, vol. 5, pp. 5858–5869, 2017. [Online]. Available: https://doi.org/10.1109/ACCESS.2017.2696121

[68] ——, "Learning data augmentation for consumer devices and services," in *Consumer Electronics (ICCE), 2018 IEEE International Conference on*. IEEE, 2018, pp. 1–3.

[69] ——, "Transfer learning of temporal information for driver action classification," in *The 28th Modern Artificial Intelligence and Cognitive Science Conference (MAICS)*, 2017.

Joseph Lemley Joseph Lemley received a B.S. degree in computer science and the Masters degree in computational science from Central Washington University in 2006 and 2016, respectively. He is currently pursuing a Ph.D. with the National University of Ireland Galway. His field of work is machine learning using deep neural networks for tasks related to computer vision. His Ph.D. is funded by FotoNation, Ltd., under the IRCSET Employment Ph.D. Program.

Anurada Kar Anuradha Kar is currently pursuing the Ph.D. degree with the National University of Ireland Galway and the Center for Cognitive, Connected, and Computational Imaging. Her research interests include human computer interaction and computational imaging. She is involved in eye gaze tracking-addressing the issues of accuracy and performance evaluation of gaze estimation systems in various platforms.

Alexandru Drimbarean Alexandru Drimbarean is the Vice President of Advanced Research team at FotoNation Ireland focusing on developing innovative computer vision and machine learning technologies for mobile, biometrics and automotive applications. Alexandru received his B.S in Electronic Engineering in Brasov Romania followed by an M.Sc. in Electronic Science at N.U.I Galway in 2002. His interests include image processing and understanding as well computer vision and machine learning. Alexandru has authored several journal articles as well as more than 30 patents.

Peter Corcoran Peter Corcoran is a Fellow of IEEE, past Editor-in-Chief of IEEE Consumer Electronics Magazine and a Professor at NUI Galway. His research interests include biometrics, imaging, deep learning, edge-AI and consumer electronics. He is co-author on 350+ technical publications and co-inventor on more than 300 granted US patents. In addition to his academic career, he is an occasional entrepreneur, industry consultant and compulsive inventor.

Appendix E

Eye Tracking in Augmented Spaces: a Deep Learning Approach

Eye Tracking in Augmented Spaces: a Deep Learning Approach

Joseph LEMLEY, Anuradha KAR and Peter CORCORAN, *Fellow, IEEE*
Center for Cognitive, Connected & Computational Imaging, College of Engineering & Informatics,
NUI Galway, Galway, Ireland
Email: { j.lemley2, a.kar2, peter.corcoran}@nuigalway.ie

***Abstract*—The use of deep learning for estimating eye gaze in augmented spaces is investigated in this work. There are two primary ways of interacting with augmented spaces. The first involves the use of AR/VR systems; the second involves devices that respond to the user's gaze directly. This domain can overlap with AR/VR environments but is not exclusive to them and contains its own unique set of issues. Deep learning methods for eye tracking that are capable of performing with minimal power consumption are investigated for both problems.**

Keywords: Augmented reality, Virtual reality, gaze estimation, deep learning, convolutional neural networks, smart spaces

I. INTRODUCTION

Eye gaze estimation is one of the most challenging frontiers of deep learning (DL) research in vision tasks and one of the few areas where conventional approaches are still dominant [1]. Recently, deep learning has been able to surpass conventional approaches in difficult gaze estimation tasks such as in DMS systems or handheld devices [2][3]. The strength of DL lies in inferring gaze, or performing subtasks in more complex gaze detection systems, even with poor image quality. Further, with novel data augmentation techniques as [4], DL has prospects to achieve good classification abilities using smaller network sizes and hardware-friendly architectures, which make it attractive for use in consumer electronic devices.

Augmented spaces are defined as any physical space with additional sensory or display elements that augment the physical function/utility/purpose of that space. These are achieved through the use of sensors, actuators, computing and networking devices incorporated within the space that enable recognition of humans, their activities and gestures in real time [5][6]. Examples may include a television that reduces power to screen when no one is looking at it, automatic room brightness adaptation based on human presence or motion sensor based security systems [7] .

With respect to eye tracking in augmented spaces, there are the two types of configurations to consider when designing gaze based interactions. The first type involves smart glasses or head mounted AR/VR systems. In these, the gaze tracking task is equivalent to that in conventional AR/VR applications where high quality close-up images of the eye are used and the subject interacts with nearby objects at least partly through the AR/VR system. The second scenario involves individual smart devices and appliances each with their own camera systems that can respond to gaze. If no one is watching the TV or digital readout, why should it operate at full power? Useful information and controls can be displayed only when someone is actively paying attention to them. These systems typically need to track the eye movements of everyone in their vicinity, in real time and with varying lighting conditions and user distances. Gaming is another domain which benefit from gaze information[8]. For example, if a player is momentarily distracted and not looking at the screen, it would be a bad time to introduce a crucial plot element or battle. Gaze information can also be used to provide an element of surprise in dynamic scenes by spawning enemy NPC (Non player characters) away from the players current gaze. However, the distance between the eye tracker camera and the player, and player head movements are significant factors determining accurate gaze tracking in gaming applications. AR/VR gaming systems typically have close eye facing cameras and are able to capture detailed eye images, whereas gaze tracking for games on remote game consoles, cell phones and other devices typically have cameras at a distance and varying user head movements, making eye tracking challenging for them.

In this paper we examine the use of deep learning for gaze tracking in AR/VR as well as remote setups for augmented environments. In the first case, DL is still behind conventional methods which can achieve accuracies as high as 0.5 degrees [9]. In the second case however, DL is emerging as the most feasible solution due to its ability to make sense of eye images that are too low quality for conventional approaches[10].

The paper is organized as follows: in section II.A, eye gaze estimation in augmented and virtual reality (AR & VR respectively) applications is explained. Gaze information has a significant impact in AR/VR environments [11][12], where gaze directions and gaze based functions are used to make user experiences more immersive, natural and effortless. In section II B gaze estimation in pure augmented spaces, without the use of AR/VR systems is explained. These include the tracking and application of gaze information for domotic controls, multimedia communications or assisted living systems [13].

The use of deep learning for gaze estimation in augmented spaces is primarily an unexplored area of research and dedicated network models or datasets for gaze estimation in such environments are not publicly available. Therefore in Section III, we describe our methodologies of using convolutional neural networks (CNN) and two different gaze datasets- one captured using a head-mounted eye tracker and another with normal remote setup to test our algorithm performance. The description of the CNN model, two datasets and proposed approaches pertaining to the two types of eye gaze estimation methods for augmented spaces are presented, followed by discussions and conclusion in Section IV.

II. Eye tracking in augmented spaces: prospects & challenges

A. *Eye tracking in AR/VR applications*

The significance of tracking user eyes in an AR/VR environment has been recently acknowledged in literature. In Augmented Reality applications, gaze information is fused with data from a scene camera to estimate the point of gaze of a user, for applications such as reading and document retrieval as described in [9]. In this work, eye tracking is used to identify the part of a document the user is reading and display relevant information on the see-through head mounted display (HMD). Gaze information, coupled with other eye movement features like dwell time and blinks can be used for object selection [12] and zooming and capturing snapshots in AR devices [14]. Gaze has also been used for wearable context aware messaging service in [15] and for attention guidance using peripheral vision in AR headsets in [16].

In Virtual reality (VR) research, eye movements can be used for interaction such as in [17]. [18] presents an immersive 3D VR interface with gaze based interactions such as menu selection and gaze directed typing of mails using a virtual keyboard. Realistic rendering of characters and social interaction between a user and virtual characters may be established by combining dynamic facial appearances and gaze directions as described in [19]. Other purposes of using gaze in VR include achieving wide view panoramas, foveated rendering and natural exploration of virtual environment [20].

B. *Eye gaze tracking in non-AR/VR augmented spaces*

Intelligent environmental controls using eye gestures fall into this category. In the consumer device domain, there has been developments of gaze controlled TV [21] which senses gaze to enable screen brightness variations, menu selection and understanding user program preferences. For assisted living applications, gaze based control of wheelchairs[22] and home appliances have been proposed[23], and also eye tracking as diagnostic technology for patients with disabilities[24].

TABLE I. SUMMARY OF SOME RECENT WORKS USING EYE GAZE IN AUGMENTED SPACES

Citation	Type	Setup	Applications	Gaze accuracy
[9]	AR/VR	HMD and eye tracker	Assisting reading activity	0.5 degrees
[14]	AR/VR	HMD with eye tracker and lenses	Eye directed zooming	0.5 degrees
[17]	AR/VR	HMD with eye tracker	Gaze based object selection	0.6 degrees
[18]	AR/VR	HMD with head and eye trackers	Gaze based multimedia use	1 degree
[19]	AR/VR	3D Stereo Rig with remote eye tracker	Gaze-aware facial re-enactment in VR	1.5 degrees
[21]	Non-AR/VR	Remote tracker with camera, LED, lens	TV display input and control	1.32 degrees
[22]	Non-AR/VR	Remote tracker	Wheelchair motion control with gaze	0.5 degrees
[23]	Non-AR/VR	Camera	Appliances detect and respond to gaze	--

C. *Challenges of eye tracking in augmented spaces*

Eye tracking in AR/VR headsets may face significant and unique challenges as described in [25]. These include motion and depth of field blur, latency, rapid calibration drifts while following smooth pursuit eye movements and loss in tracking due to varying orientation of the eye camera with respect to the eye. Complicated system design, bulky processing units and high power consumption may also limit the usability of these devices in the consumer electronics domain.

Major problems in using remote gaze trackers for estimating gaze as an input modality arise from the issue of Midas touch [26], difficulty in handling complex tasks by gaze gestures and expensive hardware.

III. Deep learning for Gaze estimation in Augmented Spaces.

A. *Concept and approach*

Deep learning has been successful in achieving good accuracy in remote gaze estimation problems [27]. However, implementing DL based eye tracking in AR/VR is difficult due to lack of publicly available datasets built for gaze tracking in AR/VR environments. In this work, gaze estimation for both the eye tracking scenarios in an augmented space is approached. For the AR/VR problem, we start by training a DL model with eye images captured using a head-mounted eye tracker. We then progressively introduce complex variations in the images typical to those faced by eye trackers in AR/VR environments. For the non-AR/VR eye tracking condition, we use a low resolution gaze dataset, which typically has the characteristics of images captured by remote eye trackers.

B. *Eye datasets used*

Since augmented spaces can be experienced through either AR/VR or through smart devices that are farther from the users eyes, it is necessary to use separate datasets to train and test the two DL methods. In this section we describe two gaze datasets: a high resolution dataset (for AR/VR) and low resolution datasets for gaze tracking on smart devices having their own cameras.

1) *High resolution datasets for AR/VR*

There is a dearth of public datasets from which deep learning systems can be trained (or evaluated) for eye gaze estimation utilizing head mounted eye-facing cameras. The only suitable publically available dataset found is the one developed by McMurrough et al called the "Point of Gaze (PoG) Eye Tracking Dataset"[28]. Unfortunately, this dataset only has images of the right eye and therefore may not be used for AR applications where knowing what a person is looking at in 3D space involves calculating the intersection of two gaze vectors. This information is still useful because there are typically only a few objects that collide with a given gaze vector that are within a person's field of view and these can all be assumed to be the gaze target in an augmented or virtual space. Despite this limitation, the PoG Eye Tracking Dataset is the most suitable publically available dataset captured with a head-mounted eye tracker and therefore used in this paper.

To create the dataset, twenty participants (18 men, two women) were asked to track target points on a video display while wearing an Applied Science Laboratories Mobile Eye™ infrared monocular recording device. The participants' right eye is centered in the video frame and is annotated for specific target points. The dataset is composed of 20 subjects with ages ranging from 21 to 54. The dataset is annotated with head gaze and eye pose information. Eye images are recorded with a resolution of 768 X 480 pixels at 29.97 Hz frame rate.

2) Low resolution datasets for smart devices.

Datasets for low resolution images are more common, for example the MPII-Gaze dataset[29], the UT Multiview dataset[30], and the Gaze Capture[3] dataset are all suitable for training appearance based Deep neural networks. That said, all of these methods come from different distributions and sometimes have incompatible annotations.

For experiments with non-AR/VR augmented spaces where the camera may be far from the user we use the MPII-Gaze dataset[29]. MPII-Gaze was captured over several days and includes annotations for 15 subjects in unconstrained situations in multiple illumination conditions and gaze angles. A Rodrigues transformation is used to map gaze vectors to angles.

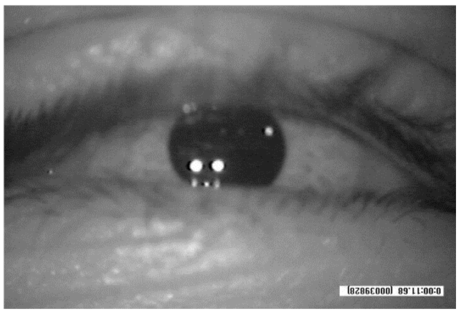


Fig 1. Example image from the McMurrough dataset [28], showing subject's right eye. This is typical of AR/VR systems that utilize eye facing cameras to estimate gaze.

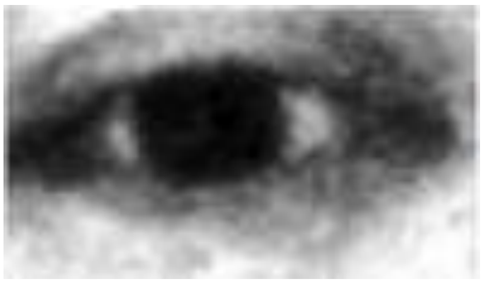


Fig 2. Example image from the MPII-Gaze dataset [29], showing subject's right eye typical of gaze tracking systems that use low quality cameras that are further from the subject.

C. Deep Learning model

Deep learning models such as those proposed by [31] have suggested architectures that work well for low resolution eye gaze estimation systems but these approaches may not be as suitable for high quality eye images as traditional approaches. In this paper, the Deep learning approach is developed for high resolution, close up images of the human eye and compared with one or more traditional approaches.

The inputs to the convolutional neural networks are eye frames from the PoG Eye Tracking Dataset scaled down to 157X96.

Two architectures are developed and tested. The first consist of 5 (3x3) convolutional layers with RELU activation functions, followed by one or more fully connected layers and are trained to perform a regression task, predicting gaze targets given a head pose and eye image as inputs. Next a Resnet 18-like [32] architecture are trained and compared with the previous model. Lastly, the two above models are compared with a traditional approach, thus providing information about how an appearance-based end-to-end deep learning approach compares with the best alternative.

TABLE II. SUMMARY OF EYE TRACKING METHODS USING LOW RESOLUTION IMAGES

Citation	**Input resolution/ eye crop size**	**Details of method used**	**Accuracy**	**Tolerance**
[33]	15x40	*ANN based, trained with 2000 images*	*1.7 deg*	---
[10]	640 × 480 (30 fps video)	*Iris center detection, ellipse fitting, eye corner detection*	1.33 deg	moderate head movements
[34]	40x30	Image reconstruction using bilinear interpolation, Hough transform	1.89 deg	---
[35]	640×480 (16 fps video)	Fourier descriptors of eye shape, classification with SVM	*90% accuracy*	Head motion lighting variations
[36]	5 × 5	Images from multiple miniature low resolution eye cameras with ANN	*2.25 deg*	Head motion

D. Deep learning for Eye tracking non-AR/VR applications

Non AR/VR approaches to eye gaze estimation require solutions that can make use of low quality eye images taken at a distance (figure 2) from the user. Traditional gaze estimation methods are unable to perform reliably in this task for unconstrained situations (i.e., outside of lab settings or carefully measured environments). Because traditional methods fail in such situations, approaches based on Deep Learning have recently become popular. These methods use convolutional neural networks, which are can generally be expected to at the level of a human expert on vision tasks when given enough samples.

In augmented spaces, such networks have the additional requirement of real time execution which involves a tradeoff between network complexity and power consumption.

Table II contains examples of networks that have been used for this task. Also it gives an idea about the existing methods that work on low resolution images and their typical accuracy.

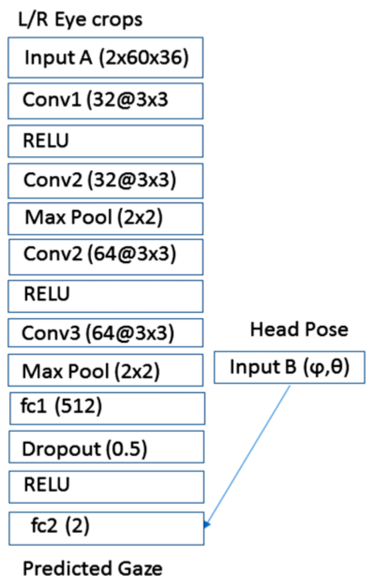


Fig. 3 Architecture of one proposed network for [18]

IV. Experiments & Methodology

A. Data pre-processing

In this paper, the major focus is the presentation of experiments and results from deep learning based gaze estimation using the "near eye" gaze and eye image data provided by the McMurrough et al dataset[28], that is typical of AR/VR devices as described above. The other case of remote gaze estimation using deep learning that uses the MPII Gaze dataset has been discussed in detail in other other work[31] and will be presented during the conference.

The McMurrough (or PoG) dataset provides eye images (video frames), gaze target coordinates as well as head pose information for twenty users which are used as inputs in the experiments for this work. In first half of the dataset, the users are asked to keep their head still while in the other, free head movement is allowed. Based on this, there are several experiments done in this work, some of which are described below and the others will be described and compared in the camera ready version as this is an ongoing work. For the first experiments, data of users with limited head movement is used and in the next phase of experiments, data having free user head movements will be used and results will be compared.

B. Experiment details

A series of experiments were conducted to determine appearance based convolutional neural network architectures that show the most promise for future exploration. In this section a summary of these experiments is provided. All experiments were trained to perform a regression task, mapping the pixels from the camera facing eye crops to the x and y coordinates that were being gazed upon. Eye crops were reduced to 1/4th their original size. All Xception[37] experiments used the Adam optimizer whereas all the other experiments used Stochastic Gradient Descent(SGD). This was because the Xception models failed to converge with SGD while the other networks suffered poor convergence with Adam. Images from the first 17 people in the dataset were used for training while the remaining were reserved as a test set. To the best of our knowledge this is the first work that utilized the Xception model for AR/VR eye gaze situations.

It may be noted that in this work, all input gaze locations values are uniformly scaled between 0 and 1 based on the maximum value of x or y in the labels, Therefore, to get the average distance in pixel space for the outputs x or y, the respective network output values must be multiplied by 1366.0 which is the maximum label value in the dataset. The accuracy in pixels from the method can therefore be stated as:

$$\text{x (or y pixel deviation)} = \text{output x (or y)}*1366 \quad (1)$$

As mentioned in the paper describing the PoG dataset[28], the monitor where gaze of participants was tracked has a 32 inch screen and the estimated dot pitch is 0.5mm. Hence the gaze tracking accuracy results from our algorithm using this dataset may be estimated as the deviation from the target location in millimeters (mm) as:

$$\text{x (or y) deviation in mm} = \text{output x(or y)} * 1366 * 0.5 \quad (2)$$

C. Experiment 1

In this experiment a network utilizing 5x5 kernels and RELU activation functions with 3 convolutional layers, separated by max pooling layers, followed by a fully connected RELU layer and dropout was used. The first convolutional layer had 15 features, the second convolutional layer had 30 features, and the final convolutional layer had 10. This network was trained for 10 epochs.

D. Experiment 2

In this experiment a model utilizing a decreasing number of units in each layer was used. The first 3 layers use 128 3x3 convolution followed by RELU activation functions. These 3 layers are followed by a dropout layer and a max pooling layer. The next block of layers consists of 2 convolutional layers with 64 units, using 3x3 kernels and RELU activation functions. This is again followed by the dropout and max pooling technique before the final block of convolutional layers. This final block of 2 convolutional layers has 32 3x3 kernels with RELU. Finally, a max pooling layer which leads to a fully connected RELU layer with 1024 units before being passed to a final linear layer with 2 outputs (x and y). This network was trained for 10 epochs.

E. Experiment 3

A very small network was developed with a similar architecture to the first experiment. Eight 7x7 convolutional kernels were in the first layer, followed by RELU and max pooling layers. The next convolutional layer had 16 units with 5x5 convolutional kernels, RELU activations and another max pooling layer. The final convolutional layer had eight 5x5 convolutional units with RELU and max pooling. Finally, a fully connected RELU layer and a dropout layer is used before the final linear layer with 2 outputs (x and y). This network was trained for 10 epochs.

F. *Xception experiments*

This subsection details the experiments performed with the Xception network. For these experiments the last two layers were removed and replaced by a fully connected RELU layer with 1024 units followed by a linear layer of 2 units (x,y). Due to the complexity and size of this architecture it is not reproduced in this paper. Instead we refer readers to the relevant literature[37]. In these experiments the modified Xception network was trained for 10,20,40, 60 and 100 epochs with a learning rate of 0.0001 with the Adam Optimizer in Keras[38].

V. RESULTS

The results from the experiments described above are presented in the subsections below. The results are mentioned both as direct outputs from the network as well as in units of centimeter, obtained using the calculations in Section IV. B.

A. Results from experiment 1

Surprisingly, experiment 1, despite being the second smallest network performed competitively with Xception at 10 epochs of training. This experiment resulted in a x axis error of 0.195970613497 (or 13.3 cm) and an y axis error of 0.124699373411 (8.5cm).

B. *Results from experiment 2*

Despite being significantly larger, this network underperformed the one from Experiment 1, with an average x error of 0.231945403853 (15.8 cm) and average y error of 0.12801038554 (8.7 cm). Interestingly, while the x error was less than that from Experiment 1, the y error is very similar

C. *Results from experiment 3*

This experiment resulted in an average error of 0.318149438847 (21 cm) on x and 0.180763828216 on the y axis (12.3 cm). This small network helps to establish a lower bound on the number of layers and units for reasonable results indicating that attempting to train networks smaller than this on this dataset is not likely to succeed. Still the similarity of this network to the one in Experiment 1 mean that only a few more neurons are sufficient to greatly increase the accuracy.

D. Results of Xception experiments

The first Xception experiment followed the pattern from the previous experiments with just 10 epochs and a learning rate of 0.0001, resulting in an x error of 0.280041239485 (19 cm) and a y error of 0.108832461737 (7.4 cm) . Increasing this to 20 epochs resulted in 0.196193765786 (13.4 cm) error on x and 0.0976574753508 (6.6 cm) error on y. Running an additional 40 epochs with a reduced learning rate (0.00001) did not show improvement but instead resulted in an x axis error of 0.223783574125 (15 cm) and an y axis error of 0.114304279187 (7.8 cm).

Finally, the second best results were obtained after 60 epochs at a 0.0001 learning rate. These results were: 0.135601494699 (9.2 cm) on x and 0.0618490625694 (4.2 cm) on y. The best results were from using 100 epochs with an average x error of 0.0935291282692 (6.3 cm) and average y error of 0.0466684513451 (3.1 cm).

VI. SUMMARY & DISCUSSIONS

It is noticed that in all experiments done with CNN models and the PoG gaze dataset, the error on the y direction is lower than on the x, although we do not currently have an explanation for this. It may result from the difference in the monitor's (where the user gaze was captured during collection of the PoG dataset) horizontal and vertical resolution but this is not sufficient to explain the accuracy differences between horizontal and vertical predations.

In this dataset, roughly half of participants wear glasses so it remains a matter of investigation as to if this factor has any impact on the result.

One consideration when comparing or evaluating these results is that the average accuracy includes frames where eyes are not open. This is a deliberate choice, as CNNs may be capable of estimating gaze of even closed eyes therefore comparing these results with those that disregard closed or partly open eyes would be misleading. An expanded CNN may utilize temporal information to increase the accuracy of these results as in [39].

This research is ongoing and currently more experiments on the near eye dataset are being implemented. Further results will be presented partly in the camera ready version of this paper and full details during the conference. These include studying the impact of head pose on eye tracking accuracy and other variable conditions like motion and camera defocus blur, extreme eye poses due to camera position drift and results on smooth pursuit movements which are commonly faced by eye tracking algorithms in AR/VR systems. The results for CNN based remote eye tracking using data from MPIIGaze dataset are promising and will be presented during the conference.

VII. CONCLUSION

This paper evaluates the use of deep neural networks for end to end mapping of eye images to gaze coordinates with applications in augmented spaces. The eye images are sourced from two small low cost eye facing cameras: one for the left and one for right eye in the case of AR/VR type systems. For the case of gaze tracking in augmented spaces that do not involve AR/VR, images from one or more cameras at a greater distance are used as input.

As deep learning begins to surpass traditional techniques in many eye gaze tasks, it is of interest to investigate its' potential for gaze estimation on AR/VR setups.

In future work, problems arising from the lack of suitable datasets for AR/VR systems are addressed by introducing artificial variations into an existing gaze dataset. However, since such augmentations may fail to capture the true distribution of the data, additional real data from an AR/VR setup may be necessary to learn further details about challenges of eye tracking in AR/VR environments. As mentioned in section III.A, there are currently no suitable public datasets that utilize two head-mounted eye-facing cameras of left and right eyes, and therefore a future work may involve creating such a dataset specifically built to implement deep learning models for augmented spaces. This problem does not exist however, for low quality images taken at a distance which makes training deep learning systems in their use easier.

REFERENCES

[1] D. W. Hansen and Q. Ji, "In the Eye of the Beholder: A Survey of Models for Eyes and Gaze," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 3, pp. 478-500, March 2010.

[2] Naqvi, R.A.; Arsalan, M.; Batchuluun, G.; Yoon, H.S.; Park, K.R. Deep Learning-Based Gaze Detection System for Automobile Drivers Using a NIR Camera Sensor. *Sensors* **2018**, *18*, 456.

[3] K. Krafka *et al*., "Eye Tracking for Everyone," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, 2016, pp. 2176-2184..”

[4] J. Lemley, S. Bazrafkan and P. Corcoran, "Smart Augmentation Learning an Optimal Data Augmentation Strategy," in *IEEE Access*, vol. 5, pp. 5858-5869, 2017..

[5] D. Surie, S. Partonia and H. Lindgren, "Human Sensing Using Computer Vision for Personalized Smart Spaces," *2013 IEEE 10th International Conference on Ubiquitous Intelligence and Computing and 2013 IEEE 10th International Conference on Autonomic and Trusted Computing*, Vietri sul Mere, 2013, pp. 487-494..

[6] A. Shahzada, "A Comprehensive Framework for the Development of Dynamic Smart Spaces," *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Florence, 2015, pp. 927-930.

[7] M. Raeiszadeh and H. Tahayori, "A novel method for detecting and predicting resident's behavior in smart home," *2018 6th Iranian Joint Congress on Fuzzy and Intelligent Systems (CFIS)*, Kerman, 2018, pp. 71-74.

[8] P. M. Corcoran, F. Nanu, S. Petrescu and P. Bigioi, "Real-time eye gaze tracking for gaming design and consumer electronics systems," in *IEEE Transactions on Consumer Electronics*, vol. 58, no. 2, pp. 347-355, May 2012..

[9] T. Toyama, A. Dengel, W. Suzuki, and K. Kise, “Wearable reading assist system: Augmented reality document combining document retrieval and eye tracking,” *Proc. Int. Conf. Doc. Anal. Recognition, ICDAR*, pp. 30–34, 2013.

[10] A. George and A. Routray, "Fast and accurate algorithm for eye localisation for gaze tracking in low-resolution images," in *IET Computer Vision*, vol. 10, no. 7, pp. 660-669, 10 2016.

[11] T. Pfeiffer, “Towards Gaze Interaction in Immersive Virtual Reality : Evaluation of a Monocular Eye Tracking Set-Up,” *Virtuelle und Erweiterte RealitatFunfter Work. der Gifachgr. VRAR*, pp. 81–92, 2008.

[12] J. Y. Lee, H. M. Park, S. H. Lee, T. E. Kim and J. S. Choi, "Design and Implementation of an Augmented Reality System Using Gaze Interaction," *2011 International Conference on Information Science and Applications*, Jeju Island, 2011, pp. 1-8..

[13] S. Bazrafkan, A. Kar, and C. Costache, “Eye Gaze for Consumer Electronics: Controlling and commanding intelligent systems.” *IEEE Consum. Electron. Mag.*, vol. 4, no. 4, pp. 65–71, 2015.

[14] Jason Orlosky, Takumi Toyama, Kiyoshi Kiyokawa, and Daniel Sonntag. 2015. ModulAR: Eye-Controlled Vision Augmentations for Head Mounted Displays. *IEEE Transactions on Visualization and Computer Graphics* 21, 11 (November 2015), 1259-1268.

[15] M Mihai Bâce, Teemu Leppänen, David Gil de Gomez, Argenis Ramirez Gomez:ubiGaze: ubiquitous augmented reality messaging using gaze gestures. SIGGRAPH ASIA Mobile Graphics and Interactive Applications 2016: 11:1-11:5.

[16] P. Renner and T. Pfeiffer, "Attention guiding techniques using peripheral vision and eye tracking for feedback in augmented-reality-based assistance systems," *2017 IEEE Symposium on 3D User Interfaces (3DUI)*, Los Angeles, CA, 2017, pp. 186-194.

[17] T. Piumsomboon, G. Lee, R. W. Lindeman and M. Billinghurst, "Exploring natural eye-gaze-based interaction for immersive virtual reality," *2017 IEEE Symposium on 3D User Interfaces (3DUI)*, Los Angeles, CA, 2017, pp. 36-39.

[18] N. Sidorakis, G. A. Koulieris and K. Mania, "Binocular eye-tracking for the control of a 3D immersive multimedia user interface," *2015 IEEE 1st Workshop on Everyday Virtual Reality (WEVR)*, Arles, 2015, pp. 15-18.

[19] J Justus Thies, Michael Zollhöfer, Marc Stamminger, Christian Theobalt, Matthias Nießner: FaceVR: Real-Time Facial Reenactment and Eye Gaze Control in Virtual Reality. CoRR abs/1610.03151 (2016)

[20] Michael Stengel, Steve Grogorick, Martin Eisemann, Elmar Eisemann, and Marcus A. Magnor. 2015. An Affordable Solution for Binocular Eye Tracking and Calibration in Head-mounted Displays. In *Proceedings of the 23rd ACM international conference on Multimedia* (MM '15). ACM, New York, NY, USA, 15-24.

[21] H. C. Lee, D. T. Luong, C. W. Cho, E. C. Lee, and K. R. Park, “Gaze tracking system at a distance for controlling IPTV,” *IEEE Trans. Consum. Electron.*, vol. 56, no. 4, pp. 2577–2583, 2010.

[22] S. Plesnick, D. Repice and P. Loughnane, "Eye-controlled wheelchair," *2014 IEEE Canada International Humanitarian Technology Conference - (IHTC)*, Montreal, QC, 2014, pp. 1-4..

[23] Jeffrey S. Shell, Roel Vertegaal, and Alexander W. Skaburskis. 2003. EyePliances: attention-seeking devices that respond to visual attention. In CHI '03 Extended Abstracts on Human Factors in Computing Systems (CHI EA '03). ACM, New York, NY, USA, 770-771.

[24] H. Khalife, M. A. Okdeh, A. Hage-Diab, A. Haj-Ali and B. Hussein, "Concussion detection using a commercially available eye tracker," *2017 Fourth International Conference on Advances in Biomedical Engineering (ICABME)*, Beirut, 2017, pp. 1-4.

[25] S. Hillaire, A. Lecuyer, R. Cozot and G. Casiez, "Using an Eye-Tracking System to Improve Camera Motions and Depth-of-Field Blur Effects in Virtual Environments," *2008 IEEE Virtual Reality Conference*, Reno, NE, 2008, pp. 47-50.

[26] B. B. Velichkovsky, M. a. Rumyantsev, and M. a. Morozov, “New Solution to the Midas Touch Problem - Identification of Visual Commands Via Extraction of Focal Fixations,” *Procedia Comput. Sci.*, vol. 39, pp. 75–82, 2014.

[27] H. Deng and W. Zhu, "Monocular Free-Head 3D Gaze Tracking with Deep Learning and Geometry Constraints," *2017 IEEE International Conference on Computer Vision (ICCV)*, Venice, 2017, pp. 3162-3171.

[28] Christopher D. McMurrough, Vangelis Metsis, Jonathan Rich, and Fillia Makedon. 2012. An eye tracking dataset for point of gaze detection. In *Proceedings of the Symposium on Eye Tracking Research and Applications* (ETRA '12), Stephen N. Spencer (Ed.). ACM, New York, NY, USA, 305-308.

[29] X. Zhang, Y. Sugano, M. Fritz, and A. Bulling, “Appearance-based gaze estimation in the wild,” in 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2015, pp. 4511–4520..

[30] Y. Sugano, Y. Matsushita, and Y. Sato, “Learning-by-synthesis for appearance-based 3D gaze estimation,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 1821–1828, 2014.

[31] J. Lemley, A. Kar, A. Drimbarean and P. Corcoran, “Efficient CNN Implementation for Eye-Gaze Estimation on Low-Power/Low-Quality Consumer Imaging Systems,”, 2018. Unpublished.

[32] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, 2016, pp. 770-778.

[33] Shumeet Baluja and Dean Pomerleau. 1994. Non-Intrusive Gaze Tracking Using Artificial Neural Networks. Technical Report. Carnegie Mellon Univ., Pittsburgh, PA, USA.

[34] Y. Fu, W. P. Zhu and D. Massicotte, "A gaze tracking scheme with low resolution image," *2013 IEEE 11th International New Circuits and Systems Conference (NEWCAS)*, Paris, 2013, pp. 1-4.

[35] Yu-Tzu Lin, Ruei-Yan Lin, Yu-Chih Lin, and Greg C. Lee. 2013. Real-time eye-gaze estimation using a low-resolution webcam. Multimedia Tools Appl. 65, 3 (August 2013), 543-568.

[36] M. Tonsen, J. Steil, Y. Sugano, and A. Bulling, “Invisibleeye: Mobile eye tracking using multiple low-resolution cameras and learning-based gaze estimation,” Proc. ACM Interact. Mob. Wearable Ubiquitous Technol., vol. 1, no. 3, pp. 106:1–106:21, Sep. 2017.

[37] Francois Chollet. Xception: Deep learning with depthwise separable convolutions. arXiv preprint arXiv:1610.02357, 2016.

[38] Francois Chollet ,2015 Keras https://github.com/fchollet/keras. (2015).

[39] Joseph Lemley, Bazrafkan Shabab, Peter Corcoran. “ Transfer Learning of Temporal Information for Driver Action Classification” Proceedings of the 28th Modern Artificial Intelligence and Cognitive Science Conference 2017. March 2017.

Appendix F

Semi-Parallel Deep Neural Network (SPDNN) Hybrid Architecture, First Application on Depth from Monocular Camera

Semiparallel deep neural network hybrid architecture: first application on depth from monocular camera

Shabab Bazrafkan
Hossein Javidnia
Joseph Lemley
Peter Corcoran

Shabab Bazrafkan, Hossein Javidnia, Joseph Lemley, Peter Corcoran, “Semiparallel deep neural network hybrid architecture: first application on depth from monocular camera,” *J. Electron. Imaging* **27**(4), 043041 (2018), doi: 10.1117/1.JEI.27.4.043041.

Journal of Electronic Imaging 27(4), 043041 (Jul/Aug 2018)

Semiparallel deep neural network hybrid architecture: first application on depth from monocular camera

Shabab Bazrafkan,[†] Hossein Javidnia,[*,†] Joseph Lemley, and Peter Corcoran
National University of Ireland Galway, College of Engineering, Department of Electronic Engineering, Galway, Ireland

Abstract. Deep neural networks have been applied to a wide range of problems in recent years. Convolutional neural network is applied to the problem of determining the depth from a single camera image (monocular depth). Eight different networks are designed to perform depth estimation, each of them suitable for a feature level. Networks with different pooling sizes determine different feature levels. After designing a set of networks, these models may be combined into a single network topology using graph optimization techniques. This "semiparallel deep neural network (SPDNN)" eliminates duplicated common network layers and can be further optimized by retraining to achieve an improved model compared to the individual topologies. Four SPDNN models are trained and have been evaluated at two stages on the KITTI dataset. The ground truth images in the first part of the experiment are provided by the benchmark, and for the second part, the ground truth images are the depth map results from applying a state-of-the-art stereo matching method. The results of this evaluation demonstrate that using postprocessing techniques to refine the target of the network increases the accuracy of depth estimation on individual mono images. The second evaluation shows that using segmentation data alongside the original data as the input can improve the depth estimation results to a point where performance is comparable with stereo depth estimation. The computational time is also discussed in this study. *© 2018 SPIE and IS&T* [DOI: 10.1117/1.JEI.27.4.043041]

Keywords: deep neural networks; depth estimation; monocular camera; machine learning.

Paper 180344 received Apr. 18, 2018; accepted for publication Jul. 17, 2018; published online Aug. 7, 2018.

1 Introduction

Computing pixel depth values provides a basis for understanding the three-dimensional (3-D) geometrical structure of images. Having the depth and 3-D information of a scene enables users to infer and understand its semantics and geometric structure as well as enabling many applications in computer vision such as autonomous navigation,[1] 3-D geographic information systems,[2] object detection and tracking,[3] medical imaging,[4] advanced graphical applications,[5] 3-D holography,[6] 3-D television,[7] multiview stereoscopic video compression,[8] disparity-based segmentation,[9] and human detection[10]/action recognition.[11,12]

As has been presented in the recent research,[13] using stereo images provides an accurate depth due to the advantage of having local correspondences; however, the processing time of these methods is still an open issue.

To solve this problem, it has been suggested to use single images to compute the depth values but extracting depth from monocular images requires extracting a large number of cues from the global and local information in the image. Using a single camera is more convenient in industrial applications. Stereo cameras require detailed calibration and many industrial use cases already employ single cameras, e.g., security monitoring, automotive and consumer vision systems, and camera infrastructure for traffic and pedestrian management in smart cities. These and other smart-vision applications can greatly benefit from accurate monocular depth analysis. This challenge has been studied for a decade and is still an open research problem.

Recently, the idea of using neural networks (NN) to solve this problem has attracted attention. In this paper, we tackle this problem by employing a deep neural network (DNN) equipped with semantic pixelwise segmentation utilizing our recently published disparity postprocessing method.

This paper also introduces the use of semiparallel deep neural networks (SPDNN). An SPDNN is a semiparallel network topology developed using a graph theory optimization of a set of independently optimized convolutional neural networks (CNNs), each targeted at a specific aspect of the more general classification problem. In Refs. 14 and 15, the effect of an SPDNN approach on increasing convergence and improving model generalization is discussed. For the depth from monocular vision problem a fully connected topology, optimized for fine features, is combined with a series of max-pooled topologies (2×2, 4×4, and 8×8) each optimized for coarser image features. The optimized SPDNN topology is retrained on the full training dataset and converges to an improved set of network weights.

It is worth mentioning that this network design strategy is not limited to the "depth from monocular vision" problem, and further application examples and refinements will be developed in a series of future publications, currently in press.

1.1 *Depth Map*

Deriving the 3-D structure of an object from a set of two-dimensional (2-D) points is a fundamental problem in

*Address all correspondence to: Hossein Javidnia, E-mail: h.javidnia1@nuigalway.ie

†These authors contributed equally to this work.

1017-9909/2018/$25.00 © 2018 SPIE and IS&T

computer vision. Most of these conversions from 2-D to 3-D space are based on the depth values computed for each 2-D point. In a depth map, each pixel is defined not by color, but by the distance between an object and the camera. In general, depth computation methods are divided into two categories:

1. Active methods.
2. Passive methods.

Active methods involve computing the depth in the scene by interacting with the objects and the environment. There are different types of active methods, such as light-based depth estimation, which uses the active light illumination to estimate the distance to different objects.[16] Ultrasound and time-of-flight (ToF) are other examples of active methods. These methods use the known speed of the wave to measure the time an emitted pulse takes to arrive at an image sensor.[17]

Passive methods utilize the optical features of the captured images. These methods involve extracting the depth information by computational image processing. In the category of passive methods, there are two primary approaches (a) multiview depth estimation, such as depth from stereo, and (b) monocular depth estimation.

1.2 *Stereo Vision Depth*

Stereo matching algorithms can be used to compute depth information from multiple images. By using the calibration information of the cameras, the depth images can be generated. This depth information provides useful data to identify and detect objects in the scene.[18]

In recent years, many applications, including ToF,[19,20] structured light,[21] and Kinect, were introduced to calculate depth from stereo images. Stereo vision algorithms are generally divided into two categories: local and global. Local algorithms were introduced as statistical methods that use the local information around a pixel to determine the depth value of the given pixel. These kinds of methods can be used for real-time applications if they are implemented efficiently. Global algorithms try to optimize an energy function to satisfy the depth estimation problem through various optimization techniques.[22]

In terms of computation, global methods are more complex than local methods, and they are usually impractical for real-time applications. Despite these drawbacks, they have the advantage of being more accurate than local methods. This advantage recently attracted considerable attention in the academic literature.[23,24]

For example, the global stereo model proposed in Ref. 23 works by converting the image into a set of 2-D triangles with adjacent vertices. Later, the 2-D vertices are converted to a 3-D mesh by computing the disparity values. To solve the problem of depth discontinuities, a two-layer Markov random field (MRF) is employed. The layers are fused with an energy function allowing the method to handle the depth discontinuities. The method has been evaluated on the new Middlebury 3.0 benchmark,[24] and it was ranked the most accurate at the time of the paper's publication on the average weight on the "bad 2.0" index.

Another global stereo matching algorithm, proposed in Ref. 25, makes use of the texture and edge information of the image. The problem of large disparity differences in small patches of nontextured regions is addressed by utilizing the color intensity. In addition, the main matching cost function produced by a CNN is augmented using the same color-based cost. The final results are postprocessed using a 5×5 median filter and a bilateral filter. This adaptive smoothness filtering technique is the primary reason for the algorithm's excellent performance and placement in the top of the Middlebury 3.0 benchmark.[24]

Many other methods have been proposed for stereo depth, such as PMSC,[24] GCSVR,[24] INTS,[26] MDP,[27] and ICSG,[28] which all aimed to improve the accuracy of the depth estimated from stereo vision or to introduce a new method to estimate the depth from a stereo pair. However, there is always a trade-off between accuracy and speed for stereo vision algorithms.

Table 1 shows an overview of the average normalized time by the number of pixels (s/megapixels) of the most accurate stereo matching algorithms as they are ranked by the Middlebury 3.0 benchmark, based on the "bad 2.0" metric. The ranking is on the test dense set. This comparison illustrates that obtaining an accurate depth from a stereo pair requires significant processing power. These results demonstrate that today, these methods are too resource intensive for real-time applications such as street sensing or autonomous navigation due to their demand for processing resources.

To decrease the processing power of stereo matching algorithms, researchers recently began to work on depth from monocular images. Such algorithms estimate depth from a single camera while keeping the processing power low.

1.3 *Deep Learning*

DNN are among the most recent approaches in pattern recognition science that are able to handle highly nonlinear problems in classification and regression. These models use consecutive nonlinear signal processing units in order to mix and reorient their input data to give the most representative results. The DNN structure learns from the input and then it generalizes what it learns into data samples it has never seen before.[33] The typical DNN model is composed of one or more convolutional, pooling, and fully connected layers accompanied by different regularization tasks. Each of these units is as follows:

1.3.1 *Convolutional layer*

This layer typically convolves the 3-D image I with the four-dimensional kernel W and adds a 3-D bias term b to it. The output is given as

$$P = I * W + b, \tag{1}$$

where the $*$ operator is nD convolution and P is the output of the convolution. During the training process, the kernel and bias parameters are updated in a way that optimizes the error function of the network output.

1.3.2 *Pooling layer*

The pooling layer applies a (usually) nonlinear transform (note that the average pooling is a linear transform, but the more popular max-pooling operation is nonlinear) on

Table 1 Comparison of the performance time between the most accurate stereo matching algorithms.

Algorithm	Time/MP (s)	$W \times H$ (ndisp)	Programming platform	Hardware
PMSC[24]	453	1500×1000 ($<=400$)	C++	i7-6700K, 4 GHz-GTX TITAN X
MeshStereoExt[23]	121	1500×1000 ($<=400$)	C, C++	8 Cores-NVIDIA TITAN X
APAP-Stereo[24]	97.2	1500×1000 ($<=400$)	Matlab+Mex	i7 Core 3.5 GHz, 4 Cores
NTDE[25]	114	1500×1000 ($<=400$)	n/a	i7 Core, 2.2 GHz-Geforce GTX TITAN X
MC-CNN-acrt[29]	112	1500×1000 ($<=400$)	n/a	NVIDIA GTX TITAN Black
MC-CNN+RBS[30]	140	1500×1000 ($<=400$)	C++	Intel(R) Xeon(R) CPU E5-1650 0, 3.20 GHz, 6 Cores-32 GB RAM-NVIDIA GTX TITAN X
SNP-RSM[24]	258	1500×1000 ($<=400$)	Matlab	i5, 4590 CPU, 3.3 GHz
MCCNN_Layout[24]	262	1500×1000 ($<=400$)	Matlab	i7 Core, 3.5 GHz
MC-CNN-fst[29]	1.26	1500×1000 ($<=400$)	n/a	NVIDIA GTX TITAN X
LPU[24]	3523	1500×1000 ($<=400$)	Matlab	Core i5, 4 Cores- 2xGTX 970
MDP[27]	58.5	1500×1000 ($<=400$)	n/a	4 i7 Cores, 3.4 GHz
MeshStereo[23]	54	1500×1000 ($<=400$)	C++	i7-2600, 3.40 GHz, 8 Cores
SOU4P-net[24]	678	1500×1000 ($<=400$)	n/a	i7 Core, 3.2 GHz-GTX 980
INTS[26]	127	1500×1000 ($<=400$)	C, C++	i7 Core, 3.2 GHz
GCSVR[24]	4731	1500×1000 ($<=400$)	C++	i7 Core, 2.8 GHz-Nvidia GTX 660Ti
JMR[24]	11.1	1500×1000 ($<=400$)	C++	Core i7, 3.6 GHz-GTX 980
LCU[24]	9572	750×500 ($<=200$)	Matlab, C++	1 Core Xeon CPU, E5-2690, 3.00 GHz
TMAP[31]	1796	1500×1000 ($<=400$)	Matlab	i7 Core, 2.7 GHz
SPS[24]	49.4	3000×2000 ($<=800$)	C, C++	1 i7 Core, 2.8 GHz
IDR[32]	0.36	1500×1000 ($<=400$)	CUDA C++	NVIDIA GeForce TITAN Black

the input image, which reduces the spatial size of the data representation after the operation.

It is common to put a pooling layer after each convolutional layer. Reducing the spatial size leads to less computational load and also prevents overfitting. The reduced spatial size also provides a certain amount of translation invariance.

1.3.3 *Fully connected layer*

Fully connected layers are the same as classical NN layers, where all the neurons in a layer are connected to all the neurons in their subsequent layer. The neurons give the summation of their input, multiplied by their weights, passed through their activation functions.

1.3.4 *Regularization*

Regularization is often used to prevent overfitting of an NN. One can train a more complex network (more parameters) with regularization and prevent overfitting. Different kinds of regularization methods have been proposed. The most important ones are weight regularization, drop-out,[34] and batch normalization.[35] Each regularization technique is suitable for specific applications, and no single technique works for every task.

1.4 *Monocular Vision Depth*

Depth estimation from a single image is a fundamental problem in computer vision and has potential applications in robotics, scene understanding, 3-D reconstruction, and medical imaging.[36–38] This problem remains challenging because there are no reliable cues for inferring depth from a single image. For example, temporal information and stereo correspondences are missing from such images.

As the result of the recent research, deep CNNs are setting new records for various vision applications. A deep convolutional neural field model for estimating depths from a single image has been presented in Ref. 39 by reformulating the depth estimation into a continuous conditional random

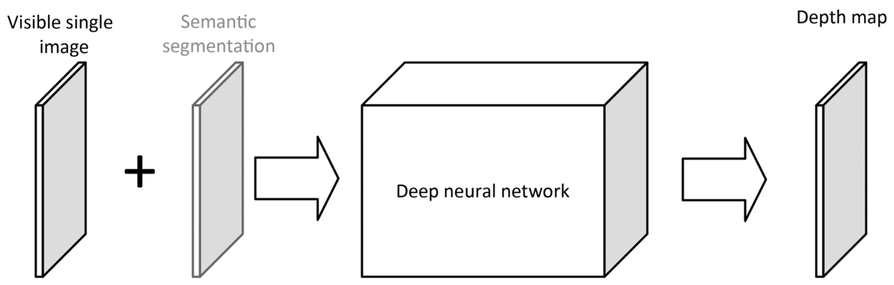


Fig. 1 The overview of the trained models in this paper. The semantic segmentation is just used in two experiments. Detailed explanation on each experiment is given in Sec. 2.3.

field (CRF) learning problem. The CNN employed in this research was composed of five convolutional and four fully connected layers. At the first stage of the algorithm, the input image was oversegmented into superpixels. The cropped image patch centered on its centroid was used as input to the CNN. For a pair of neighboring superpixels, a number of similarities were considered and were used as the input to the fully connected layer. The output of these two parts was then used as input to the CRF loss layer. As a result, the time required for estimating the depth from a single image using the trained model decreased to 1.1 s on a desktop PC equipped with NVIDIA GTX 780 GPU with 6-GB memory.

It has been found that the superpixelling technique of Ref. 39 is not a good choice to initialize the disparity estimation from mono images because of the lack of the monocular visual cues such as texture variations and gradients, defocus or color/haze in some parts of the image. To solve this issue, an MRF learning algorithm has been implemented to capture some of these monocular cues.[40] The captured cues were integrated with a stereo system to obtain better depth estimation than with the stereo system alone. This method uses a fusion of stereo + mono depth estimation.

At small distances, the algorithm relies more on stereo vision, which is more accurate than monocular vision. However, at further distances, the performance of stereo degrades; and the algorithm relies more on monocular vision.

The problem of depth estimation from monocular images has been also studied in Ref. 41, where a network is designed with two components. First, the global structure of the scene is estimated and later refined using local information. Although this approach enables the early idea of estimating monocular depth using CNNs, the output depth maps do not clearly represent the geometrical structure of the scene.

In another approach,[42] an unsupervised convolutional encoder is trained to estimate the depth from monocular images. The depth is estimated considering the small motion between two images (stereo set as input and target). Later, the inverse warp of the target image is generated using the predicted depth and the known displacement between cameras, which results in reconstructing the source image. In a similar research,[43] an unsupervised CNN is trained by exploiting epipolar geometry constraints to estimate disparity from single images. The idea is to learn a function that is able to reconstruct one image from the other by utilizing a calibrated pair of binocular cameras. A left-right disparity consistency loss is also introduced, which combines smoothness, reconstruction, and left-right disparity consistency terms and keeps the consistency between the disparities produced relative to both the left and right images.

In Refs. 44 and 45, authors presented a method to mix the output information of multiple CNNs using CRF where two different models are proposed, one with a cascade of CRFs and the other with a unified graph. They trained and tested the networks on NYU Depth V2,[46] KITTI,[47] and Make 3-D datasets.[48,49] The best results were drawn from ResNet50.

1.5 *Paper Overview*

In this paper, a DNN is presented to estimate depth from monocular cameras. The depth map from the stereo sets is estimated using the same approach as Ref. 50 and they are used as the target to train the network while using information from a single image (the left image in the stereo set) as input. Four models are trained and evaluated to estimate the depth from single camera images. The network structure for all the models is the same. In the first case, the input is simply the original image. In the second case, the first channel is the original image and the second channel is its segmentation map. For each of these two cases, one of two different targets is used; specifically, these targets were the stereo depth maps with or without postprocessing explained in Ref. 50. Figure 1 shows the overview of the general approach used in this paper. In this figure, the DNN is shown as a black box. The semantic segmentation has been used in two experiments out of four. A detailed explanation of each experiment is given in Sec. 2.3.

1.6 *Contributions*

In this paper, two major contributions are presented:

1. SPDNN[15] is a method to mix and merge several DNNs. This method is versatile enough to be applied to any DNN design. In this work, this method is utilized to design a network to approximate the depth from the monocular images. Network design procedure is described in detail in Appendix A.
2. The application of DNNs and SPDNN on estimating depth from a monocular camera.
3. The effect of using segmentation information in approximating depth is investigated.
4. Two different ground truth sets have been used to train the network and comparisons of the network performances for each ground truth have been investigated.

The rest of the paper is organized as follows: in the next section, the network structure, database preparation, and the training process are presented. Section 3 discusses the results and evaluation of the proposed method. The conclusion and discussions are presented in the last section.

2 Methodology

2.1 *Network Structure*

2.1.1 *Semiparallel deep neural network*

This paper introduces the SPDNN concept, inspired by graph optimization techniques. In this method, several DNNs are parallelized and merged in a way that facilitates the advantages of each. The final model is trained for the problem. References 14 and 15 show that using this method increases the convergence and generalization of the model compared to alternatives.

The merging of multiple networks using SPDNN is described in the context of the current depth mapping problem. In this particular problem, eight different networks were designed for the depth estimation task. These are described in detail in Appendix A. None of these networks on their own gave useful results on the depth analysis problem. However, it was noticed that each network tended to perform well on certain aspects of this task while failing at others. This led to the idea that it would be advantageous to combine multiple individual networks and train them in a parallelized architecture. Our experiments showed that better output could be achieved by merging the networks and then training them concurrently.

Combined model/architecture. The process of the network design is discussed in detail in Appendix A. In the final model presented in Fig. 2, the input image is first processed in four parallel fully convolutional subnetworks with different pooling sizes. This provides the advantages of different networks with different pooling sizes at the same time. The outputs of these four subnetworks are concatenated in two different forms; one is to pool the larger images to be the same size as the smallest image in the previous part, and the other one is to unpool the smaller images of the previous part to be the same size as the largest image.

After merging these outputs, the data are led to two different networks. One is the fully convolutional network (FCN) to deepen the learning and release more abstract features of the input, and the other network is an autoencoder network with different architectures for encoder and decoder.

It is mentioned in the network design section in Appendix A that having a fully connected layer in the network is crucial for correct estimation of the image's depth, which is provided in the bottleneck of the autoencoder. The results from the autoencoder and the fully convolutional subnetwork are again merged in order to give a single output after applying a one channel convolutional layer.

In order to regularize the network, prevent overfitting, and increase the convergence, batch normalization[35] is applied after every convolutional layer, and the drop-out technique[34] is used in fully connected layers. The experiments in this paper show that using weight regularization in the fully connected layers gives slower convergence; therefore, this regularization was eliminated from the final design. All the nonlinearities in the network are the ReLU nonlinearity, which is widely used in DNNs, except for the output layer, which took advantage of the sigmoid nonlinearity. The value repeating technique was used in the unpooling layer due to nonspecificity of the corresponding pooled layer in the decoder part of the autoencoder subnetwork.

The value repeating technique, shown in Fig. 3, involves repeating the value from the previous layer in order to obtain the unpooled image. The figure shows the 2×2 unpooling, and the process is the same for other unpooling sizes.

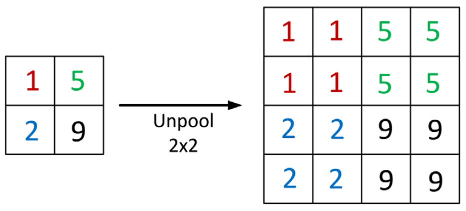


Fig. 3 The repeating technique used in unpooling layers.

2.2 *Database*

In this paper, the KITTI Stereo 2012, 2015 datasets[47] are used for training and evaluation of the network. The database is augmented by vertical and horizontal flipping to expand

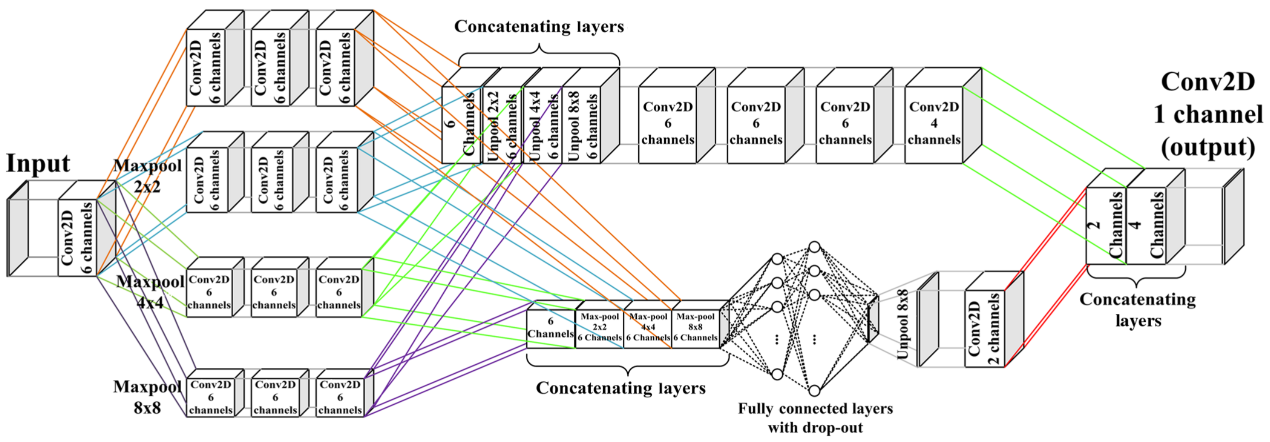


Fig. 2 The model designed for the depth estimation from monocular images. The network design is explained in Appendix A.2.

the total size to 33,096 images. 70% of this dataset is used for training, 20% for validation, and 10% for testing. Dividing the database to train-validation-test subsets is performed before scrambling the indices so there is minimum correspondence between the samples in each subset. The reason for this is because the database is drawn from sequences of images wherein each two consecutive samples look very similar. Each model is trained for two sets of input samples and two sets of output targets. The input and target preparation are explained in the following sections.

2.2.1 *Data preparation*

Input preparation. Two different sets have been used as the input of the network. The first set includes the visible images given by the left camera. The second set is the visible image + the semantic segmentation of the corresponding input. This gives the opportunity of investigating the segmentation influence on the depth estimation problem. The segmentation map for each image is calculated by employing the well-known model "SegNet."[51,52] This model is one of the most successful recent implementations of DNN for semantic pixelwise image segmentation and has surpassed other configurations of FCNs both in accuracy and simplicity of implementation. A short description of SegNet is given in Appendix B.

In our experiments, SegNet was trained using stochastic gradient descent with learning rate 0.1 and momentum 0.9. In this paper, the Caffe implementation of SegNet has been employed for training purposes.[53] The gray-scale CamVid road scene database (360×480)[54] has been used in the training step.

Target preparation. The targets for training the network are generated from the stereo information using the adaptive random walk with restart algorithm.[55] The output of the stereo matching algorithm suffers from several artifacts, which are addressed and solved by a postprocessing method in Ref. 50. In the present experiments, both depth maps (before postprocessing and after postprocessing) are used independently as targets. The postprocessing procedure is based on the mutual information of the RGB image (used as a reference image) and the initial estimated depth image. This approach has been used to increase the accuracy of the depth estimation in stereo vision by preserving the edges and corners in the depth map and filling in the missing parts. The method was compared with the top eight depth estimation methods in the Middlebury benchmark[24] at the time the paper was authored. Seven metrics, including mean square error (MSE), root mean square error (RMSE), peak signal-to-noise ratio (PSNR), signal-to-noise ratio (SNR), mean absolute error (MAE), structural similarity index (SSIM), and structural dissimilarity index were used to evaluate the performance of each method. The evaluation ranked the method as first in five metrics and second and third in other metrics

2.3 *Training*

As described in Sec. 2.2.1, there are two separate sets as inputs and two separate sets as targets for the training process. This will give four experiments in total as follows:

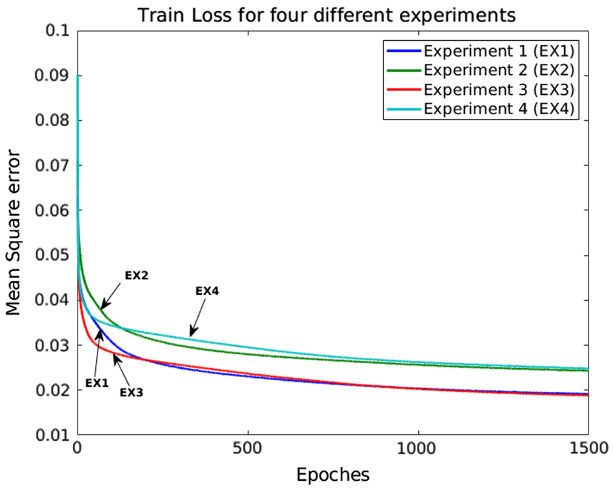


Fig. 4 Train loss for each experiment.

1. Experiment 1: Input: left visible image + pixelwise segmented image. Target: postprocessed depth map
2. Experiment 2: Input: left visible image. Target: postprocessed depth map.
3. Experiment 3: Input: left visible image + pixelwise segmented image. Target: depth map.
4. Experiment 4: Input: left visible image. Target: depth map.

The images are resized to 80×264 pixels during the whole process. Training is done on a standard desktop with an NVIDIA GTX 1080 GPU with 8 GB memory.

In the presented experiments, the MSE value between the output of the network and the target values have been used as the loss function, and the Nestrov momentum technique[56] with learning rate 0.01 and momentum 0.9 has been used to train the network. The training and validation loss for each of these experiments are shown in Figs. 4 and 5, respectively.

These figures show that using the postprocessed depth map as the target results in lower loss values, which means that the network was able to learn better features in those

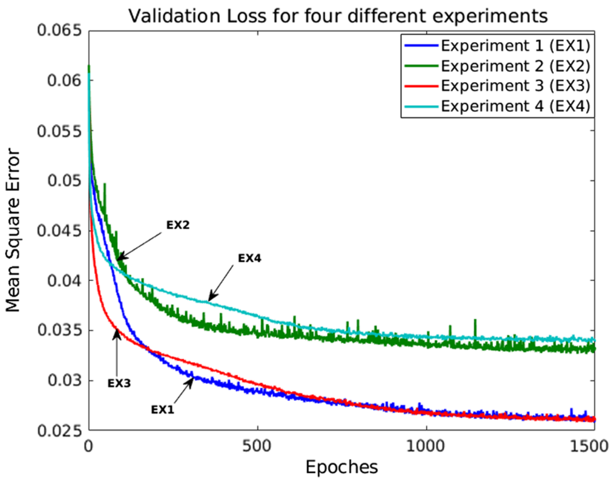


Fig. 5 Validation loss for each experiment.

experiments, while semantic segmentation decreases the error only marginally.

The reason that the postprocessed depth maps are considered as the target in two experiments is twofold: first, the postprocessing pipeline is proven to be effective in increasing the performance of the depth estimation methods by considering the geometrical structure of the scene. Second, it helps the network to avoid the densification process of the sparse depth maps, which are captured using LIDAR scanners.

3 Results and Evaluations

The evaluation in this paper has been done in four parts. In the first two parts, the four experiments given in Sec. 2.3 are compared to each other, given different ground truths. The third part compares the proposed method to a stereo matching method and the last part shows the comparison against the state-of-the-art monocular depth estimation method. For evaluation purposes, eight metrics including PSNR, MSE (between 0 and 1), RMSE (between 0 and 1), SNR, MAE (between 0 and 1), SSIM (between 0 and 1),[57] universal quality index (UQI) (between 0 and 1),[58] and Pearson correlation coefficient (PCC) (between -1 and 1)[59] are used. For the metrics PSNR, SNR, SSIM, UQI, and PCC, the larger value indicates better performance, and for MSE, RMSE, and MAE, the lower value indicates better performance. PSNR, MSE, RMSE, MAE, and SNR represent the general similarities between two objects. UQI and SSIM are structural similarity indicators and PCC represents the correlation between two samples. To the best of our knowledge, there have been no other attempts at estimating depth from a mono camera on the KITTI benchmark.

3.1 *Comparing Experiments Given Benchmark Ground Truth*

The KITTI database came with a depth map ground truth generated by a LIDAR scanner.

The test set has been forward propagated through the four different models trained in the four experiments, and the output of the networks has been compared to the benchmark ground truth. The results are shown in Table 2. The best value for each metric is presented in bold.

Table 2 Numerical comparison of the models given the benchmark's ground truth.

	Exp. 1	Exp. 2	Exp. 3	Exp. 4
PSNR	**14.3424**	13.7677	13.8333	13.8179
MSE	**0.0382**	0.0436	0.0435	0.0439
RMSE	**0.1937**	0.2069	0.206	0.2066
SNR	4.4026	3.8279	**6.1952**	6.1798
MAE	**0.1107**	0.1212	0.1236	0.1234
SSIM	**0.9959**	0.9955	0.9955	0.9955
UQI	0.9234	**0.9252**	0.9053	0.9064
PCC	0.7687	**0.8485**	0.7702	0.7729

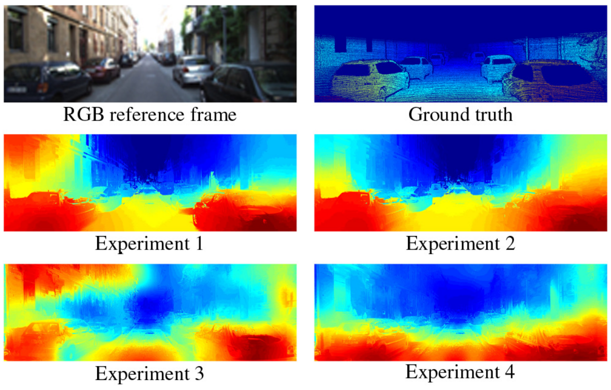


Fig. 6 Estimated depth maps from the trained models, example 1, for experiments 1 to 4 explained in Sec. 2.3.

Figures 6–8 represent the color-coded depth maps computed by the trained models using the proposed DNN, where the dark red and dark blue parts represent closest and furthest points to the camera, respectively. On the top right of each figure, the ground truth given by the benchmark is illustrated. For visualization purposes, all of the images presented in this section are upsampled using joint bilateral upsampling.[60] The results show that using semantic segmentation along with the visible image as input will improve the model

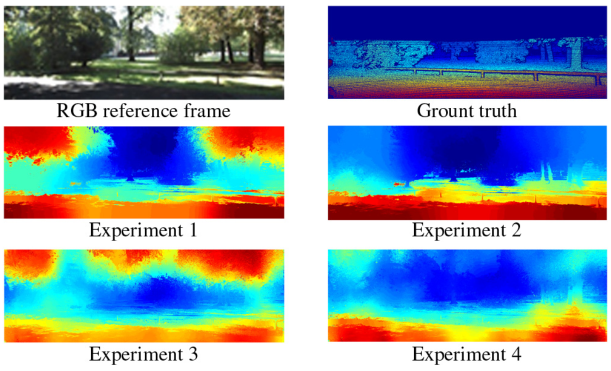


Fig. 7 Estimated depth maps from the trained models, example 2, for experiments 1 to 4 explained in Sec. 2.3.

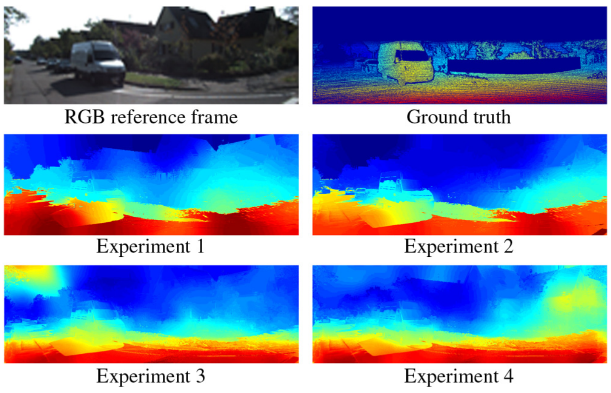


Fig. 8 Estimated depth maps from the trained models, example 3, for experiments 1 to 4 explained in Sec. 2.3.

marginally. Using the postprocessed target in the training stage helps the model to converge to more realistic results.

As it is shown in Figs. 6–8, the depth map generated in experiment 1 contains more structural details, and more precise, less faulty depth levels compared with the other experiments. In general, the presented models in this paper are able to handle occlusions and discontinuities at different depth levels.

3.2 *Comparing Experiments Given the Ground Truth from Stereo Matching*

In this section, proposed models are compared to see which one produces closer results to the target value. This gives an idea whether using deep learning techniques on the mono camera can produce reasonable results or not.

Images in the test set have been forward propagated through the models trained in Sec. 2.3, and the outputs are compared with the depth map generated in Ref. 50. The numerical results are shown in Table 3.

The best value for each metric is presented in bold. Figures 9–11 represent the color-coded depth maps computed by the trained models using the proposed DNN, where the dark red and dark blue parts represent closest and furthest points to the camera, respectively. On the top right of each figure, the ground truth calculated in Ref. 50 is illustrated. For visualization purposes, all of the images presented in this section are upsampled using joint bilateral upsampling.[60]

The results show that using semantic segmentation along with the visible image as input will improve the model marginally. Using the postprocessed target in the training stage helps the model to converge to more realistic results.

Figures 9–11 show that the trained models in this paper are able to estimate depth maps comparable to state-of-the-art stereo matching with structural accuracy and precise depth levels. This is also a result of using the semantic segmentation data and injecting the structural information into the network.

3.3 *Comparing Mono Camera Results with Stereo Matching*

In this section, the results from the mono camera depth estimation given by the proposed method are compared with one of the top-ranked stereo matching methods given in Ref. 50. The ground truth for this comparison is the set of depth maps provided by the KITTI benchmark.

Table 3 Numerical comparison of the models given the ground truth from stereo matching.

	Exp. 1	Exp. 2	Exp. 3	Exp. 4
PSNR	**15.0418**	14.1895	13.3819	14.0491
MSE	**0.0378**	0.0447	0.0535	0.0441
RMSE	**0.1854**	0.203	0.2223	0.2039
SNR	**8.822**	7.9696	5.4271	6.0943
MAE	**0.1442**	0.1581	0.1673	0.153
SSIM	**0.9952**	0.9943	0.994	0.9951
UQI	**0.8401**	0.8369	0.7951	0.8178
PCC	**0.8082**	0.795	0.704	0.6919

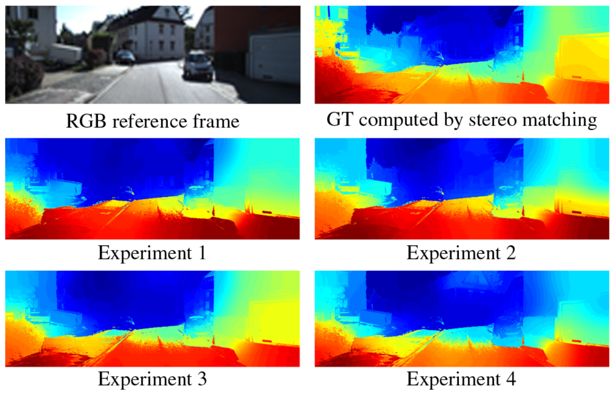


Fig. 10 Estimated depth maps from the trained models, example 2, for experiments 1 to 4 explained in Sec. 2.3.

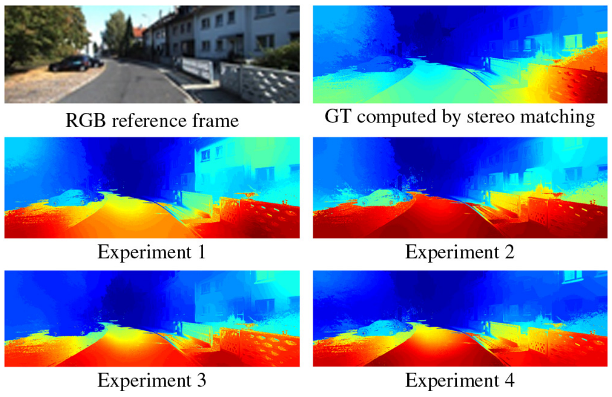


Fig. 9 Estimated depth maps from the trained models, example 1, for experiments 1 to 4 explained in Sec. 2.3.

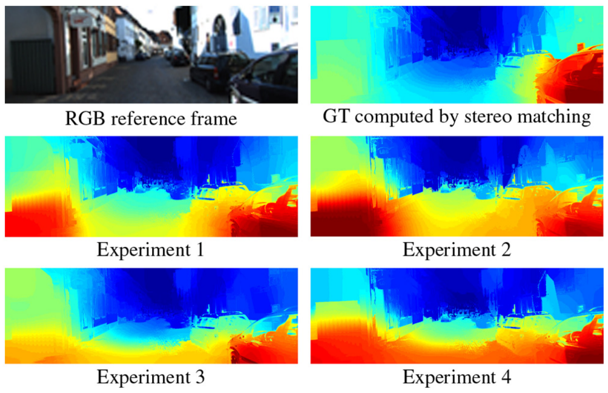


Fig. 11 Estimated depth maps from the trained models, example 3, for experiments 1 to 4 explained in Sec. 2.3.

Table 4 Numerical comparison between stereo matching and the proposed mono camera model.

	Stereo matching[50]	Mono camera DNN
PSNR	14.8234	14.3424
MSE	0.0351	0.0382
RMSE	0.1845	0.1937
SNR	4.8836	4.4026
MAE	0.1017	0.1107
SSIM	0.9966	0.9959
UQI	0.9353	0.9234
PCC	0.823	0.7687

The test images have been forward propagated through the models trained in Sec. 2.3 and the best results are compared with the stereo matching technique. The results are shown in Table 4.

The results indicate that using mono camera images and deep learning techniques can provide results that are comparable to stereo matching techniques. As shown in Table 4, the mono camera DNN method was able to provide depth maps similar to the stereo matching methods, represented by PSNR, MSE, MAE, RMSE, and SNR.

Having close values for SSIM (0.9966 and 0.9959 in the range [0,1]) and UQI (0.9353 and 0.9234 in the range [0,1]) shows how the mono camera DNN method is able to preserve the structural information, as compared to the stereo matching method.

3.4 *Comparison against Other Monocular Depth Estimation Methods*

In this section, the proposed network is compared against the method presented in Refs. 39 and 41–43. Table 5 represents the performance of the proposed network compared to the state-of-the-art methods based on seven metrics including absolute relative difference, squared relative difference, and RMSE/RMSE log. The metrics are defined as follows:

- Mean relative error (Rel): $\frac{1}{P}\sum_{i=1}^{P}\frac{|\tilde{d}_i-d_i^*|}{d_i^*}$;
- Root mean squared error (RMSE): $\sqrt{\frac{1}{P}\sum_{i=1}^{P}(\tilde{d}_i-d_i^*)^2}$;
- Mean log 10 error: $\frac{1}{P}\sum_{i=1}^{P}\|\log_{10}(\tilde{d}_i)-\log_{10}(d_i^*)\|$;
- Accuracy with threshold t: Percentage (%) of d_i^*, subject to $\max\left(\frac{d_i^*}{\tilde{d}_i},\frac{\tilde{d}_i}{d_i^*}\right)=\delta<t\ (t\in[1.25,1.25^2,1.25^3])$.

These numbers indicate that the unsupervised CNN proposed by Godard et al.[43] outperforms the others because of the left-right disparity consistency term, which allows the network to optimize the disparity values based on both left and right images. However, we believe that the proposed network has a competitive performance compared to the studied methods considering the fact that our models are trained using only the left image without taking into account the influence of the right disparity values.

3.5 *Comparing Running Times*

In this section, the computational time of the proposed method is compared against the stereo matching methods provided in Table 1. The evaluations indicate that the proposed method is able to perform at a rate of ~1.23 s/MP on a desktop computer equipped with i7 2600 CPU @ 3.4 GHz and 16 GB of RAM.

Table 5 Results on the KITTI 2015 stereo 200 training set disparity images.

Method	Stereo	Dataset	Abs Rel	Sq Rel	RMSE	RMSE log	$\delta<1.25$	$\delta<1.25^2$	$\delta<1.25^3$
Eigen et al.[41] coarse	No	KITTI	0.361	4.826	8.102	0.377	0.638	0.804	0.894
Eigen et al.[41] fine	No	KITTI	0.203	1.548	6.307	0.282	0.702	0.890	0.958
Liu et al.[39] DCNF-FCSP FT	No	KITTI	0.201	1.584	6.471	0.273	0.68	0.898	0.967
Garg et al.[42] L12 Aug 8× cap 50 m	Yes	KITTI	0.169	1.080	5.104	0.273	0.740	0.904	0.962
Godard et al.[43]	Yes	KITTI	0.148	1.344	5.927	0.247	0.803	0.922	0.964
Saxena et al.[61]	No	KITTI	0.280	—	8.734	0.327	0.601	0.820	0.926
Zhou et al.[62]	No	KITTI	0.208	1.768	6.858	—	0.678	0.885	0.957
Kuznietsov et al.[63] (only supervised)	No	KITTI	—	—	4.815	—	0.845	0.957	**0.987**
Kuznietsov et al.[63]	Yes	KITTI	—	—	4.621	—	**0.852**	**0.960**	0.986
Xu et al.[44]	No	KITTI	**0.125**	**0.899**	4.685	**0.154**	0.816	0.951	0.983
Ours	No	KITTI	0.288	1.065	**4.071**	0.401	0.51	0.77	0.893

In columns 4–7 lower is better, in columns 8–10 higher is better.

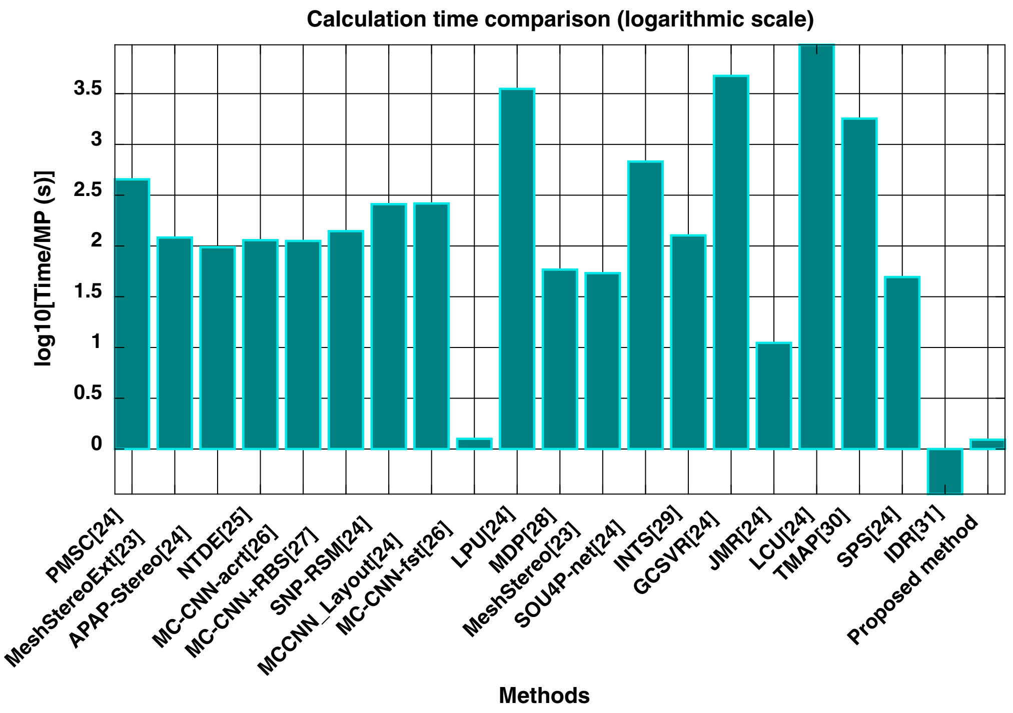


Fig. 12 Comparison of computational time in logarithmic scale.

Figure 12 shows the comparison of the computational times. The comparison is done in a logarithmic scale due to the large range of computational times between different methods.

3.6 *Effects of Scaling, Rotation, and Translation*

In this section, the effect of the scaling, rotation, and translation is explained for the proposed method. The test data have been manipulated in three ways:

1. Scaling: Images have been cropped with random values for height in $U[37,70]$ and width in $U[128,240]$ where $U[a,b]$ is the uniform distribution between a and b. The cropped images are resized to [80, 264] using bilinear interpolation.
2. Rotating: Each sample in the test set has been rotated randomly between 3 deg and 30 deg and also −3 deg and −30 deg.
3. Translating: Each sample has been translated with random values between 5 and 30 pixels in height and 50 and 100 pixels in width.

Each set of these samples has been tested on the proposed networks for each experiment and the results are given in Tables 6–9.

This experiment shows that the method is relatively robust to scaling in comparison to rotation and translation. Translation introduces more error than rotation. The main reason is the change in the sky position in the translated images. Since the network is trained on samples where the

Table 6 Network trained in experiment 1, tested on scaled, rotated, and translated samples.

	Exp. 1, original	Exp. 1, scaled	Exp. 1, rotated	Exp. 1, translated
PSNR	14.3424	13.4155	7.3789	6.9341
MSE	0.0382	0.0561	0.1890	0.2083
RMSE	0.1937	0.2253	0.4313	0.4533
SNR	4.4026	1.4561	3.0047	2.5507
MAE	0.1107	0.1811	0.3480	0.3609
SSIM	0.9959	0.9924	0.9791	0.9754
UQI	0.9234	0.2234	0.343	0.530

Table 7 Network trained in experiment 2, tested on scaled, rotated, and translated samples.

	Exp. 2, original	Exp. 2, scaled	Exp. 2, rotated	Exp. 2, translated
PSNR	13.7677	13.1384	7.1016	6.7348
MSE	0.0436	0.0600	0.2017	0.2218
RMSE	0.2069	0.2328	0.4454	0.4660
SNR	3.8279	1.8092	3.7111	2.8531
MAE	0.1212	0.1882	0.3501	0.3665
SSIM	0.9955	0.9919	0.9776	0.9738
UQI	0.9252	0.2244	0.708	0.764

Table 8 Network trained in experiment 3, tested on scaled, rotated, and translated samples.

	Exp. 3, original	Exp. 3, scaled	Exp. 3, rotated	Exp. 3, translated
PSNR	13.8333	11.5803	7.2379	6.2700
MSE	0.0435	0.0808	0.1962	0.2420
RMSE	0.206	0.2740	0.4388	0.4890
SNR	6.1952	4.1375	4.6070	5.3933
MAE	0.1236	0.2187	0.3580	0.3787
SSIM	0.9955	0.9897	0.9780	0.9715
UQI	0.9053	0.826	0.293	0.489

Table 9 Network trained in experiment 4, tested on scaled, rotated, and translated samples.

	Exp. 4, original	Exp. 4, scaled	Exp. 4, rotated	Exp. 4, translated
PSNR	13.8179	12.0590	8.0581	7.5241
MSE	0.0439	0.0725	0.1649	0.1847
RMSE	0.2066	0.2595	0.4009	0.4253
SNR	6.1798	3.6592	3.5843	4.3851
MAE	0.1234	0.2023	0.3191	0.3294
SSIM	0.9955	0.9908	0.9817	0.978
UQI	0.9064	0.878	0.250	0.468

sky is at the top of the image, the translating the sky position induces a large amount of uncertainty on the output values.

The other observation is for using segmentation as auxiliary information for depth estimation. The observations show that the segmentation is not introducing any helpful information while dealing with scaling, rotation, and translation.

4 Conclusion and Discussion

In this paper, we have introduced the use of the SPDNN method. An SPDNN is a network topology developed using a graph theory optimization of a set of independently optimized CNNs, each targeted at a specific aspect of the more general classification problem. For depth estimation from a monocular setup, a model including fully connected topology optimized for fine features is combined with a series of max-pooled topologies. The optimized SPDNN topology is retrained on the full training dataset and converges to an improved set of network weights. Here, we used this design strategy to train an accurate model for estimating depth from monocular images.

In this work, eight different DNNs have been mixed and merged using the SPDNN method in order to take advantage of each network's qualities. The mixed network architecture was then trained in four separate scenarios where each scenario used a different set of inputs and targets during training. Four distinct models have been trained. The pixelwise segmentation and depth estimations given in Ref. 50 were used to provide samples for use in the training stage. The KITTI benchmark was used for training and experimental purposes.

Each model was evaluated in two sections, first against the ground truth provided by the benchmark, and second against the disparity maps computed by the stereo matching method (Secs. 3.1 and 3.2). The results show that using the postprocessed depth map presented in Ref. 50 for training the network results in more precise models and adding the semantic segmentation of the input frame to the input helps the network preserve the structural information in the output depth map. The results in Sec. 3.2 show how close the proposed depth estimation using mono camera can be to the stereo matching method. The semantic segmentation information helps the network converge to the stereo matching results, although the improvement is marginal in this case. The results of the third comparisons in Sec. 3.3 show a slightly higher accuracy obtained by employing the stereo matching technique, but our results demonstrate that there is not a big difference between the depths from the models trained by the proposed DNN and the values computed by stereo matching. The numerical results of this evaluation show the similarity between the mono camera using the DNN method and the stereo matching method, and also the power of the presented method in preserving the structural information in the output depth map.

An important advantage of these models is the processing time of ∼1.23 s/MP. This is equal to 38 fps for an input image of size (80×264) on an i7 2600 CPU @ 3.4 GHz and 16 GB of RAM. This makes the model suitable for providing depth estimation in real time. This performance is comparable to the stereo methods MC-CNN-fst[29] and JMR,[24] whose times are 37 and 4 fps, respectively, for the same size of the image, taking advantage of GPU computation power (NVIDIA GTX TITAN X and GTX 980, respectively). The IDR method[32] can give up to 131 fps for the same image size by using an NVIDIA GeForce TITAN Black GPU and CUDA C++ implementation, but the performance on a CPU is not given by the authors, so any comparisons with this method would be unfair.

Using pixelwise segmentation as one of the inputs of the network slightly increased the accuracy of the models, and also helped the model preserve the structural details of the input image. However, it also brought some artifacts, such as wrong depth patches on the surfaces. The evaluation results also illustrate the higher accuracy of the models where a postprocessed depth map was used as the target in the training procedure.

4.1 *Future Works and Improvements*

The model presented in this work is still a big model to implement in low power consumer electronic devices (e.g., handheld devices). Future work will include a smaller design that is able to perform as well as the presented model. The other consideration for the current method is the training data size (which is always the biggest consideration with deep learning approaches). The amount of stereo data available in the databases is usually not big enough to train a DNN. The augmentation techniques can help to expand databases, but the amount of extra information they provide is limited. Providing a larger set with accurate depth maps will improve the results significantly.

The future works also involve designing and training networks on other databases, such as NYU Depth V2[46] and Make3D,[48,49] and performing interdatabase evaluation wherein the network is trained on one database and tested on another one. We will also utilize the SPDNN method to design a network for a mixed database to get more generalization power.

The SPDNN approach is currently being applied to other problems and is giving promising results on both classification and regression problems. Those results will be presented in future publications.

Appendix A: Network Design

A1 Individual Networks for Depth Analysis

The network shown in Fig. 13 is a deep fully CNN (a fully CNN is a network wherein all the layers are convolutional layers) with no pooling and no padding. Therefore, no information loss occurs inside the network, as there is no bottleneck or data compression; this network is able to preserve the details of the input samples. But the main problem is that this model is unable to find big objects and coarse features in the image. In order to solve this problem, three other networks have been designed as shown in Figs. 14–16. These three networks take advantage of the max-pooling layers to gain transition invariance and also to recognize larger objects and coarser features inside the image. These networks use 2×2, 4×4, and 8×8 max-pooling operators,

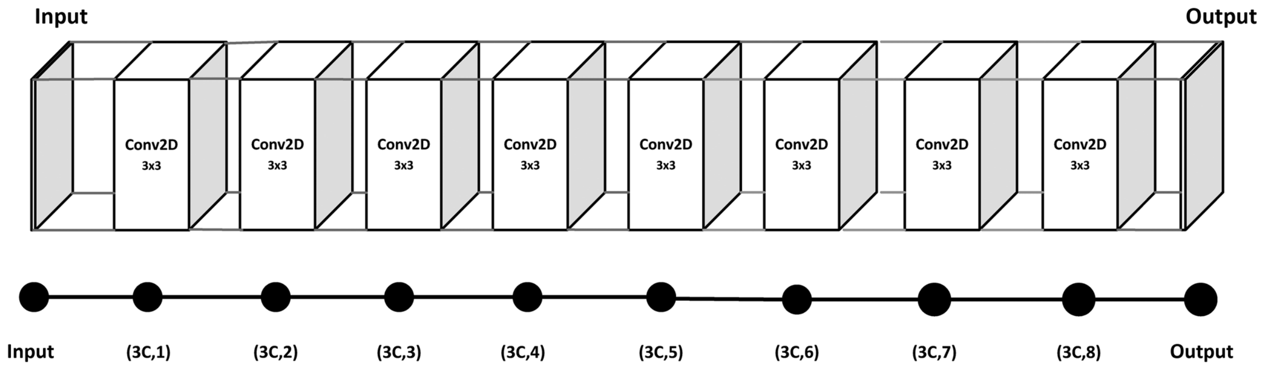


Fig. 13 Top row: network 1, Bottom row: graph corresponds to network 1.

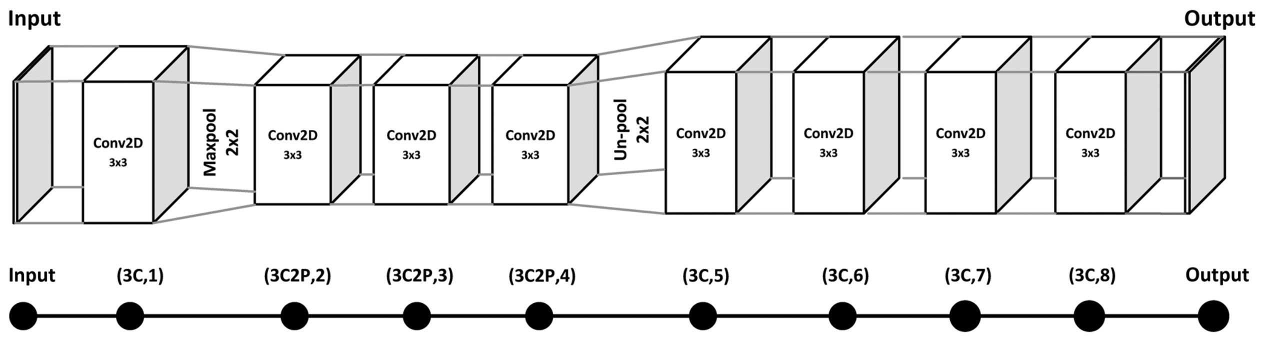


Fig. 14 Top row: network 2, Bottom row: graph corresponds to network 2.

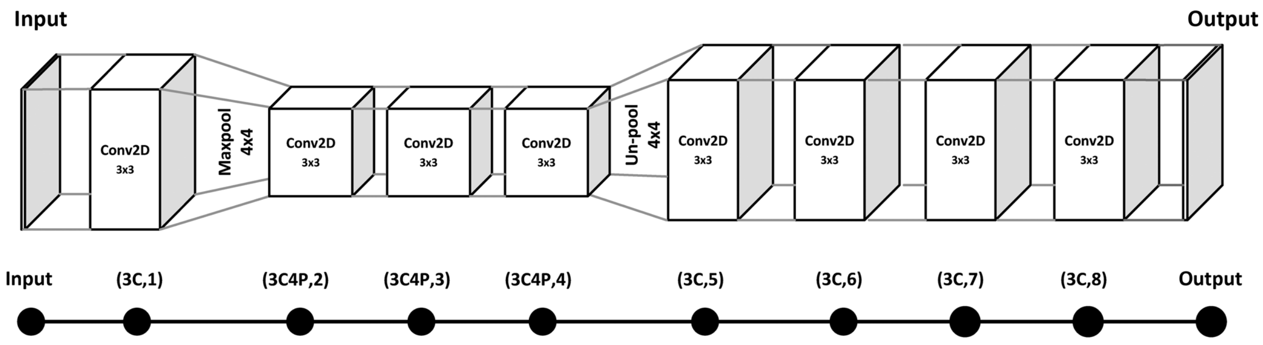


Fig. 15 Top row: network 3, Bottom row: graph corresponds to network 3.

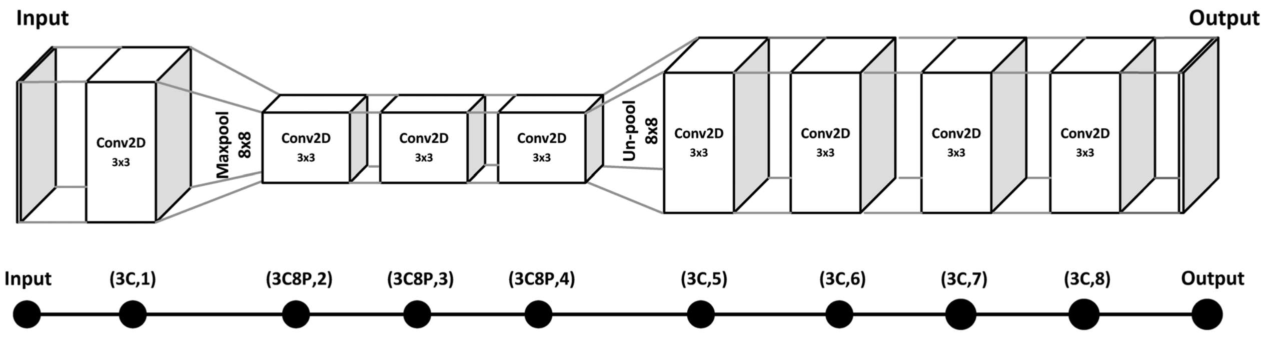


Fig. 16 Top row: network 4, Bottom row: graph corresponds to network 4.

respectively. Larger pooling kernels allow coarser features to be detected by the network. The main problem with these networks is that the spatial details vanished as a result of data compression in pooling layers.

After several attempts of designing different networks, the observations showed that in order to estimate the depth from an image, the network needed to see the whole image as one object. To do that requires the kernel to be the same size as the image in at least one layer that is equivalent to a fully connected layer inside the network.

In fully connected layers, each neuron is connected to all neurons in the previous/next layer. Due to the computationally prohibitive nature of training fully connected layers and their tendency to cause overfitting, it is desirable to reduce the number of these connections. Adding fully connected layers results in a very tight bottleneck, which seems to be crucial for the depth estimation task, but also causes the majority of the details in the image to be lost. In Figs. 17–20, the networks with fully connected layers are shown. These networks correspond to networks in Figs. 13–16 but with convolutional layers replaced with fully connected layers on the righthand side of the network. Using different pooling sizes before the fully connected layer will cause the network to extract different levels of features, but all these configurations introduce loss of detail.

Each of these eight configurations has its own advantages and shortcomings, from missing the coarse features to

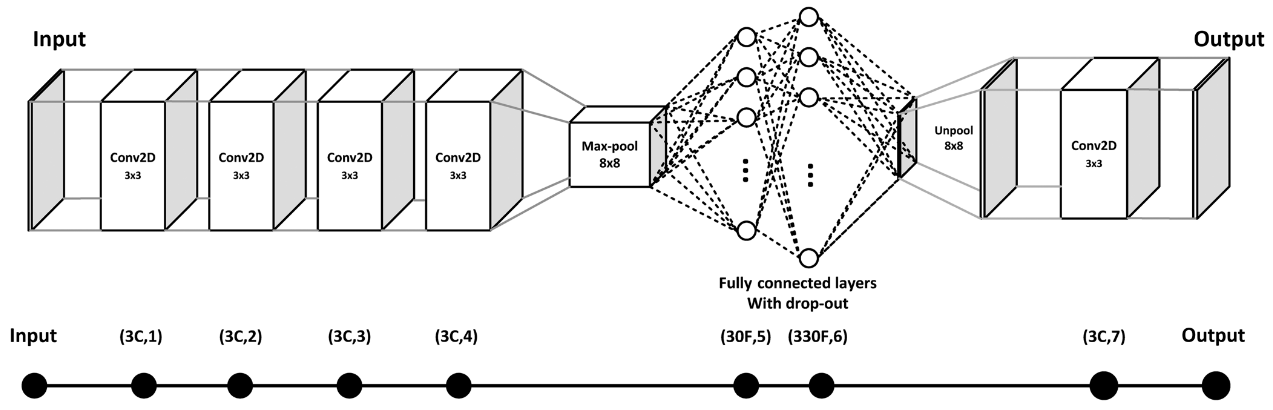


Fig. 17 Top row: network 5, Bottom row: graph corresponds to network 5.

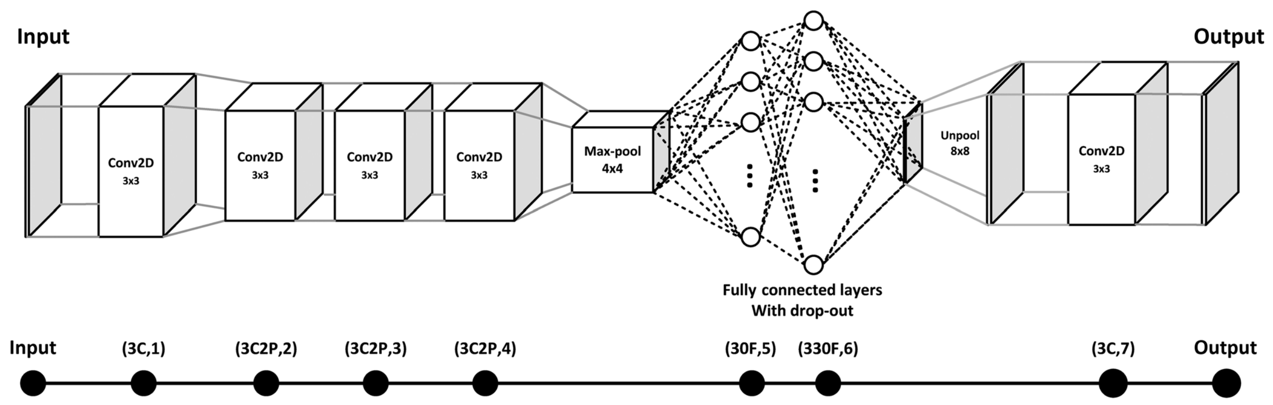


Fig. 18 Top row: network 5, Bottom row: graph corresponds to network 6.

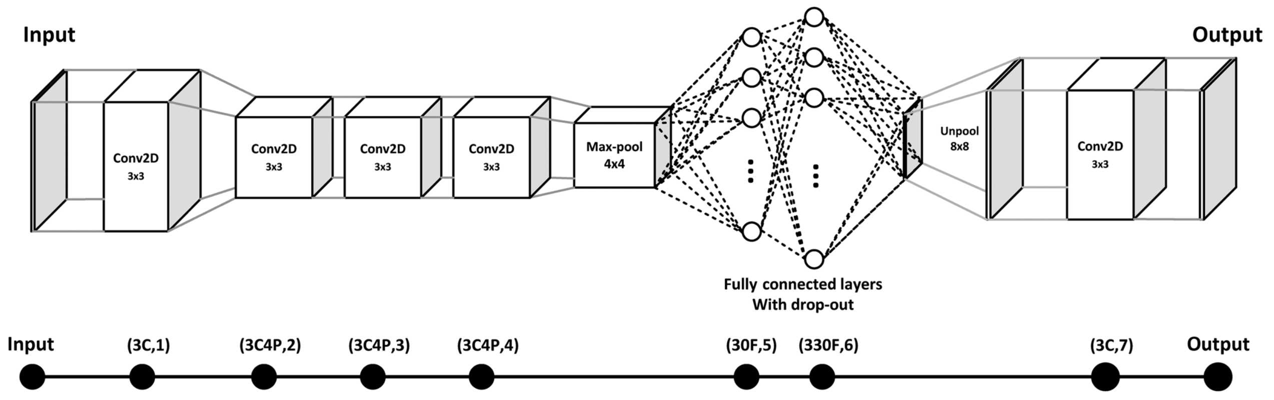


Fig. 19 Top row: network 7, Bottom row: graph corresponds to network 7.

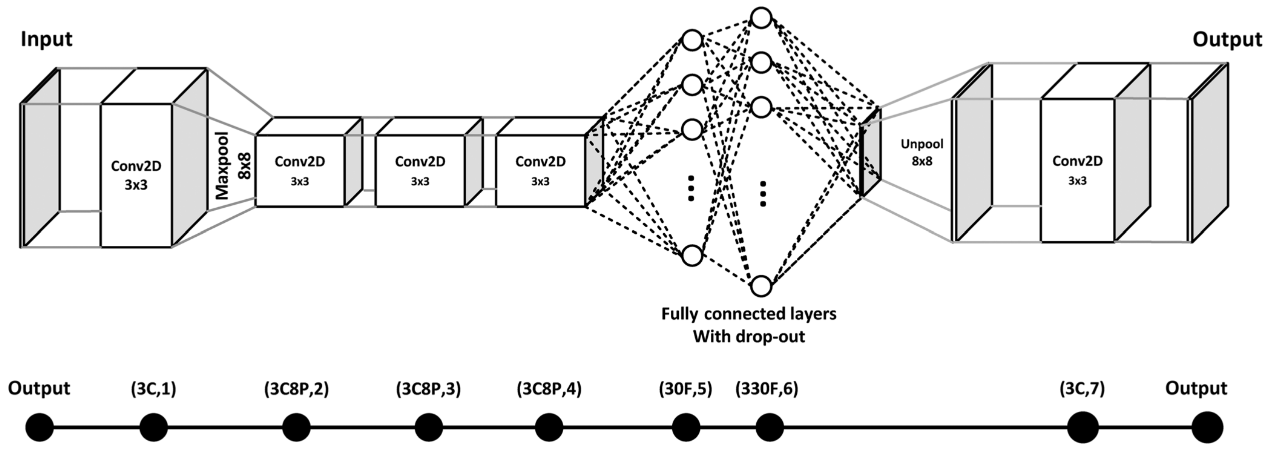


Fig. 20 Top row: network 8, Bottom row: graph corresponds to network 8.

missing the details. None of these designs converged to a reasonable depth estimation model.

The main idea of the SPDNN method is to mix and merge these networks and generate a single model, which includes all the layers of the original models in order to be able to preserve the details and also detect the bigger objects in the scene for the depth estimation task.

A.2 SPDNN Parallelization Methodology

A.2.1 *Graph Contraction*

A consideration while parallelizing NNs is that having the same structure of layers with the same distance from the input might lead all the layers to converge to similar values. For example, the first layer in all of the networks shown in Figs. 13–20 is a 2-D convolutional layer with a 3×3 kernel.

The SPDNN idea uses graph contraction to merge several NNs. The first step is to turn each network into a graph in which it is necessary to consider each layer of the network as a node in the graph. Each graph starts with the input node and ends with the output node. The nodes in the graph are connected based on the connections in the corresponding layer of the network. Note that the pooling and unpooling layers are not represented as nodes in the graph, but their properties will stay with the graph labels, which will be explained later.

Figures 13–20 presents the networks and their corresponding compressed graphs. Two properties are assigned to each node in the graph. The first property is the layer structure, and the second one is the distance of the current node to the input node. To convert the network into a graph, a labeling scheme is required. The proposed labeling scheme uses different signs for different layer structures, C for convolutional layer (e.g., 3C mean a convolutional layer with 3×3 kernel), F for fully connected layer (e.g., 30F means a fully connected layer with 30 neurons), and P for pooling property (e.g., 4P means that the data have been pooled by the factor of 4 in this layer).

Some properties, such as convolutional and fully connected layers, occur in a specific node, but pooling and unpooling operations will stick with the data to the next layers. The pooling property stays with the data except when an unpooling or a fully connected layer is reached. For example, a node with the label (3C8P, 4) corresponds to a convolutional layer with a 3×3 kernel, the 8P portion of this label indicates that the data have undergone 8×8 pooling, and the four at the end indicates that this label is at a distance of four from the input layer. The corresponding graphs, with assigned labels for each network, are shown in Figs. 13–20.

The next step is to put all these graphs in a parallel format sharing a single input and single output node. Figure 21 shows the graph in this step.

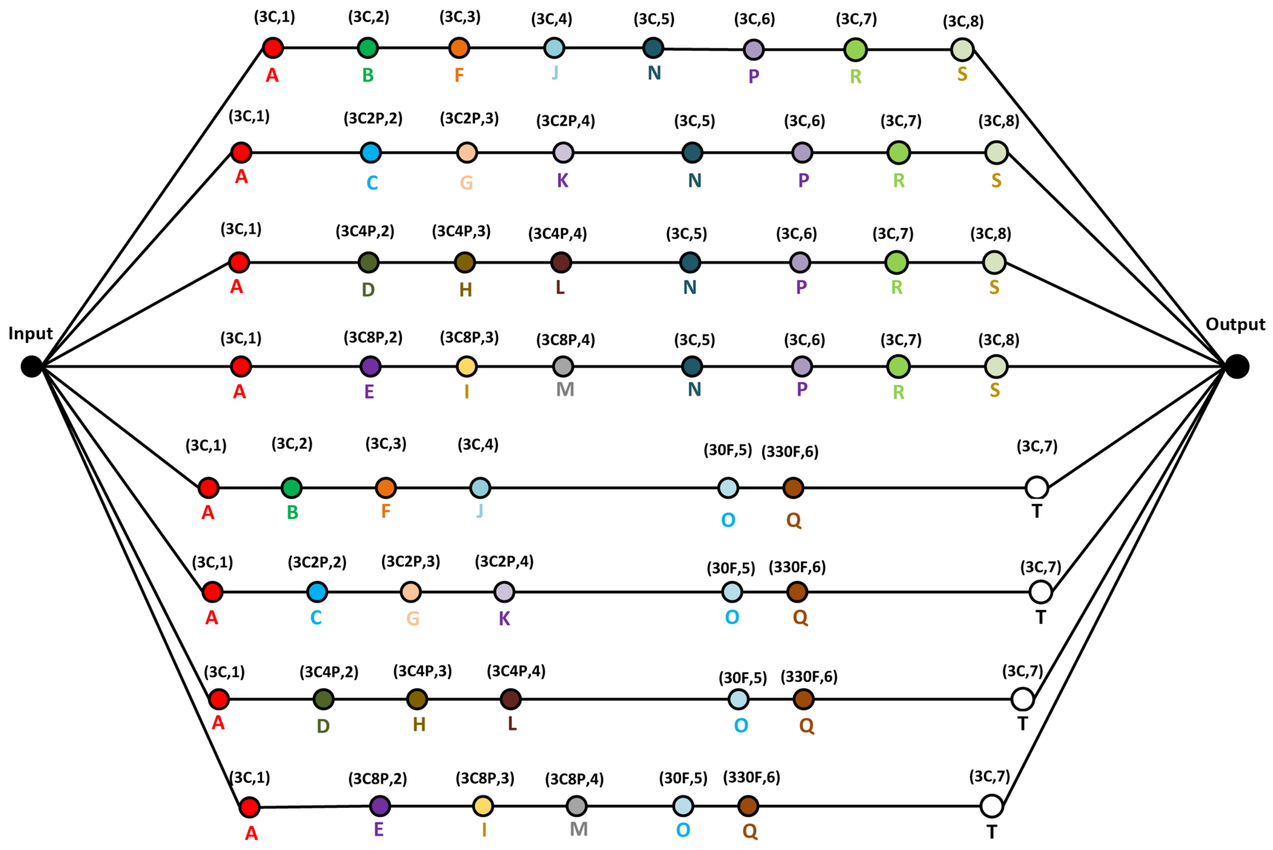


Fig. 21 Parallelized version of the graphs shown in Figs. 13–20 sharing a single input node and single output node.

In order to merge layers with the same structure and the same distance from the input node, nodes with the exact same properties are labeled with the same letters. For example, all the nodes with properties (3C, 1) are labeled with letter A, and all the nodes with the properties (3C2P, 4) are labeled K, and so on.

The next step is to apply graph contraction on the parallelized graph. In the graph contraction procedure, the nodes with the same label are merged to a single node while saving their connections to the previous/next nodes. For instance, all the nodes with label A are merged into one node, but its connection to the input node and also nodes B, C, D, and E are preserved. The contracted version of the graph in Fig. 21 is shown in Fig. 22.

Afterward, the graph has to be converted back to the NN structure. In order to do this, the preserved structural properties of each node are used. For example, node C is a 3×3 convolutional layer that has experienced a pooling operation. Note that the pooling quality will be recalled from the original network.

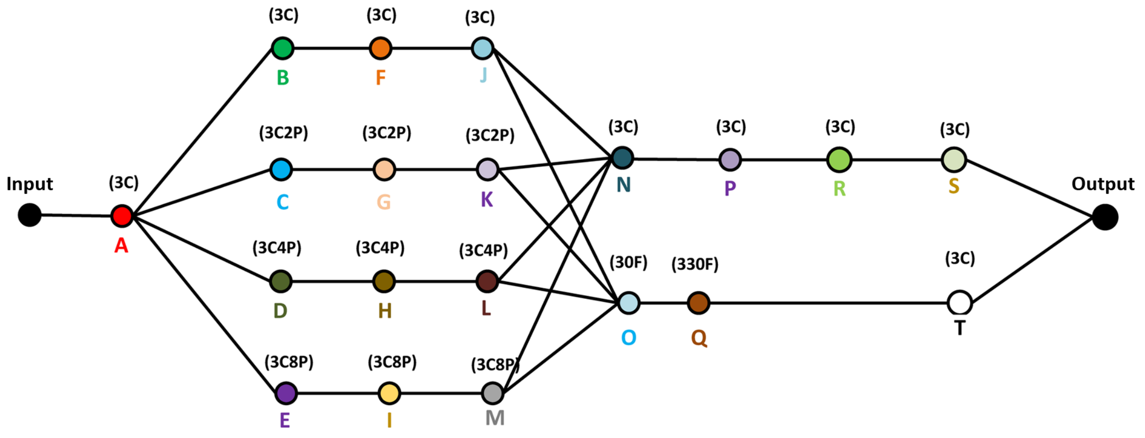


Fig. 22 Contracted version of the big graph shown in Fig. 21.

The concatenation layer is used in the NN in order to implement the nodes wherein several other nodes lead to one node. For example, in nodes N and O the outputs of nodes J, K, L, and M are concatenated with the pooling qualities taken from their original networks.

The graph is translated back to a DNN. The network corresponds to the graph shown in Fig. 22.

A.3 SPDNN: How It Works and Why It Is Effective?

One might ask why the SPDNN approach is effective and what the difference is between this approach and other mixing approaches. Here, the model designed by the SPDNN scheme is investigated in the forward and backpropagation steps. The key component is in the backpropagation step where the parameters in parallel layers influence each other. These two steps are described below:

Forward propagation: Consider the network designed by the SPDNN approach shown in Fig. 23. This exemplary network is made of five subnetworks. Just the general view of the network is shown in this figure and the layers' details are ignored since the main goal is to show the information flow within the whole network.

When the input samples are fed into the network, the data travel through the network along three different paths shown in Fig. 24.

At this stage, the parallel networks are blind to each other, i.e., the networks placed in parallel do not share any information with each other. As shown in Fig. 24, the data traveling in Sub-Net 1 and Sub-Net 2 are not influenced by each other since they do not share any path together, as in Sub-Net 3 and Sub-Net 4.

Backpropagation: While training the network, the loss function calculated based on the error value at the output of the NN is a mixed and merged function of the error value corresponding to every data path in the network. In the backpropagation step, the parameters inside the network update based on this mixed loss values, i.e., this value backpropagates throughout the whole network as shown in Fig. 25. Therefore, at this stage of training, each subnetwork is influenced by the error value from every data path shown

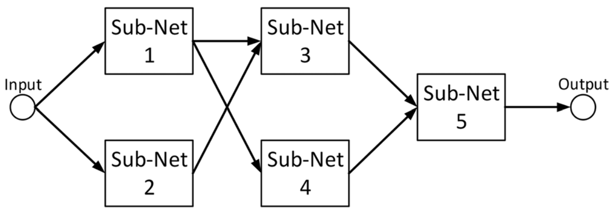


Fig. 23 A network designed using the SPDNN approach. It contains five subnetworks placed in parallel and semiparallel forms.

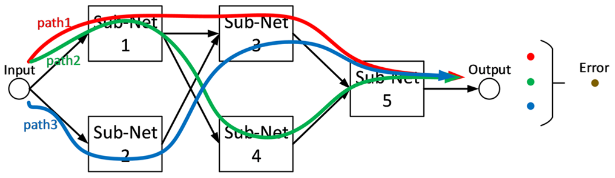


Fig. 24 Forward propagation inside the SPDNN. There are three different paths on which the information can flow inside the network.

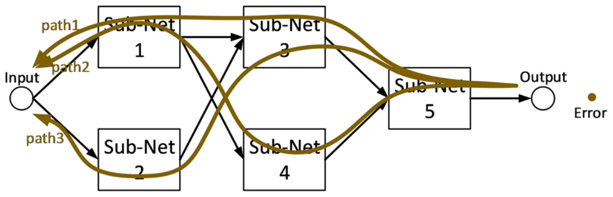


Fig. 25 Backpropagation for SPDNN. The mixed error is backpropagated throughout the network while updating parameters.

in Fig. 25. This illustrates the way each subnetwork is trained to reduce the error of its own path and also the error from the mixture of all paths.

The main difference between the SPDNN approach and other mixing approaches, such as the voting approach, lies in the backpropagation step where different subnets are influenced by the errors of each other and try to compensate for each other's shortcomings by reducing the final mixed error value. In the voting approach, different classifiers are trained independently of each other and they do not communicate to reduce their total error value.

A.3.1 *SPDNN versus Inception*

One of the approaches that has superficial similarities to SPDNN is the inception technique.[64] For clarity, and to aid the reader in understanding, the authors list four significant points of difference between SPDNN and inception with regard to mixing networks.

1. The main idea in SPDNN is to maintain the overall structure of the networks, but to mix them in a reasonable way. For example, if there is a big kernel such as 13×13 in one of the configurations, the SPDNN method always preserves the structure (13×13 kernel) inside the final network. This contrasts with inception,[64] which reduces larger kernels into smaller ones.
2. In the inception method, all the layers are merged into one final layer, which does not happen with the SPDNN approach.
3. The number of the layers in the SPDNN architecture is less than or equal to the number of layers in the original networks. In contrast, the inception idea aims to increase the number of layers in the network by (it breaks down each layer into several layers with smaller kernels).

The SPDNN idea is to design a new network from existing networks that perform well at some task or subtask while the idea in inception is to design a network from scratch.

A.4 Comparisons of Individual Networks

In this section, the behavior of each subnetwork is investigated and compared with the final network. Each of the eight networks proposed in Appendix A.2 is trained on the training data explained in Sec. 2.2. The training is performed in the Lasagne library on top of Theano in Python. Training is done on a standard desktop with an NVIDIA GTX 1080 GPU with 8 GB memory.

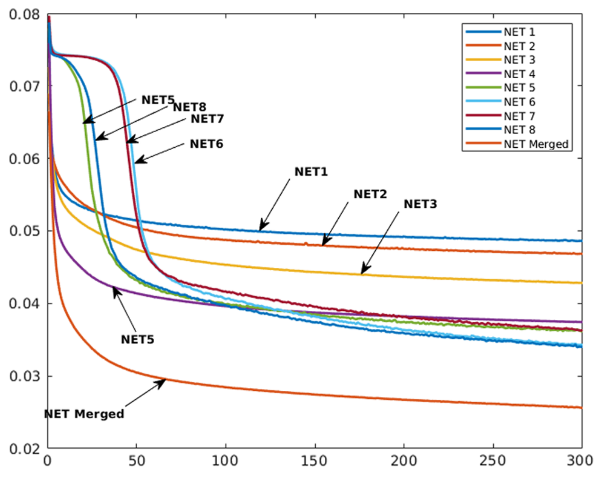


Fig. 26 Training loss for each subnetwork and also the merged network.

In the presented experiments, the MSE value between the output of the network and the target values has been used as the loss function, and the Nestrov momentum technique with learning rate 0.01 and momentum 0.9 has been used to train the network. The training and validation losses for the first 300 epochs are shown Figs. 26 and 27, respectively.

The convergence of the network is significantly increased after merging the networks for both training and validation sets. The fluctuations in validation are less for the merged network, which demonstrates less variance in the loss value. This shows a more stable training process when the SPDNN is applied.

This is also shows how much pooling can help the network to converge and also that networks with fully connected layers are providing better overall outputs, but they miss details in the depth maps since they observe the input sample as a single entity.

The evaluation of each network on the test set is presented in Table 10.

The merged network is giving superior results compared to each individual network for all measurements. This is while the final merged network is designed to have the same number of parameters as each individual subnetwork. This means that the same memory efficiency and also the processing speed for the merged network stay same as for the subnetworks.

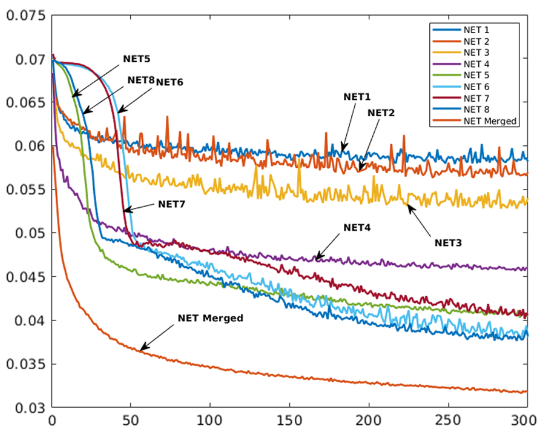


Fig. 27 Validation loss for each subnetwork and also the merged network.

Appendix B: SegNet

SegNet is a fully convolutional semantic image segmentation framework presented in Refs. 51 and 52. This model uses the convolutional layers of the VGG16 network as the encoder of the network and eliminates the fully connected layers, thus reducing the number of trainable parameters from 134 M to 14.7 M, which represents a reduction of 90% in the number of parameters to be trained. The encoder portion of SegNet consists of 13 convolutional layers with ReLU nonlinearity followed by max-pooling (2×2 window) and stride 2 in order to implement a nonoverlapping sliding window. This consecutive max-pooling and striding results in a network configuration that is highly robust to translation in the input image but has the drawback of losing spatial resolution of the data.

This loss of spatial resolution is not beneficial in segmentation tasks where it is necessary to preserve the boundaries of the input image in the segmented output. To overcome this problem, the following solution is given in Ref. 51. As most of the spatial resolution information is lost in

Table 10 Test loss for each subnetwork.

	Net #1	Net #2	Net #3	Net #4	Net #5	Net #6	Net #7	Net #8	Net merged
PSNR	12.2409	12.0153	12.3419	12.4382	12.1458	13.0247	12.3698	13.2233	**15.0418**
MSE	0.0640	0.0672	0.0639	0.0628	0.0676	0.0556	0.0639	0.0527	**0.0378**
RMSE	0.2486	0.2549	0.2471	0.2447	0.2535	0.2293	0.2467	0.2238	**0.1854**
SNR	0.8944	1.2153	1.3206	1.5260	2.5726	0.8139	1.5435	0.9104	**8.822**
MAE	0.2028	0.2081	0.2007	0.1938	0.2009	0.1864	0.2001	0.1798	**0.1442**
SSIM	0.9918	0.9914	0.9918	0.9922	0.9916	0.9927	0.9918	0.9932	**0.9952**
UQI	0.0455	0.0526	0.0856	0.1044	0.0954	0.1067	0.1154	0.1015	**0.8401**

the max-pooling operation, saving the information of the max-pooling indices and using this information in the decoder part of the network preserves the high-frequency information.

Note that for each layer in the encoder portion of the network, there is a corresponding decoder layer. The idea of SegNet is that wherever max-pooling is applied to the input data, the index of the feature with the maximum value is preserved. Later these indices will be employed to make a sparse feature space before the deconvolution step, applying the unpooling step in the decoder part. A batch normalization layer[35] is placed after each convolutional layer to avoid overfitting and to promote faster convergence. Decoder filter banks are not tied to corresponding encoder filters and are trained independently in the SegNet architecture.

Acknowledgments

The research work presented here was funded under the Strategic Partnership Program of Science Foundation Ireland (SFI) and co-funded by SFI and FotoNation Ltd. Project ID: 13/SPP/I2868 on "Next Generation Imaging for Smartphone and Embedded Platforms." This work is also supported by an Irish Research Council Employment Based Programme Award. Project ID: EBPPG/2016/280.

References

1. F. Tombari, S. Mattoccia, and L. D. Stefano, "Stereo for robots: quantitative evaluation of efficient and low-memory dense stereo algorithms," in *11th Int. Conf. on Control Automation Robotics and Vision* (2010).
2. S. Yang et al., "Extraction of topographic map elements with SAR stereoscopic measurement," in *Int. Symp. on Image and Data Fusion* (2011).
3. R. Muñoz-Salinas, E. Aguirre, and M. García-Silvente, "People detection and tracking using stereo vision and color," *Image Vision Comput.* **25**(6), 995–1007 (2007).
4. P. Haigron et al., "Depth-map-based scene analysis for active navigation in virtual angioscopy," *IEEE Trans. Med. Imaging* **23**(11), 1380–1390 (2004).
5. G. Yahav, G. J. Iddan, and D. Mandelboum, "3D imaging camera for gaming application," in *Digest of Technical Papers Int. Conf. on Consumer Electronics* (2007).
6. M. Grosse et al., "3D shape measurement of macroscopic objects in digital off-axis holography using structured illumination," *Opt. Lett.* **35**(8), 1233–1235 (2010).
7. P. Kauff et al., "Depth map creation and image-based rendering for advanced 3DTV services providing interoperability and scalability," *Image Commun.* **22**(2), 217–234 (2007).
8. P. Merkle et al., "The effect of depth compression on multiview rendering quality," in *3DTV Conf.: The True Vision—Capture, Transmission and Display of 3D Video* (2008).
9. S. R. Malireddi et al., "HandSeg: a dataset for hand segmentation from depth images," arXiv:171105944 (2017).
10. L. Tian et al., "Robust 3D human detection in complex environments with depth camera," *IEEE Trans. Multimedia* 1 (2018).
11. J. Liu et al., "Skeleton-based human action recognition with global context-aware attention LSTM networks," *IEEE Trans. Image Process.* **27**(4), 1586–1599 (2018).
12. J. Liu et al., "Skeleton-based action recognition using spatio-temporal LSTM network with trust gates," *IEEE Trans. Pattern Anal. Mach. Intell.* 1 (2017).
13. M. Weber, M. Humenberger, and W. Kubinger, "A very fast census-based stereo matching implementation on a graphics processing unit," in *IEEE 12th Int. Conf. on Computer Vision Workshops, ICCV Workshops* (2009).
14. S. Bazrafkan and P. Corcoran, "Semi-parallel deep neural networks (SPDNN), convergence and generalization," arXiv:171101963 (2017).
15. S. Bazrafkan and P. M. Corcoran, "Pushing the AI envelope: merging deep networks to accelerate edge artificial intelligence in consumer electronics devices and systems," *IEEE Consum. Electron. Mag.* **7**(2), 55–61 (2018).
16. B. Freedman et al., "Depth mapping using projected patterns," Google Patents, Patent No. US8493496B2 (2013).
17. A. Govari et al., "Tissue depth estimation using gated ultrasound and force measurements," Google Patents, Patent No. EP3189785A1 (2016).
18. D. Nair, "3D Imaging with NI LabVIEW," http://www.ni.com/white-paper/14103/en/ (24 August 2016).
19. C. Niclass et al., "A 100 m-range 10-frame/s 340×96-pixel time-of-flight depth sensor in 0.18-μm CMOS," in *Proc. of the ESSCIRC (ESSCIRC)* (2011).
20. C. Niclass et al., "Design and characterization of a 256×64-pixel single-photon imager in CMOS for a MEMS-based laser scanning time-of-flight sensor," *Opt. Express* **20**(11), 11863–11881 (2012).
21. D. Scharstein and R. Szeliski, "High-accuracy stereo depth maps using structured light," in *Proc. of IEEE Computer Society Conf. on Computer Vision and Pattern Recognition* (2003).
22. R. K. Gupta and S.-Y. Cho, "A correlation-based approach for real-time stereo matching," *Lect. Notes Comput. Sci.* **6454**, 129–138 (2010).
23. C. Zhang et al., "MeshStereo: a global stereo model with mesh alignment regularization for view interpolation," in *IEEE Int. Conf. on Computer Vision (ICCV)* (2015).
24. D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *Int. J. Comput. Vision* **47**(1–3), 7–42 (2002).
25. K. R. Kim and C. S. Kim, "Adaptive smoothness constraints for efficient stereo matching using texture and edge information," in *IEEE Int. Conf. on Image Processing (ICIP)* (2016).
26. X. Huang, Y. Zhang, and Z. Yue, "Image-guided non-local dense matching with three-steps optimization," *ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci.* **III-3**, 67–74 (2016).
27. A. Li et al., "Coordinating multiple disparity proposals for stereo computation," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, Nevada (2016).
28. M. Shahbazi et al., "Revisiting intrinsic curves for efficient dense stereo matching," *ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci.* **III-3**, 123–130 (2016).
29. J. Žbontar and Y. LeCun, "Stereo matching by training a convolutional neural network to compare image patches," *J. Mach. Learn. Res.* **17**(1), 2287–2318 (2016).
30. J. T. Barron and B. Poole, "The fast bilateral solver," arXiv:151103296 (2016).
31. E. T. Psota et al., "MAP disparity estimation using hidden Markov trees," in *IEEE Int. Conf. on Computer Vision (ICCV)* (2015).
32. J. Kowalczuk, E. T. Psota, and L. C. Perez, "Real-time stereo matching on CUDA using an iterative refinement method for adaptive support-weight correspondences," *IEEE Trans. Circuits Syst. Video Technol.* **23**(1), 94–104 (2013).
33. B. Yegnanarayana, *Artificial Neural Networks*, PHI Learning Pvt. Ltd., New Delhi (2009).
34. N. Srivastava et al., "Dropout: a simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.* **15**(1), 1929–1958 (2014).
35. S. Ioffe and C. Szegedy, "Batch normalization: accelerating deep network training by reducing internal covariate shift," arXiv:150203167 (2015).
36. F. Mahmood, R. Chen, and N. J. Durr, "Unsupervised reverse domain adaptation for synthetic medical images via adversarial training," *IEEE Trans. Medical Imaging* 1 (2018).
37. F. Mahmood and N. J. Durr, "Deep learning-based depth estimation from a synthetic endoscopy image training set," *Proc. SPIE* **10574**, 1057421 (2018).
38. F. Mahmood and N. J. Durr, "Deep learning and conditional random fields-based depth estimation and topographical reconstruction from conventional endoscopy," arXiv:171011216 (2017).
39. F. Liu et al., "Learning depth from single monocular images using deep convolutional neural fields," *IEEE Trans. Pattern Anal. Mach. Intell.* **38**(10), 2024–2039 (2016).
40. A. Saxena, J. Schulte, and A. Y. Ng, "Depth estimation using monocular and stereo cues," in *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence*, Hyderabad, India (2007).
41. D. Eigen, C. Puhrsch, and R. Fergus, "Depth map prediction from a single image using a multi-scale deep network," in *Proc. of the 27th Int. Conf. on Neural Information Processing Systems*, Montreal, Canada, Vol. 2 (2014).
42. R. Garg et al., "Unsupervised CNN for single view depth estimation: geometry to the rescue," *Lect. Notes Comput. Sci.* **9912**, 740–756 (2016).
43. C. Godard, O. M. Aodha, and G. J. Brostow, "Unsupervised monocular depth estimation with left-right consistency," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)* (2017).
44. D. Xu et al., "Monocular depth estimation using multi-scale continuous CRFs as sequential deep networks," *IEEE Trans. Pattern Anal. Mach. Intell.* 1 (2018).
45. D. Xu et al., "Multi-scale continuous CRFs as sequential deep networks for monocular depth estimation," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)* (2017).
46. N. Silberman et al., "Indoor segmentation and support inference from RGBD images," *Lect. Notes Comput. Sci.* **7576**, 746–760 (2012).

47. M. Menze and A. Geiger, "Object scene flow for autonomous vehicles," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)* (2015).
48. A. Saxena, S. H. Chung, and A. Y. Ng, "3-D depth reconstruction from a single still image," *Int. J. Comput. Vision* **76**(1), 53–69 (2008).
49. A. Saxena, S. H. Chung, and A. Y. Ng, "Learning depth from single monocular images," in *Proc. of the 18th Int. Conf. on Neural Information Processing Systems*, Vancouver, British Columbia (2005).
50. H. Javidnia and P. Corcoran, "A depth map post-processing approach based on adaptive random walk with restart," *IEEE Access* **4**, 5509–5519 (2016).
51. V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: a deep convolutional encoder-decoder architecture for image segmentation," arXiv:151100561 (2015).
52. A. Kendall, V. Badrinarayanan, and R. Cipolla, "Bayesian segNet: model uncertainty in deep convolutional encoder-decoder architectures for scene understanding," arXiv:151102680 (2015).
53. A. Kendall, V. Badrinarayanan, and R. Cipolla, "Caffe implementation of SegNet," https://github.com/alexgkendall/caffe-segnet (18 April 2018).
54. G. J. Brostow et al., "Segmentation and recognition using structure from motion point clouds," *Lect. Notes Comput. Sci.* **5302**, 44–57 (2008).
55. S. Lee et al., "Robust stereo matching using adaptive random walk with restart algorithm," *Image Vision Comput.* **37**, 1–11 (2015).
56. I. Sutskever et al., "On the importance of initialization and momentum in deep learning," in *Int. Conf. Machine Learning*, Vol. 28, pp. 1139–1147 (2013).
57. W. Zhou et al., "Image quality assessment: from error visibility to structural similarity," *IEEE Trans. Image Process.* **13**(4), 600–612 (2004).
58. W. Zhou and A. C. Bovik, "A universal image quality index," *IEEE Signal Process. Lett.* **9**(3), 81–84 (2002).
59. K. Pearson, "Note on regression and inheritance in the case of two parents," *Proc. R. Soc. Lond.* **58**, 240–242 (1895).
60. J. Kopf et al., "Joint bilateral upsampling," *ACM Trans. Graph* **26**(3), 96 (2007).
61. A. Saxena, M. Sun, and A. Y. Ng, "Make3D: learning 3D scene structure from a single still image," *IEEE Trans. Pattern Anal. Mach. Intell.* **31**(5), 824–840 (2009).
62. T. Zhou et al., "Unsupervised learning of depth and ego-motion from video," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)* (2017).
63. Y. Kuznietsov, J. Stückler, and B. Leibe, "Semi-supervised deep learning for monocular depth map prediction," in *2017 IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)* (2017).
64. C. Szegedy et al., "Going deeper with convolutions," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition* (2015).

Shabab Bazrafkan received his BSc degree in electrical engineering from Urmia University, Urmia, Iran, in 2011 and his MSc degree from Shiraz University of Technology (SuTECH) in telecommunication engineering, image processing branch in 2013. Currently he is a PhD student at the National University of Ireland, Galway (NUIG) and also works with Fotonation Ltd. His field of working is deep neural networks and neural network design.

Hossein Javidnia received his master's degree in information technology engineering from the University of Guilan, Iran, in 2014. He received his PhD in electrical engineering from the National University of Ireland, Galway, in 2018. His current research interests include image processing, machine vision, and automotive navigation.

Joseph Lemley received his BS degree in computer science from Central Washington University in 2006. After working in industry and cofounding a start-up, he went back to Central Washington University in 2014 and received his master's degree in computational science in 2016. Currently he is a PhD student at the National University of Ireland, Galway (NUIG) funded by Fotonation Ltd. under the IRCSET "Employment PhD" program. His field of work is machine learning using deep neural networks for tasks related to computer vision.

Peter Corcoran is an IEEE fellow; 500+ technical publications and patents; 80+ peer reviewed papers and articles; 100+ international conference papers; coinventor on 300+ granted US patents, university professor for 28 years; member IEEE Consumer Electronics Society 20+ years; editor-in-chief and founding editor of IEEE Consumer Electronics Magazine; former vice-dean of research and graduate studies (7 year tenure) in the College of Engineering and Informatics at NUI Galway; cofounder of several start-up companies including FotoNation (www.fotonation.com); and industry consultant and expert witness.

Appendix G

Contributions to Deep Learning Education

G.1 Deep Learning For Consumer Devices and Services

IEEE Consumer Electronics Magazine is a peer reviewed technical magazine with an impact factor of 3.273. Throughout the course of my PhD I have published 4 articles in this magazine on deep learning for consumer devices aimed at researchers and practitioners in the consumer electronics industry.

The first of these articles "Deep Learning for Consumer Devices and Services" won a best paper award from the IEEE Consumer Electronics society in 2018, and as of this writing is my second most cited publication. This first article in the series aims to introduce deep learning to a CE audience starting from the basics. It also serves to augment the introduction to this thesis, providing details on foundational techniques in deep learning.

The next article in the series, "Deep Learning for consumer devices and services II", describes the current state of "Edge AI", focusing on emerging trends and technologies. It is the cover article on the September/October issue of IEEE Consumer Electronics Magazine and discusses deep learning in "edge" scenarios including AR/VR, DMS, bio-metrics and other applications. The emergent shift from cloud based to device based inference is also discussed.

The third article explains traditional augmentation techniques and the fourth describes techniques for learnable augmentation" for a CE audience. This last article was written concurrently with this thesis and borrows heavily from section 2.2. These last two articles have been accepted for publication and are expected to be published in early 2020.

G.2 Contributions to Deep Learning Education

I'm grateful to have had the opportunity to share my knowledge and passion for deep learning research in a formal classroom setting at NUIG. I was also able to run a successful workshop at IEEE GEM 2018, attracting 20 participants, which enabled dissemination of this knowledge beyond the classroom.

Since Engineering students are increasingly likely to encounter technologies based on deep learning in their careers, it is important to give them a foundation in deep learning techniques and methodologies, even if they don't intend to directly utilize neural networks in their future.

G.2.1 Mobile Device Technology: A First DL Lab

My first opportunity to design a lab at NUIG was for electronic engineering masters students in the Mobile Device Technology (EE5116) class in 2017 organized as a double lab. This was the 7th and final lab for the students. On the day of the lab, I started with a brief lecture on the fundamentals of Deep Learning and passed out a Raspberry Pi disk image containing code and libraries that the students would need for completing their assignments, which were to be done in Tensorflow, Keras, and Python. A copy of the lab handout I wrote for this class can be found in appendix H.

Designing such a disk image was more difficult in 2017 than it would be in 2020 as Tensorflow had no official ARM support at that time. Fortunately, an unofficial Tensorflow port existed for ARM, so I included a prebuilt version of this and other dependencies in the image.

The first portion of the lab was designed to introduce students to deep learning tools and to give them a chance to utilize a neural network, both at inference time and training time, as well as to appreciate the amount of time needed for training as opposed to inference. When they were finished with part 1, they understood why we train neural networks on powerful GPUs and not small ARM processors such as those in the Raspberry Pi but just as importantly, they learned that these processors are more than good enough for inference.

In part 2 of the lab, the students downloaded pretrained weights for Squeezenet [80]. I supplied them with a Python script that would take an image as a command line argument and, using supplied weights and class labels, provide a prediction and confidence value. Squeezenet is a binarized network that was chosen due to the fact that such quantized or compressed networks are important in embedded devices, where computing resources are scarce.

As in part one, the students used the Raspberry Pi, the supplied code, and instructions to identify the classes and confidences of 5 images of everyday animals and objects. They were asked to download these images from the internet using an image search tool such as Google images and were able to choose any image they liked. At this point they were not to worry if the image they picked was one of the classes that the network had been trained on yet. During the lab students were asked if they could find images that were misclassified by the model.

At this point, the students had been given the experience of training a model and of using a model for specific images.

The last section of the lab (part 3) was designed to be more difficult; the previous two tasks allowed them to treat a network as a "black box", but the final task required modifying the supplied code to create an independent assessment of the network's performance on a separate dataset, which the students made themselves during the lab.

This is an important skill because the reported accuracy for neural networks are seldom as high as when tested on a separate dataset. This allowed us to have a discussion on generalization, datasets, and being skeptical about how well reported results will be reflected in a related but different use-case.

G.2.2 The Edge AI Workshop at IEEE GEM 2018

In the spring of 2018, I became involved in helping to organize the IEEE Games, Entertainment and Media conference, which would take place in Galway, Ireland. A copy of the handout for the "hands on" section of the workshop can be seen in Appendix I.

Given the success of the Deep Learning lab for the masters students, I decided to expand the masters lab to a workshop/tutorial that would be co-located with this conference. Participants paid between 150 dollars to 400 dollars depending on IEEE membership and whether they also were registered for the full conference. Fotonation/Xperi paid for 7 of their engineers to attend, but we had attendees from all over the world. I served as the workshop chair and organizer. My colleague Shabab Bazrafkan also contributed to the materials but I had the overall responsibility to co-ordinate the material and its delivery.

The first part of the workshop (from 9:00 to 11:00) contained practical and theoretical lectures on deep learning and edge AI. These lectures were intended to give participants with no previous machine learning background the ability to understand what Edge AI is about and to feel comfortable participating in the last two sessions 11:30 to 16:00. The lectures were given by Shabab Bazrafkan, Peter Corcoran, and myself.

Since this was an all day event and we had a budget for materials, I was able to expand the Masters lab to include training on GPUs.

Before the workshop I prepared 20 USB sticks with Ubuntu Preinstalled with Tensorflow-gpu, Caffe, Keras and everything the participants would need to train deep learning algorithms.

After the workshop, each participant received one of these USB sticks so that they could have the same build environment, tools, and GPU drivers used during the lab. This was important because the process of configuring a PC with a GPU to run deep learning software was still error prone and could sometimes result in days of work just to get the environment built correctly for engineers. With this stick, getting a working development environment to get started with deep learning simply required, they select the provided USB drive at boot.

For the first hands-on session of the workshop participants would train a classifier on MNIST using the GPUs and also on the Raspberry Pis. Participants were put in groups of 2 to enable collaborative learning. Besides this training, the first hands-on session was nearly identical to the Masters lab described previously.

The second hands-on session (from 13:30-16:00) involved building various Edge AI technology prototypes using a Raspberry Pis and Movidius sticks.

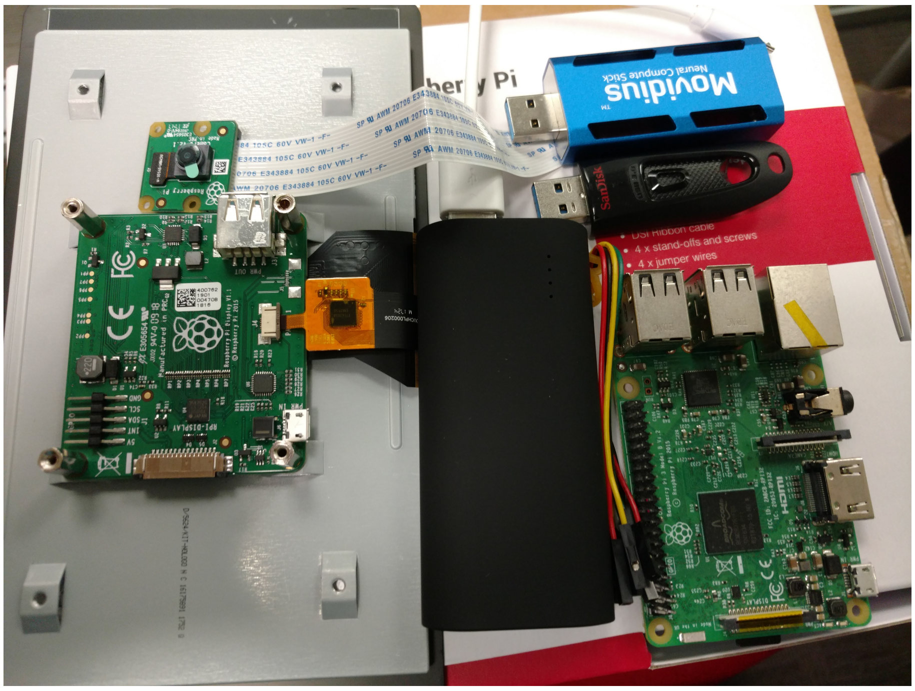


Fig. G.1 An Edge-AI kit supplied to participants in the Edge-AI workshop at IEEE GEM 2018 in Galway Ireland.

The first such task was to take the network they had previously trained on MNIST with the GPU and use it to create a real-time hand-written digit recognizer using a Raspberry Pi with a PiCAM, a Movidius stick, and a 7 inch touch screen.

I provided the code to do this, but to make their digit recognizer work well on actual hand-written digits (which they were encouraged to write themselves on a sheet of paper) they needed to figure out how to modify the provided code to change the color images to black and white images that look like those the network was trained on. I distributed working "answer" code as well as hints to the TAs ahead of time so that if anyone was struggling they could get help from the TAs or from myself, but few needed it.

An important lesson from this first task was that even a network that has over 99% accuracy on hand written digits must be given data that looks similar to what it was trained on if it's to be useful in any way.

The final task involved building a battery powered, real-time object detector based on MNIST with their Raspberry PI, PiCAM, and Movidius stick. We purchased a collection of toys that participants could use to see if their device worked properly.

I ran tests before the workshop to ensure that the batteries we provided could provide power for such a use case for at least 4 hours before running out of charge. Feedback from the workshop was very positive and the participants seemed to enjoy it.

G.2.3 Mobile Device Technology and ESAP labs

As a result of student feedback from 2017 and previous years, it was desired that the Embedded Systems Applications Programming class at NUIG be redesigned to utilize continuous lab-based assessment in place of a final exam.

Under Peter Corcoran's supervision, my colleague Aoife and I redesigned the labs to fit this style of assessment and also updated the class with new topics for 2018.

At the same time, we redesigned the Mobile Device Technology labs taken by the graduate students similarly, although they also had exams.

We designed 9 of these lab classes, with topics such as shell scripting and Linux, Python, version control with git, image processing, IOT devices and lastly Deep Learning.

The labs were designed such that approximately 70% of the grade came from correctly following instructions and paying attention to the previous labs, whereas the remaining 30% was intended to be challenging. Thus the usual format was to have 2 or 3 "challenges".

To prepare the students for the deep learning section, I gave several lectures on deep learning to the ESAP undergraduate and MDT graduate class. This was my first time giving formal lectures at NUIG.

The deep learning lab was largely based on our practical experiences from the workshop except that due to resource constraints we could not provide the students access to suitable GPUs to use for training networks. Because this lab was more challenging than the others, students were given 2 lab sessions to complete it.

Another difference was that a final challenge was added whereby students would modify the YOLO code so that the boxes predicted with YOLO were run through the Squeezenet network they used previously, thus expanding the number of classes that could be predicted from 20 to 1000. The handout given to the students for the deep learning lab can be found in Appendix J

Appendix H

Worksheet for MDT lab 7 (EE5116) on Deep Learning

Lab EE5116 - Week 7

Content: First experience using Deep Learning Neural Networks.
Evaluate Neural Network performance.

Pre-requisite: 16 Gb SD-cards

Milestones:

1) Flash Raspbian image onto SD-card (Win32DiskImager, SD Formatter)
2) Download the required documents (provided on BlackBoard)
3) Train a network using the RPi
4) Evaluate the network using an image
5) Evaluate the network using 3 sets of images

Part 0: Flash Raspbian image

1. Copy the Raspbian image provided by the TAs and unzip its contents (approx. 16Gb)
2. Download and install Win32DiskImager and SD Card Formatter
3. Insert the sd-card in your computer using an sd-card reader
4. Format the sd-card you have (clear all partitions) with SD Card Formatter
5. Open Win32DiskImager and set the path to the 16 Gb image
6. Flash the image onto the sd-card by clicking on 'Write' (approx. 10 minutes)

Part 1: -- Training a deep neural network

1. Insert SD card and start up pi
2. Enter the "deeplearning" directory.
3. Run the mnist demo with python mnist.py

Allow the CNN to train a while to get a feel for the time it takes to train on a raspberry pi compared to a GPU. Let it run for at least 3 minutes.

4. Before you stop it, write down the current epoch, the "ETA, the loss, and the accuracy in a separate file for a TA to view later.

Question: What epoch will be reached by the Raspberry Pi if we don't stop it? The student making the best guess is going to receive 0.25 points extra ☺

Part 2: -- Using/Deploying a deep neural network.

Since training on a raspberry pi is typically wasteful in terms of time and power consumption, we will instead use a pre-trained model. You can find pre-trained networks for a variety of tasks online and these can be deployed to a PI.
It's important to choose a model wisely as many of them are excessively large and unwise to use on embedded or mobile electronics. For this exercise we'll be using **squeezenet** [1], a binarised network that achieves similar accuracy to larger well known models on imagenet while being about 2 orders of magnitude smaller. For example, VGG16 is 528 MB and inception V3 is 92 MB whereas squeezenet is 5 MB with similar performance. Squeezenet can be deployed on mobile phones, raspberry pi's, and other resource constrained hardware. The size of the network is 5 MB.

1. Make a new directory under deeplearning called squeeze
2. Enter that directory
3. Download a pretrained version of squeezenet with weights in a format tensorflow will understand

```
wget https://github.com/avoroshilov/tf-squeezenet/raw/master/sqz_full.mat
```

4. Download class labels:

```
wget https://github.com/avoroshilov/tf-squeezenet/raw/master/synset_words.txt
```

5. Download script deeplearning.py from blackboard.

6. Use google image search to locate 5 images of everyday objects or animals. Download these images and use the script to get the class predictions and confidences for each of the 5 images you download.
7. Write the class predictions and confidences to a file for later discussion with a TA.

Part 3: -- Evaluating a Deep neural network.

Neural networks often report accuracy information on specific datasets, but these datasets may or may not be similar enough to yours to be useful for you. In this exercise, you will create a very small dataset, annotate that dataset with class labels, and evaluate squeezenet on specific classes.

1. Open the file synset_words.txt and take a moment to browse the class labels. The class labels look like this: n12267677, beside each class label is one or more corresponding words.
2. Decide on three classes that you want to use to evaluate the network.
3. Using google image search or another tool of your choice, locate and download 5 images for each class by using the words that correspond to the class label.
4. Prepare a text document in the same directory that your files are in. You may format your file in CSV format like this:

```
classlabel , filename
classlabel , filename
```

5. Write all 15 of your image file names and class labels to the file using the format above and save it as annotations.txt
6. Make a copy of the python script from Part 2 and save it as **deeplearning2.py**

7. In the function **def main():** , note the line **imgname=sys.argv[1]**. It accepts the name of an image file as its first argument. Instead we want it to accept annotations.txt that we made in the previous stage and to create two python lists: a list of file names and a list of classes from this annotations file. You can use the python "split" function or **csvreader** (use online resources!!)
8. You'll notice that the code (deeplearning.py) loads just one image and reports the predicted class and confidence (probability) for that prediction. **Modify the code so that it will provide predictions for all 15 of your images**. The efficiency of the code will be taken in consideration when evaluated by a TA.
9. Write code to automatically check the predictions from above against the "real" or "ground truth" labels that you read from annotations.txt
10. Change the output of the program to report the total number of times the predicted labels were not the same as the true labels.

REFERENCES

[1] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size," pp. 1–13, 2016.

Appendix I

GEM 2018 Workshop

Hands on session 1 (11:30-13:00)

Part 1A: -- Training a deep neural network on GPU

In this exercise you will train a neural network on the famous MNIST dataset of hand written digits.

The computers you are using contain NVIDIA 1080 TI GPUs.

1. Log in to PC and open a terminal window with CTRL+ALT+T
2. Type nvidia-smi and note the details of the graphic cards including any tasks running on them. nvidia-smi is one of the most useful commands for determining if any jobs are running on the GPU as well as resource utilization information.
3. Navigate to ~/workspace/ncappzoo/tensorflow/mnist and type make train. This starts the process of training the network on MNIST and will provide a graph file for later use.
4. When training is complete a graph file will be created. Let this process run as you proceed to the next steps.

Part 1B: -- Training a deep neural network on RPIs

1. Connect mouse, keyboard, and monitor to the Raspberry PI using provided material.
2. Insert SD card and start up pi,
3. Enter the "~workspace/deeplearning" directory.
4. Run the mnist demo with python3 mnist.py

Allow the CNN to train a while to get a feel for the time it takes to train on a raspberry pi compared to a GPU. Let it run for at least 3 minutes.

Part 2: -- Using/Deploying a deep neural network.

Since training a neural network on a raspberry pi is typically wasteful in terms of time and power consumption, we will instead use a pre-trained model. You can find pre-trained networks for a variety of tasks online and these can be deployed to a PI.
It's important to choose a model wisely as many of them are excessively large and unwise to use on embedded or mobile electronics. For this exercise we'll be using **squeezenet** [1], a binarized network that achieves similar accuracy to larger well known models on imagenet while being about 2 orders of magnitude smaller. For example, VGG16 is 528 MB and inception V3 is 92 MB whereas squeezenet is 5 MB with similar performance. Squeezenet can be deployed on mobile phones, raspberry pi's, and other resource constrained hardware.

1. Inside the workspace directory make a new directory called squeeze
2. Enter that directory
3. Download a pretrained version of squeezenet with weights in a format tensorflow will understand

```
wget https://github.com/avoroshilov/tf-squeezenet/raw/master/sqz_full.mat
```

4. Download class labels:

```
wget https://github.com/avoroshilov/tf-squeezenet/raw/master/synset_words.txt
```

5. Type cp workspace/deeplearning.py /workspace/squeeze/
6. Use Bing image search to locate 5 images of everyday objects or animals. Download these images to the same directory your script is stored in and use the script to get the class predictions and confidences for each of the 5 images you download.
7. Did the network perform well or poorly? Was your accuracy the same as this network on imagnet? (top 1 accuracy is ~57% and top 5 accuracy ~80%).

Part 3: -- Evaluating a Deep neural network.

Neural networks often report accuracy information on specific datasets, but these datasets may or may not be similar enough to yours to be useful for you. In this exercise, you will create a very small dataset, annotate that dataset with class labels, and evaluate squeezenet on specific classes.

1. Open the file synset_words.txt and take a moment to browse the class labels. The class labels look like this: n12267677, beside each class label is one or more corresponding words.
2. Decide on three classes that you want to use to evaluate the network.
3. Using google bing search or another tool of your choice, locate and download 5 images for each class by using the words that correspond to the class label.
4. Prepare a text document in the same directory that your files are in. You may format your file in CSV format like this:

```
classlabel , filename
classlabel , filename
```

5. Write all 15 of your image file names and class labels to the file using the format above and save it as annotations.txt
6. Make a copy of the python script from Part 2 and save it as **deeplearning2.py**

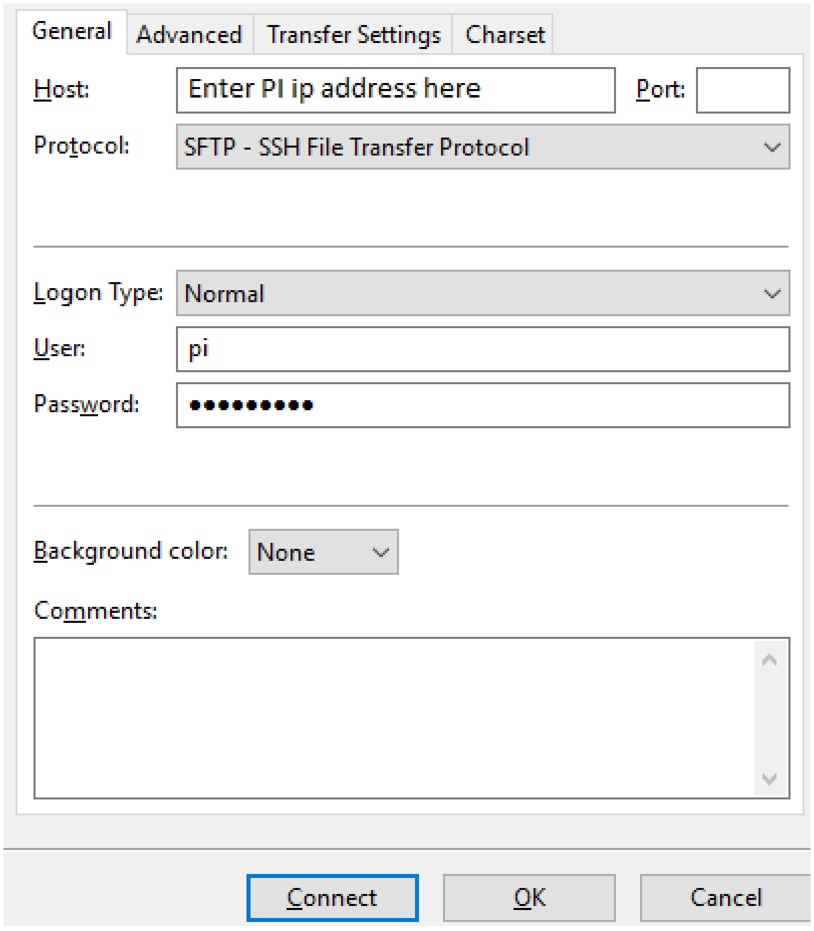
7. In the function **def main():** , note the line **imgname=sys.argv[1]**. It accepts the name of an image file as its first argument. Instead we want it to accept the annotations.txt file that we made in the previous stage and to create two python lists: a list of file names and a list of classes from this annotations file. Hint: Use the python “split” function or **csvreader**.
8. You’ll notice that the code (deeplearning.py) loads just one image and reports the predicted class and confidence (probability) for that prediction. **Modify the code so that it will provide predictions for all 15 of your images**.
9. Write code to automatically check the predictions from above against the “real” or “ground truth” labels that you read from annotations.txt
10. Change the output of the program to report the total number of times the predicted labels were not the same as the true labels.

For more information on squeezenet view: https://arxiv.org/pdf/1602.07360.pdf

Hands on session 2 (13:30-14:50 [as late as 16:00 for those who wish to stay longer])

Part 1: -- Assembling and using a battery powered Edge AI device

1. Connect Camera, 7 inch touch screen, and plug in movidious stick as shown in Adrian's demonstration. Connect keyboard.
2. Boot up PI.
3. Using keyboard find IP address of PI using ifconfig
4. On your PC, open an SSH connection to the PI using: ssh pi@[ip] in a terminal window. Experiment with a few commands to make sure everything is configured correctly. If you want to run any gui programs you can connect with ssh -XY : ssh pi@[ip] first.
5. On your PC open filezilla.
6. In the file menu select site manager.
7. Enter the information for your PI as shown below and click connect



Host: Enter PI ip address here
Port:
Protocol: SFTP - SSH File Transfer Protocol
Logon Type: Normal
User: pi
Password: ●●●●●●●●●●
Background color: None
Comments:

8.
9. Copy the mnist graph file to [directory] on the raspberry PI.

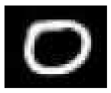
Information covered in this part will help speed up development and debugging in the remaining parts.

Part 2: -- Assembling a battery powered Digit recognizer

As you saw in the first session, your model performs very well on the MNIST dataset. But how well does it work on real data from a PI CAM? In this subsection the graph file from the first session will be used to make a hand written digit recognizer.

1. Write digit on piece of paper.
2. Move the graph file you copied to ~workspace/mnist
3. cd ~workspace/mnist
4. mnist.py uses the NCS to perform inference on live frames from a camera. Before we run it there are a few things you should know.
    a. We're using the NCS 2.0 SDK, which is the latest library for the movidius neural compute sticks because it has several advantages over 1.0, including the ability to queue multiple graphs and inputs. The 2.0 SDK was released recently and is incompatible with the 1.0 SDK. If you find code written for the 1.0 version of the library, the conversion is easy. You can follow the directions here to do it: https://movidius.github.io/ncsdk/ncapi/python_api_migration.html
5. Type python3 mnist.py
6. Try to get it to recognize a written digit by pointing the camera at it. Did it work as well as you thought it would? What's going wrong?

   The problem is, your network was trained on images that are 28 by 28 with each digit centered, written with white on black with non-noisy backgrounds. Neural networks are not magic, they can only work with data that is similar to information they are trained on. To get your live digit recognizer to work you'll need to use some image processing techniques make the digits you wrote look more those in the mnist dataset.



7. See if you can get your input similar to this by converting the images to white on black, cropping, and any other techniques you can think of. If you are struggling with this, ask a TA for a "hint" or for a working code snippet.
8. Now that you've made these improvements, try again. Center the camera directly over the digit you want to recognize and you should get a better result.

Part 3: -- Real time Object Detector.

In this task a battery powered, live object recognizer is developed based on the yolo algorithm.

Delegates set up battery powered real time yolo on a movidius stick with a picam and a raspberry pi for object recognition. For more information about the yolo algorithm see: https://pjreddie.com/darknet/yolo/

1. On your raspberry PI, navigate to ~/workspace/yolo
2. In idle3 or nano, open runyolo.py
3. This script won't run as is. Read through the code and locate the parts marked with "TODO" comments. If you need any help, don't hesitate to ask a TA for hints or working solutions. Use the mnist example as a reference.
4. When you've implemented the changes, walk around the room and see what predictions your hand held object recognizer gives you for various people and objects. At what distance does it stop working? There are a variety of toys for you to test your object recognizer on. See what happens when you change perspectives and backgrounds or when you adjust the threshold.

Part 4: -- Bonus round!

If additional time is left and, you are free to train, test, or modify any demo from the ~workspace/ncappzoo or ncsdk/examples located both on the USB stick and on the raspberry PI's SD card or to try some bigger models on the NVIDIA GPUs. Feel free to test out any ideas you may have. As always, don't hesitate to contact the TAs if you would like advice.

Appendix J

Deep learning labs in 2018

ESAP lab 9 and 10

Task 1 – Try training a digit recognizer on a raspberry pi.

1. Connect mouse, keyboard, and monitor to the Raspberry PI using provided material.
2. Insert SD card and start up pi,
3. Create and enter a directory called ESAP9
4. Clone the esap 9 on github.
5. Download mnist.py from the lab 9 blackboard python3 mnist.py

Allow the CNN to train a while to get a feel for the time it takes to train on a raspberry pi compared to a GPU. Let it run for at least 3 minutes.

6. Stop the program with ctrl+c, copy and paste the network's progress information and save it to a file called progress.txt
7. This code was training a digit recognizer on your raspberry pi. This would take less than minutes to fully train on a modern GPU and approximately 8 hours to finish training on your PI.

Task 2: -- Using/Deploying a deep neural network.

Since training a neural network on a raspberry pi is typically wasteful in terms of time and power consumption, we will instead use a pre-trained model. You can find pre-trained networks for a variety of tasks online and these can be deployed to a PI.

It's important to choose a model wisely as many of them are excessively large and unwise to use on embedded or mobile electronics. For this exercise we'll be using **squeezenet** [1], a binarized network that achieves similar accuracy to larger well known models on imagenet while being about 2 orders of magnitude smaller. For example, VGG16 is 528 MB and inception V3 is 92 MB whereas squeezenet is 5 MB with similar performance. Squeezenet can be deployed on mobile phones, raspberry pi's, and other resource constrained hardware.

1. Copy deeplearning.py to a new directory called squeeze
2. Enter that directory
3. Download a pretrained version of squeezenet with weights in a format tensorflow will understand

```
wget https://github.com/avoroshilov/tf-squeezenet/raw/master/sqz_full.mat
```

4. Download class labels:

```
wget https://github.com/avoroshilov/tf-squeezenet/raw/master/synset_words.txt
```

5. Use Bing image search to locate 5 images of everyday objects or animals. Download these images to the same directory your script is stored in and use the script to get the class predictions and confidences for each of the 5 images you download.
6. To use deeplearning.py simply give it the name of an image file as an argument.
7. Did the network perform well or poorly? Was your accuracy the same as this network on imagnet? (top 1 accuracy is ~57% and top 5 accuracy ~80%).

8. Save the class predictions, filepath, and accuracy for these 5 images that you chose in a file called predictions.txt for your TA to view later.

Task 3: -- Evaluating a Deep neural network.

Neural networks often report accuracy information on specific datasets, but these datasets may or may not be similar enough to yours to be useful for you. In this exercise, you will create a very small dataset, annotate that dataset with class labels, and evaluate squeezenet on specific classes.

1. Open the file synset_words.txt and take a moment to browse the class labels. The class labels look like this: n12267677, beside each class label is one or more corresponding words.
2. Decide on three classes that you want to use to evaluate the network.
3. Using bing search locate and download 5 images for each class by using the words that correspond to the class label.
4. Prepare a text document in the same directory that your files are in. You may format your file in CSV format like this:

```
classlabel , filename
classlabel , filename
```

5. Write all 15 of your image file names and class labels to the file using the format above and save it as annotations.txt
6. Make a copy of the python script from Part 2 and save it as **deeplearning2.py**
7. In the function **def main():** , note the line **imgname=sys.argv[1]**. It accepts the name of an image file as its first argument. Instead we want it to accept the annotations.txt file that we made in the previous stage and to create two python lists: a list of file names and a list of classes from this annotations file. Hint: Use the python “split” function or **csvreader**.
8. You’ll notice that the code (deeplearning.py) loads just one image and reports the predicted class and confidence (probability) for that prediction. **Modify the code so that it will provide predictions for all 15 of your images**.
9. Write code to automatically check the predictions from above against the “real” or “ground truth” labels that you read from annotations.txt
10. Change the output of the program to report the total number of times the predicted labels were not the same as the true labels.

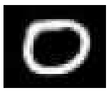
For more information on squeezenet view: https://arxiv.org/pdf/1602.07360.pdf

Task 4: -- Assembling a battery powered Digit recognizer

The mnist model you are using reports an accuracy of greater than 99%. But how well does it work on real data from a PI CAM? In this subsection the graph file from the first session will be used to make a hand written digit recognizer.

1. Write digit on piece of paper.

2. Assemble python tablet with picam and NCS stick. Feel free to use your battery for a power source if you'd like to be more mobile.
3. digit_recogniser.py uses the NCS to perform inference on live frames from a camera. Before we run it there are a few things you should know.
    a. We're using the NCS 2.0 SDK, which is the latest library for the movidius neural compute sticks because it has several advantages over 1.0, including the ability to queue multiple graphs and inputs. The 2.0 SDK was released recently and is incompatible with the 1.0 SDK. If you find code written for the 1.0 version of the library, the conversion is easy. You can follow the directions here to do it: https://movidius.github.io/ncsdk/ncapi/python_api_migration.html
4. In digit_recogniser.py, change the line "graph_filename = "mnist_inference.graph"" to graph_filename = "inference.graphs"
5. Type python3 digit_recogniser.py
6. Try to get it to recognize a written digit by pointing the camera at it. Did it work as well as you thought it would? What's going wrong?
   The problem is, your network was trained on images that are 28 by 28 with each digit centered, written with white on black with non-noisy backgrounds. Neural networks are not magic, they can only work with data that is similar to information they are trained on. To get your live digit recognizer to work you'll need to use some image processing techniques make the digits you wrote look more those in the mnist dataset.



7. See if you can get your input similar to this by converting the images to white on black, cropping or using other computer vision techniques.
8. Now that you've made these improvements, try again. Center the camera directly over the digit you want to recognize and you should get a better result.

Task 5: -- Real Time Object Detector.

In this task a battery powered, live object recognizer is developed based on the yolo algorithm.

You will set up a battery powered real time yolo based object detector on a movidius stick with a picam and a raspberry pi for object recognition. For more information about the yolo algorithm see: https://pjreddie.com/darknet/yolo/

1. On your raspberry PI, open runyolo.py
2. This script won't run as is. Read through the code and locate the parts marked with "TODO" comments. Use the mnist example as a reference.
3. When you've implemented the changes, walk around the room and see what predictions your hand held object recognizer gives you for various people and objects. At what distance does it stop working? There are a variety of toys for you to test your object recognizer on. See what happens when you change perspectives and backgrounds or when you adjust the threshold.
4. Try pointing it at a youtube video playing on a laptop or phone.

Task 6: -- Combining everything

There are a limited number of classes in yolo. Wouldn't it be nice if we could get more detailed information about what each bounding box has in it?

For this task, copy the file you made from task 6 to a new file called task7.py

1. Modify tasky7.py so that it automatically saves each bounding box as an image every half second. Each of these images should be stored in a subdirectory called task7_images

2. Copy the program you used in task 2 to a new file called task7_background.py
3. task7_background.py should scan the task7_images directory every 2 seconds and print to the console the top 5 class labels of any newly added images. This means you'll need to keep track of previously read images in a list or use some other technique to ensure you don't reclassify the same images. This program should run continuously until closed and should use tensorflow on the raspberry PIs cpu as in task 2.
4. In different terminals start both task7_background.py and task7.py
5. Walk around the room for at least 30 seconds, allowing your pi to take images of a diverse range of things automatically and test your code. You should see bounding boxes drawn on your touchscreen and detailed classes for each object in a terminal window (with quite a bit of lag). This task would be more impressive if you could do it outside or in a space with a wide variety of objects and the battery would certainly allow this. Depending on the weather a short bonus project for additional points may be announced involving this during the lab.