Image Classification Using Deep Convolutional Neural Networks

Final Report for CS39440 Major Project

Author: Theodoros Nikopoulos-Exintaris (thn2@aber.ac.uk)
Supervisor: Dr. Chuan Lu (cul@aber.ac.uk)

4th May 2016
Version: 1.0 (Release)

This report was submitted as partial fulfilment of a BSc degree in Computer Science and Arificial Inteligence (GG47)

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB
Wales, UK

Declaration of originality

I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the regulations on Unacceptable Academic Practice from the University's Academic Quality and Records Office (AQRO) and the relevant sections of the current Student Handbook of the Department of Computer Science.
- In submitting this work I understand and agree to abide by the University's regulations governing these issues.

Name Theodoros Nikopoulos-Exintaris.....

Date 04/05/2015.....

Consent to share this work

By including my name below, I hereby agree to this dissertation being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name Theodoros Nikopoulos-Exintaris.....

Date 04/05/2015.....

Acknowledgements

I'd like to thank Keiron O' Shea and Dr. Chuan Lu for volunteering to answer questions regarding neural networks and providing useful feedback and bibliography to get started on the topic.

We would also like to thank the Aberystwyth University Computer Science, The Bioinfomatics research team, as well as the Machine Learning lecture staff for providing resources and information that enabled us to write this dissertation.

Finally I would like to thank my family for supporting me during my 3 years of this degree as well as providing emergency financing without which this project would be impossible, as well as my mother for providing last minute proofreading for this report.

Abstract

An investigation of the use of Deep Convolutional Neural Networks for image classification using statistical means. This report aims to test the hypothesis that DCNNs are suitable to classify images in a product style environment. This report will investigate a number of hyper parameters and their impact in performance, practicality, and training in an attempt to produce a successful classifier that can be reproduced by third party sources. Wherever possible our aim is full disclosure of any assumptions and any compromises that have been made to enhance the reproducibility of our results.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Background & Objectives

1.1 Introduction

This project is an investigation into training Artificial Neural Networks for the purpose of image classification and labelling. We will explore alternative approaches in computer vision to solve this problem arriving to the what is considered the current state of the art in machine learning, comparing it to a number of different alternatives, both contemporary but also historical.

1.1.1 Motivation

Perhaps the first question we might wish to ask ourselves is ”why we wish to embark on this project in the first place”. Computer vision is a field of computer science in essence, involving extracting information and building models of the real world; this might involve sensors, cameras or perhaps even audio.

There are numerous applications and types of computer vision. The problem we are looking in particular is identifying images and labelling them, extracting metadata from the image itself. Naturally this has many applications. Self-driving cars, search engines, and any problem where we need a computer to be able to identify images.

Our interest is particularly related to image acquisition and labelling. Suppose a user has a large set of images and you are looking for a particular one. They know what it is but not its title. Or perhaps its title does not hold semantic meaning to them, but wish to search the semantics of the image itself.

1.1.2 Machine Learning and Pattern Recognition

To solve this problem, computer scientists set out to devise a set of ’rules’ and algorithms that were good at solving these problems. Eventually, it became apparent that no single algorithm could solve the problem for all sets of cases as the ’rules’ change between cases. The rules that define a cat are different the ones that define a dog and so on, so one would have to write rules for each edgecase. Where you needed to solve this kind of problem one would analyse the data they expect to receive and think of features they are interested in extracting. This approach is still used for some CV problems.

A solution that emerged for this was machine learning (ML), that is sets of algorithms that could be used to derive the rules based on data. Most machine learning algorithms work by identifying patterns in the data, either via statistical analysis as in Bayesian Learning, or other means. Suppose you have a set of data describing cats and dogs. Cats are small, but there is both small and large types of dogs. Most cats have pointed ears while some dogs have floppy ones. A machine learning algorithm would process a set of examples of cats and dogs, then it would derive patterns that can be used to classify them.

1.2 Background

1.2.1 SVMs and other Machine Learning Alternatives

To use an SVM and most other ML methodologies, you begin by extracting the features you wish to learn from. If for example you are looking at a shape classifier, you would begin by considering what defines your classes. For this example, perhaps the number of edges might be a feature, perhaps the relative length of each straight edge, or maybe the number of detected straight edges.

These can be extracted from an image using computer vision techniques, or in principle, machine learning would be used in a table of features that have already been extracted. Extracting features can be quite challenging on its own and can take a lot of effort. Having extracted the features however, it is easy to gain insight on the decisions a ML algorithm makes.

1.2.2 Artificial Neural Networks

Artificial Neural Networks(ANN) are a type of machine learning inspired by the way some biological organisms process stimulus and learn. In essence an ANN is a very naive emulation of a "meat computer", with the biological processes being replaced with mathematical approximations of the actual process of biological neurons.

Being that these are only approximations, they have several components which help them function and emulate select processes of real neurons.

1.2.2.1 Advantages and Disadvantages of ANNs

ANNs have many advantages and disadvantages to conventional ML methods.

Some of the most important ones include:

- They do not require a feature extraction step
- They can easily be applied in many different problems

It is possible to apply ANNs in to solve many problems, by just feeding in appropriate training data and get reasonably accurate results in some cases, matching the state of the art for that problem.

As for disadvantages:

- They are difficult to train both in terms of requirements and speed

- It is comparatively difficult to gain insight on why a neural network has learned something

Recent advances in ANNs have greatly improved on both of these areas. GPU compute in particular, has made training quite large neural networks practical, on consumer computers. There is also very active research at the moment in improving insight and visualising neural network decisions. We aim to talk in more detail about all of these later in our report.

1.2.2.2 Datasets

With most methodologies having data to test against is important. With ML, data is what drives the whole process so to produce any sort of solution to this problem one needs to decide on the dataset to use. For classification problems a dataset might be a set of inputs and outputs, separated in a set to train on and a set to test performance against afterwards.

For our network we are looking at datasets with actual images with associated labels. There is several of that type of image dataset, we will discuss them in more detail in a later section of this report.

1.3 Analysis

To complete this project, we must solve a number of problems and answer a number of questions. As part of our analysis we identified some of them, while several others would emerge as we embarked on our project.

1.3.1 Types of ML and ANN architectures in use

To start we should look at what models have been successful in the past for this kind of problem. After all this area has several decades of research behind it which we would not be able to replicate from scratch.

Looking at older results also enables us to get a baseline for what our model might be capable of doing and provides goals to hit or exceed. For this report we will be looking at SVMs, Multi Layer Perceptrons (MLP) and Convolutional Neural Networks (CNN)

1.3.2 Variables to test and optimise

We need to decide on the architecture of our neural network. For CNN choices appear to be relatively simple, with most networks opting for very similar architectures. However, there are still aspects that can be varied.

We will need to investigate different ways we can vary our model and attributes that we can change in order to limit the scope of the research area. This process involves, reading documentation for our solutions as well as research into the technology we are using to determine the highest value optimisations to make.

1.3.3 Are CNNs ready for market?

One other interesting question to answer, as a conclusion, is whether CNNs are ready to be deployed in commercial products and services. Is it possible to produce something robust enough for such uses?

1.4 Research Method

To tackle any large undertaking, whether it is by a team of a single developer and their tenacity, or involving a team of expert code acrobats working in unison, we as developers, require a set of routines and rituals; this project is no different.

Prompted by a discussion on our Agile Methodologies module, we begun thinking Agile as a solution to this problem. We concluded that Agile would be appropriate to the spirit of our project, however due to this being a solo endeavour, we opted not to adopt any mainstream Agile methodologies. What we did instead is: we took elements we liked from Extreme Programming and combined them with parts of Scrum to create a methodology more closely tailored to our project.

What we needed was a way to drive quick prototyping and creative experimentation in a controlled manner, such that we could plan experiments while maintaining a rudimentary level of quality; while the nature of this project does not necessarily require the end product to be production-ready, since there is no customer nor a production environment. However, poor software quality would only hinder the experimental progress.

In our 'creative experimentation' process, we incorporate the concept of a sprint, but since we expect to go through a small iteration daily, we restrict time to the equivalent of a weekly sprint. Because we lack customers and a product owner, we replace them with our supervisor reporting progress after each week's sprint. The discussion is used to drive the direction and goals of the next sprint.

We are also using a form of continuous integration to deploy our changes, and GIT source control to track changes and experiments. We evaluate progress based on feedback from results and from milestones at the end of each 'sprint'. We considered Github issues as well but the were considered superfluous. Polyvinyl Carbonate Figure Debugging was implemented however. See fig. 1.1

Figure 1.1: A Pretty Soldier of Hard Work and Guts: Takaya Noriko.

The overall lifecycle looks like this:

- First few cycles are exclusively dedicated to research and coming up with a set of requirements for the test infrastructure and programs that need to be developed to drive the project.
- After this we focus on designing and testing models with the aid of the basic software we designed in stage 1.
- New features will be added as needed to advance the project. Data loading, performance monitoring visualisation etc.

For experiments we begin to identify the variables and results we wish to test e.g. size of convolutional filters, then we identify the pre-requisites to run the tests. i.e. do we have the framework to run the experiment, and of course the desired data and whether we can extract them. For the third one we found that there was no need to add more output to run some experiments in our scope. Based on the results of an experiment we decide on what other experiments to run afterwards. Since experiments can take many hours to run, we need to plan a maximum of 2 experiments, a day with a expectation of perhaps doing less on some days or more on others.

Finally, we planned some basic overall schedule we use in 3 major milestones, research, implementation and writeup. Adjustments were to be made based on progress, initial planning was to use 2 weeks for research, 6 weeks for experiments and additional research as needed and 4 weeks for the report.

We have decided to name this methodology 'Hard Work and Guts Programming'. In the spirit of this, we are using this figurine of Takaya Noriko from the 1988 OVA 'Aim for the Top! Gunbuster' as poster child and rubber duck debugging surrogate $(fig.1.1)$

Chapter 2

Methods

2.1 Introduction

This chapter is concerned with introducing the concepts and methodologies we will be using to complete this project. The aim is to familiarise the viewer with the theory behind our experiments, as well as hopefully to illustrate, in clear and concise terms, the motivation and methods behind them. This, combined with the next chapter, will give a complete overview of the experiment design and implementation which form the core of this project.

2.2 The Importance of Data in Machine Learning

For any ML problem the first and most important problem after determining the problem itself is deciding on the data.

Any ML algorithm begins with data ($TrainingData$) and ends with data ($TestData, PerformanceData$). The purpose of machine learning is to extract information based on examples, so without examples it would be impossible to progress.

Of course that's not all we use data for. In order to evaluate our models, data is also used to drive other aspects of the ML lifecycle. We also need data to evaluate our model's performance, and our performance data is itself a very important data decision in ML.

When it comes to data having more of it is usually best, but there are cases where less, better quality data is more important. Indeed, unless data is of sufficient quality we cannot use it. Quality in this sense refers to our ability to learn from it. There are many qualities a set of data might possess that might make its quality poor.

If for example we train a model to classify between dogs and cats but have only 1 example of a cat for every 10 dogs, then perhaps our model will perform poorly, while in other tasks that's not really an issues. Different qualities of data are needed or desirable for different tasks.

Performance data is important as it allows us to validate decisions and to direct our efforts in such a way as to achieve good general performance which is usually the objective in ML, the use of a trained model to map new examples to a learned output.

A data driven approach to problems is essential in ML.

2.3 Dataset

A dataset is a set of data that is to be used as inputs for an ML algorithm either for Training, Testing, or both. A dataset will usually consist of a set of inputs and a set of outputs.

What those sets of inputs and outputs are will depend on the type of dataset and potentially the problem the dataset is designed to solve. For instance a dataset might be a set of images of Donald Trump and pictures of assorted barnyard animals, labeled as 'trump' and 'not-trump' This dataset might be suitable to train a model to identify pictures of Donald Trump.

If we wished to use a dataset like this to identify individual animals we might struggle however. In simple terms an ML algorithm can only learn things contained in the dataset it trains on. This creates a demand for many datasets to be created for many different purposes.

Creating, Normalising, and Evaluating a Dataset is a hard and complicated task, involving a lot of labour. Yet datasets are numerous as the problems they wish to solve. Choosing the correct dataset then goes hand in hand with selecting a problem.

2.4 Artificial Neural Networks

2.4.1 Introduction

Artificial Neural Networks are a series of ML models and methods that are based on a naive model of the biological function of neurons. They have origins as early as the 1940s [10] when Mathematical Biophysicists were studying the functions of neurons and were attempting to model them mathematically.

Neural Networks have progressed tremendously since those days but many of the fundamental concepts remain the same. At their core they still have basic simulated neurons and they still are trying to completely emulate the functions of a real neuron whether than be human, cat, or another organism.

As the name implies an ANN is a Network of Artificial Neurons, the way these neurons are connected to each other and 'fire' give them their properties including abilities to 'learn'. The way ANN's learn is quite useful for many problems where ML is traditionally applied.

Especially advantageous is an ANN's ability to learn without the need for feature extraction. Indeed ANNs are well known for being able to recognise patterns in even raw data.

The main disadvantage touted on the other hand is that because everything happens inside the ANN it is very hard to understand what your model is actually doing. This has lead to several anecdotes like Google's ANN learning that birds have branches as part of their legs, or some researchers that were trying to detect tanks from photos which ended up teaching their ANN that desert sceneries mean tank.

Of course this is not completely true and it the last few years we have seen a lot of research being done on this exact same topic. The days of ANNs being blackboxes is almost over.

While ANN is an umbrella term for this type of model there are many components that serve different functions or try to emulate different features of biological brain. Some of these components and models will be detailed in the following sections.

2.4.2 Optimisers

An optimisation function is one of the core concepts of ML. Its purpose is quite simple. To optimise the weights such that the ML algorithm learns. For our Networks we will be looking at one of many options that are available. Stochastic Gradient Descent.

Stochastic Gradient descent works by computing the negative gradient of the cost function by tracing through the network and then back-propagating through the true label. The main benefit of SGD over normal gradient descent is that it updates one value at a time which makes it really efficient especially to train on batches which reduce the computation overhead significantly. [7]

The second benefit is they are suitable for online learning which we will not be looking at in this report.

2.4.3 Loss Functions

2.4.4 Activation Functions

Activation functions are the parts of the ANN that give its neurons their biological like property. Real Neurons do not just fire when stimulated. Instead they have complicated chemical thresholds and other mechanisms that determine whether a neuron fires passing a message along to another in the network.

Activation Functions together with weighs are meant to simulate this mechanism. We will explain two of the ones we will be using bellow.

2.4.4.1 ReLU

2.4.4.2 Softmax

2.4.5 Perceptrons

A perceptron, or fully connected perceptron is one of the most basic ANN architectures available. It consists of an input layer of neurons, one or more 'hidden layers', and finally an output layer.

We call it a fully connected model because each layer is connected to every neuron of every adjacent layer. This structure is reflected in figure 2.1 Perceptron Networks with more than 1 layer are called Multi-Layer Perceptrons, or MLP for short.

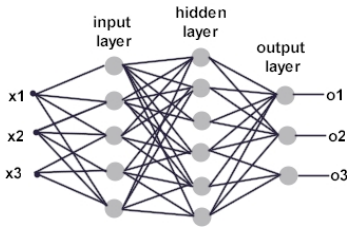


Figure 2.1: Single Layer Perceptron
Source

Multi-Layer Perceptrons are quite powerful networks on their own and until a few decades ago were the dominant ANN in use, for many tasks they still perform quite well, while they have a few key disadvantages in vision problems like the problem we are addressing.

An MLP requires as many input neurons as there are channels so for an RGB image of size 32x32 we would need 3072 inputs because each of those inputs is fully connected that means

that the number of calculations that is necessary can grow quite quickly.

For simple solving tasks this is not really an issue and as we will discuss later MLPs are still used as a component in a CNN.

2.4.6 Convolutional Neural Networks

A Convolutional Neural Network, otherwise referred to as a CNN is a type of Neural Network that has been available since around the 1990s. It is inspired by studies on how the cat visual system works. [7] It is quite common in ML literature and especially in the world of ANNs to find references to Biology.

A common ANN structure works by collecting coarse features through a series of Convolutional Filters which are aggregated via pooling. Then this is fed back to a set of finer filtering which extract more features. The process is repeated until the desired detail is extracted and the outputs are fed through an MLP and converted to answers for our ANN.

A cat's visual cortex works much the same by extracting features with increased detail as it goes from input to output.

Feature Maps and Convolutions work on two concepts. Sparse connectivity and encouraging locality. Neurons are made such that they only connect with neurons that are near in their 'visual field' this is justified because more often than not 'interesting things' tend to be quite near each other, an object will usually be a single body and so on so forth.

The main genefit of a CNNs approach is it can extract more information from an image than an MLP alone can while being relative inexpensive to build. Despite of this benefit, practical ANNs have only been around for a few years with Computer Scientists like LeCan trying to promote their use in computer vision [8]. The advent of GPU compute has also been revolutionary in the field.

Because of their involvement one of the basic CNN designs has come to be know as LeNet.

2.4.6.1 Structuring a CNN

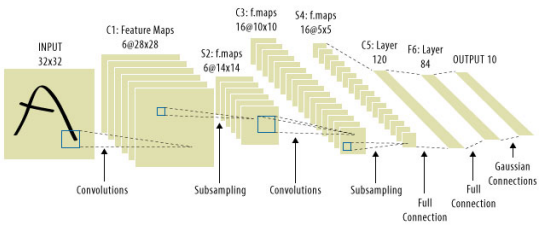


Figure 2.2: Example of a common CNN design. Source

As we previously discussed a CNN consists of a number of layers figure 2.2 illustrates the model. We have 2 or more feature layer, followed by a pooling layer, usually max pooling, followed by more feature layers. The feature layers are just convolutions.

Max pooling layers work by taking n inputs and outputting their max value. So suppose you had a max pooling layer of a pooling of 2x2, what would happen is you get 1 output with the max activation of 4 neurons.

With each iteration of this you accumulate features such that at the end your MLP can solve the feature maps to associate them to outputs. Depending on the type of problem the so-

lutions are put into a different kind of output layers. In a multi-label classification problem like outs we woudl use something such as softmax from which the probabilities are mapped to classes.

2.5 Network training

One of the ways of training a neural network is through the Stochastic Gradient Descent Optimizer. What we will be focusing on with this report is two concepts, an epoch, a period of time where out ANN encounters the complete set of training data and gets evaluated with optimisations applied, and a batch.

For each epoch, we go through each batch which is a portion of the total training images and optimise our neural network weights. The Neural Network trains by completing each epoch successfully until it is time to stop. Hyper Parameters like the bach-size, momentum, and learning rate affect this significantly but we decided to keep these locked after some early tweaking and focus on different optimisations.

2.6 Network evaluation

2.6.1 Confusion Matrices

A confusion matrix is a way of representing classification error as a metric to use for evaluating a model and gaining an insight over the neural network. Each axis represents the model's prediction and actual test data.

		Predicted	
		Positive	Negative
True	Positive	True Positives (TP)	False Negatives (FN)
	Negative	False Positives (FP)	True Negatives (TN)

Figure 2.3: Example of a confusiong matrix Source

You then aggregate the co-ordinates on the matrix. The way this is done you would have totals for all True class A classied as B and so on. Correct results should produce a strong pattern down the diagonal while strong confusions will produce off diagonal readings. By looking at the numbers we can evaluate our model on a per class basis.

2.6.2 Classification Reporting

2.7 Overfitting

Overfitting is a term we use for when we optimise our loss in such a way that our model begins training successfully but loses general classification performance which is our primary goal.

The problem is that ML methods will always optimise for our data so especially models that can extract exact features can be prone to overfit.

Overfitting affects neural networks more than other ML techniques because we don't have much control over the features that are extracted so we may not directly prevent it.

There is no real solution to overfitting as it is an inherent flaw with ML, but there is methods to mittigate and detect its effects.

2.7.1 Dropout

2.7.2 Data Augmentation

With data augmentation we hope to enhance our data by trying to trick our ANN into thinking that there is more data than there actually is. The trick to this is to try and distort the image just emough so the features are still clearly visible, but we know have a form of 'noise' added into the image.

We can do this by blending noise into an image, rotations, flips, shifts etc. Anything that will alter the exact pixels of the image but leave the image intact has the potential to work as data augmentation.

2.7.3 Cross Validation

Cross Validation is a statistical method to improve reliability of results. The basic idea is you use more than one set of testing data in order to obtain more accurate classification results but there is a variety of methods to do this.

The method we are interested in is Cross Validation Split. It works by spliting our training data into a training set and validation set. Once we have a validation set we check and monitor our validation loss at the end of each epoch.

This gives us a hint about the general classification performance while training. We can use this as a method to detect overfitting before it becomes significant and take action to prevent either by giving up or by applying countermeasures. We can see an example of such a use in figure 2.4

2.7.4 Training History

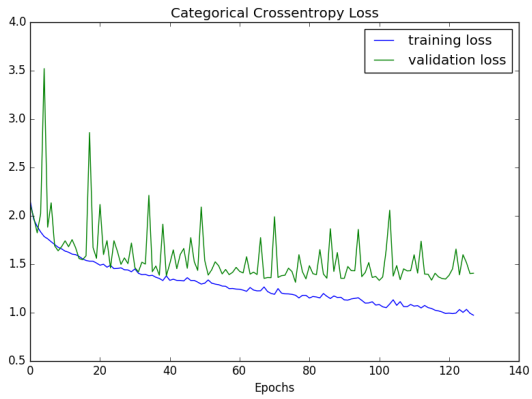


Figure 2.4: Example of a Training History

When training new models one of the things that is desirable to know is the training history. This is meant to indicate overfitting over time. When training we wish to know this to tweak hyper parameters to avoid it.

Figure 2.4 shows a plot of the training history of a deep MLP $(8layers)$ as you can see loss is going down as the SGD algorithm optimizes it, the oscillating pattern is a signature of the SGD optimiser in this case.

Validation loss is another history item we wish to have. Being able to get a forecast of classification performance while training has several uses when training models both in the training itself and as a diagnostic tool.

2.7.5 Early Stopping

One of the things we might want to do when overfitting does settle in to stop and load our most promising model up to that point. If you again look at figure 2.4 you will again notice that loss drops pretty consistently throughout the training oscillation aside but remember, for a good general classifier, loss isn't enough. Instead Validation Loss should be considered.

We notice a series of local minima in the graph. A naive solution would be to stop wherever our VAL loss begins increasing again because overfitting has started but we see our VAL loss has both local minima and eventually gets on track. If we just stopped when one was encountered we could be quiting training early.To prevent this we can do a number of things

The most popular solution is patience. We store the best model then let the model run for a while and see if Val loss improves from the min. If it does reset the waiting period and repeat if not return the best model. This way we can mitigate the risk of overfitting and get closer to the global minimum which is our goal.

Chapter 3

Implementation and Experimentation

3.1 Summary

To produce this research experiments must be run, and for this to happen experiments and the framework to carry them out need to be designed and implemented. This chapter will explain the key decisions and work carried out to produce our experimental results.

3.2 Choice of Dataset

For any machine learning task one of the first tasks that need to be undertaken once the nature of the task is determined is finding an appropriate dataset. The choice of dataset is important as it can dictate a number of our later decisions.

For image classification and labeling problems three of there are three common datasets that are used. We will briefly detail each one and choose which one to use.

There are of course other alternatives out there but these have been around for reasonably long and are small enough to work with the limited resources we had available at the start of the project.

3.2.1 MNIST

This dataset is a reduced subset of the NIST handwritten digit database and is made up of 60,000 hand written digits for training examples and 10,000 testing examples. The dimensions of each image is 28x28px.

These are used as a proof of concept for the neural network and a successful model using this dataset should be able to recognise a wide variety of normalised hand-written digits.

This dataset is considered one of the easiest of the ones we considered to perform well on, with top performing models getting accuracies as high as 99.79 percent.

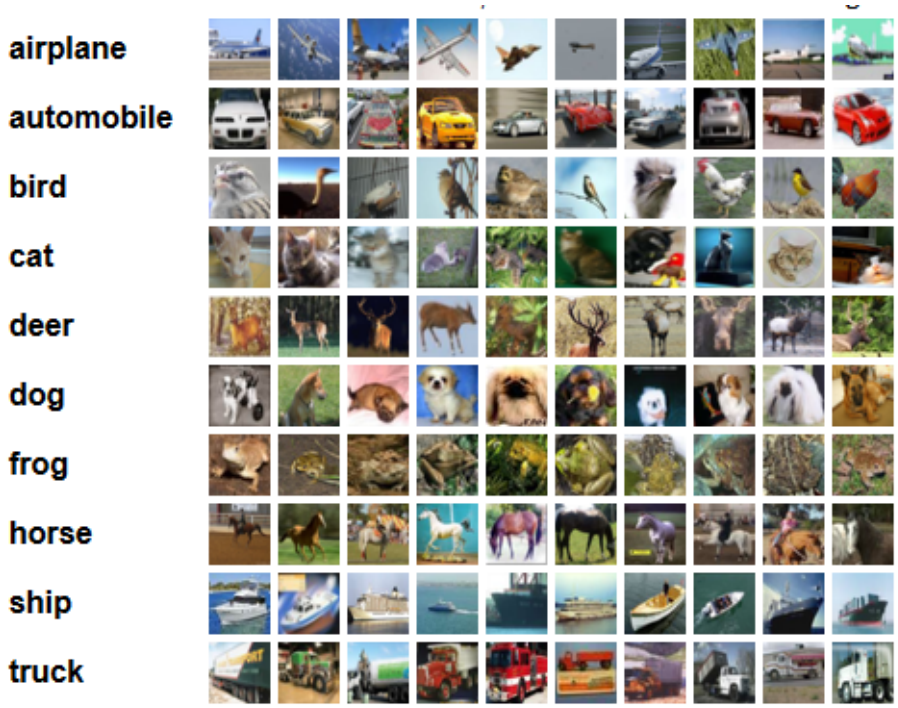


Figure 3.1: Example of CIFAR-10 images taken from CIFAR website

3.2.2 CIFAR-10

This dataset was created using images harvested from the internet. It consists of 60,000 32x32 images organised into 10 classes hence its name. Each class is 6,000 images big and in addition contains 1,000 testing images per class for a total of 10,000 test images.

This figure shows some examples of the images that are available in the CIFAR-10 dataset in their respective classes. You will note that 'Truck' and 'Automobile' are both classes of image. However the dataset has been designed such that there is no intersection between classes so a 'Truck' cannot be an automobile and vice-versa.

3.2.3 CIFAR-100

The CIFAR-100 dataset is very similar to CIFAR-10. It contains the same number of images except we now have 100 classes of 600 training images and 20 superclasses for those 100 classes. Each class also includes 100 test images.

Both CIFAR-10 and 100 are created by the same group of people and are based on the same mined internet data.

Due to the lack of training examples and the significantly larger set of classes available, it is a much harder dataset to perform well in with top models getting 75.73 percent accuracy versus 96.53 percent for top performing CIFAR-10 results.

3.2.4 The Verdict

We ended up opting to focus our efforts on CIFAR-10. The reasoning was quite brief. MNIST is too simple and we tackled it early on for test runs. Results were reasonably high.

What we wished to present for our ending point in our project was a classifier which we wanted to use to classify internet images so MNIST wouldn't do for that.

CIFAR-100 on the other hand had features we were interested in trying to work with like the hierarchy which was interesting on the next project we want to do with this model but is quite difficult to get good results with and would push this projects beyond the scope of what can be done in MMP.

CIFAR-10 was a good compromise between the two so it was a good initial goal allowing us to scale to CIFAR-100 if we were satisfied with our results.

3.3 Choice of Framework

To start we need to decide how we are going to implement the neural networks. This involved several steps, of course one would be tempted to write their own implementation for an ANN, however that might be a major project on its own so a better solution is required.

We would like to minimise implementation effort in parts that do not improve the final project as much as possible so a library of some sort is necessary.

For our framework we wanted something that is both performant but also easy to use. We also want support for the latest ANN features while it needs to work on our workstation.

One of the greatest developments of the past 8 years in ANNs and machine learning in general is the advent and wide availability of parallel computing. Specifically GPU compute solutions like nVidia's CUDA technology have been invaluable in making Deep Neural Network research viable offering orders of magnitude better training times. Our frameword must run on a GPU.

We would like a library that makes testing an idea quickly, two modern candidates emerged at that point Google's new Tensor Flow software was one of them and was our favourite for the early parts of the project. Keras was another. Both use Python to define models and a backend implementation to do processing

We were not looking for specific features in this part of our project, what we were looking for however was a framework that is used in current research and is geared towards experimental CNNs both Keras and Tensor Flow support similar advanced CNN features which makes them fairly equivalent as a choice. They were both alpha software when the project started but so are most libraries in the field as CNNs are fairly new. We initially settled on Tensor Flow.

Most of the frameworks for this task are designed to work on Linux primarily. Tensor Flow works on Linux and OSX. This was a problem when we were running Windows on our main machine so the initial solution was to run Tensor Flow on a Virtual Machine with Linux Mint. This initially appeared to work well but we quickly discovered issues with this approach. To get good training performance CUDA is required. However CUDA requires a direct connection to the graphics adapter.

A solution we considered was to use Intel's VT-d and pass the PCI-E device of the GPU

directly to the VM using the integrated Intel GPU in Windows instead allowing us to use it under a VM but this proved to be too complicated to set up and any other solution was equally complex. We had to abandon Tensor Flow. Keras, supported Windows as well so we ended up switching our efforts to that. This was relatively early on in the project's lifecycle so we could afford to make the change.

3.4 The Stack

3.4.1 GPU Drivers and CUDA

Because of the task of training and running through a neural network is an inherently parallel task we use Nvidia's CUDA technology in order to run computations on the GPU which produces at least an order of magnitude in performance gains over a modern traditional CPU architecture.

Nvidia's CUDA API has emerged as the dominant solution for scientific and industrial compute applications after competing open APIs like OpenCL failed to gain any traction. If you want good ANN performance CUDA is necessary.

3.4.2 CuDNN

CuDNN is an nVidia library written in C++ which provides performance optimisations for several ANN computations to run on CUDA enabled GPUs. This library isn't necessary, but it provided us with a significant performance boost so ended up being invaluable in our experiments.

3.4.3 Backend Theano

To execute computations we are using a library called Theano. Keras interacts with Theano and does weight computations on either the CPU or GPU giving a layer of compatibility and portability to our software as well as provides performance which is essential for training larger models. It uses CuDNN if it's installed to further increase performance on some nVidia GPUs.

3.4.4 Keras

Keras is a framework for creating and testing ANNs. This is the level our code directly interacts with. It provides us with implementations of ANN layers we use to build our models as well as data loaders analytics tools and more.

It can use either Theano or Tensor Flow as a backend to run its computations on the GPU or CPU. Theano then compiles the model in C++ which the CUDA SDK is compatible with.

3.4.5 Our Application

Inside our application we define models and hyper parameters which we are using to test. Our application trains a model and stops when a local minimum is identified for a set span of epochs and

then saves the best model ie the local minimum. The model is then evaluated and a classification report is output.

There is methods also for loading models and to use a loaded model to classify an image input. These interfaces can be used to allow an external application to use our system.

3.5 The Components of our Application

Our code is organised in two major structures:

3.5.1 Utilities

This file contains implementations of some image processing algorithms as well as other utility functionality our program needs these are 'features' we use which aren't part of the experiment logic but enable certain experiments to be conducted.

3.5.1.1 Colour Conversion

A useful utility is the ability to convert training example between RGB colour and grayscale. We achieve this by implementing a number of conversion methods to try and extract the luminance of an RGB image. These methods range from averaging the intensity of the RGB channels at their simplest and least effective to weighted averages using the NTSC/W3C and Rec.702 colour weights for each RGB channel.

3.5.1.2 Scaling and Cropping

For this we use a library called Python Imaging Library. Because our neural networks expect inputs to have certain dimensions we must first ensure that the pictures match. Aspect ration must first be matched by cropping trying to get the most prominent object in the cropped frame, then we scale to the desired size using PIL's scaling routines to match our ANN inputs.

3.5.2 Main Program

This comes in two separate forms, one is using a simple Multi-Layer Perceptron model and was used early in the project to obtain baseline results, the other is based on a Convolutional Neural Network model and was used in later development. The former was ported over to the framework used in the later for the final results.

For all intents and purposes information applies to both unless otherwise noted.

3.6 Challenges and Solutions

3.6.1 Data loading and pre-processing

The first component of our training framework is loading of data. To do this we are taking advantage of built in Keras data loaders. For CIFAR-10/100 and MNIST these come packaged in and take care of downloading the data as well as loading it if it's not already provided. Tensor Flow and other frameworks also provide this.

After data is loaded we process it such that we extract information from it to determine the shape of the inputs and convert RGB values to a float32 value divided by 255. This gives us our ANN inputs. This is also where we do any splitting and colour conversion.

3.6.1.1 Greyscale Luminance

One of the main advantages and disadvantages of this process is it allows us to reduce the number of inputs into our ANN by a factor of 3. This in turn reduces training time by at least 50 percent which is a good advantage and greatly simplifies our network's structure. At the same time we reduce image detail which can have a negative impact in classification performance.

Because of the tradeoff we use both in our experiments, the faster training and simple models is appealing when trying a new hyper parameter for the first time and want to see quick results. Whereas higher performance is for when we have optimised all other aspects of the network.

We implemented our own colour conversion mechanisms that apply to a single image or sets of images. They are based on documentation for the NTSC and Rec.702 colourspaces as published in their respective standards. Exact references and discussion of the different sources for these are included in appendix 3.

3.6.2 Regularisation

As we explained in the previous chapter overfitting is a problem for ML and ANNs in particular, in a situation where we wish to achieve the best generalised performance. Training algorithms can only extract patterns from the data which they are given, so it follows that the more we train on data overfitting eventually takes place, i.e our classifier becomes more tailored to the training data at the expense of general performance.

Keras implements many ways to do this but what we are interested in is which ones we should use, in what variation where and how. How do they impact our performance metrics?

3.6.2.1 Cross-Validation Split

One of the most important problems to solve when combating overfitting is to be able to determine when it is happening so we may act on it.

What Keras enables us to do is to provide a set for validation which we can test against after each training epoch while we monitor our loss to train on our training data. Keras also has a feature to split a portion of the nth portion of the data for this purpose.

Our implementation is to use Keras's validation split for normal data and our own implementation for augmented data. This is because Keras until recently did not provide this feature for its data generator. In CIFAR-10 and 100 data is randomly distributed so we can just split the data at a point from the end and use that for this purpose.

What we do instead is we first shuffle in parallel the training data and labels then split the nth portion at the end. This approach would have been more useful if we were averaging runs. As it stands it means we have unnecessary variations in our data but our results were not visibly affected by this, we could have re-designed this omitting the shuffling or storing the seed for repeatability, but ideally we'd use Kera's better implementation instead now that it's available.

3.6.2.2 Early Stopping and Model Checkpoint

Once we can monitor validation loss and can detect when a model might be overfitting we need something to act upon this. These methods are what we use.

Keras has a callback interface that allows us to call code in certain instances during training to do things. Early Stopping allows us to, with a patience parameter stop training when a model's validation results stop improving for a certain number of epochs. We use 50 epochs in most of our experiments. Because of how training works we want patience to be higher to avoid local minima since what we are looking for is actual local minimum validation loss.

Model checkpoint on the other hand solves another related issue. When validation results stop improving we stop after a few epochs but actually our best result might be several epochs old by the time we stop which means our model will already show signs of overfitting by the time it stops. We can avoid this issue by saving a copy of our most general model and loading it after our training has stopped and use that for validation.

3.6.2.3 Data Augmentation

Keras provides a capability to pass training data through a data generator before they are fed to the neural network for training. The idea is you can set filters that get applied randomly on each image such that every time you encounter an image in training it is slightly different in appearance.

The way this works is that by changing the actual pixel data in the inputs just enough so that the image features are intact you can mitigate the neural network overfitting to the data somewhat.

What we use is random rotation, which rotates the the image around its centrer by a random value of a given range, e.g. -1-1 degrees. Horizontal flip, which flips the image horizontally with a 50 percent likelyhood, and shifting the image by 2 pixels in either direction thus re-positioning it.

The filters work cyclically such that for n images n+1 image will be image 1 but each time the parameters of the filters will vary.

Another later consideration which came from a thought when we were examining data augmentation was to apply random noise to the inputs in hopes of reducing overfitting using the same principle. Keras does support the feature as a normalisation layer but at that point we already had data augmentation implemented and we found out that this kind of augmentation would be ill suited to the very small images we have in our network.

3.6.2.4 Dropout

We are using Keras implementation of dropout to reduce overfitting.

3.6.3 Obtaining and visualising results

For our experiments to be worth their salt we need to be able evaluate our models and visualise our results. For this we turn to SciKit Learn, which is an excellent library for machine learning statistics and visualisations. It helps us generate classification reports and confusion matrices which is the basis of our result reporting.

For evaluating the effects of our augmentations and our training and overfitting characteristics we are plotting loss and validation loss over time. This was a late addition to our framework so sadly we don't have much data for this.

We also initially intended to have a Google Inception-like visualisation of the learned features but we run out of time before implementing this.

3.6.4 Saving Models

Once a model is trained we save all our metrics with a label together with a serialised model and weights. This allows us to regenerate most reports as needed but also to use a trained model for whatever purposes we have.

3.7 Our Basic Model

We are basing our models on a standard Deep CNN architecture. This is very similar in appearance to something like figure 2.2

We are varying certain aspects of our model such that we can test the impact of changes in architectures and other hyper parameters in classification performance.

Chapter 4

Results and Conclusions

In this section we will describe our results and any conclusions we may have drawn from them as well as decisions that came from these results.

4.1 Convolutional Neural Networks: Results

All the neural network here are conv nx3x3 - conv nx3x3 - max pool 2x2 no strides - conv nx3x3 - conv nx3x3 - max pool 2x2 no strides - Dense 256 - Dense 128 - Dense Softmax 10 with Relu activations in the rest of the layers. Where this is not true we will note as such.

We are also using Stochastic Gradient Descent as an optimiser with Nesterov decay of 1e-6, a learning rate of 0.01, a momentum of 0.90 and categorical crossentropy as the loss function. A validation loss is calculated and is used for early stopping with a patience of 50 epochs, the set which is used is a 15 percent slice of the end of the data which is first shuffled to ensure validation data is uniform in class composition.

4.1.1 Number of Filters

For this we are evaluating the following models:

- 8-8-14-14 with 3x3 filters
- 24-24-32-32 with 3x3 filters
- 32-32-64-64 with 3x3 filters
- 48-48-96-96 with 3x3 filters
- 128-128-256-256 with 3x3 filters

For the results we are presenting here we are using data augmentation in all cases to reduce overfitting as well as early stopping to get the best model possible from our training. We have results for both colour and black and white for this evaluation.

We will only be presenting the colour results in this part as we will be evaluating the impact of colour in a later part.

4.1.1.1 Effects on Training Speed

As we increase the amount of filters we see our training speed declining as the number of neurons increases, this makes sense as you increase the number of computations that need to be done each epoch somewhat.

We ommit actual data here we were unable to control running parameters exactly due to these tests having to be run over several nights at different system workloads which would affect the results but our observations showed that we can get a range of 12s per epoch on our smallest model to 57s per epoch on our most complicated model

4.1.1.2 Effects on Performance

We begin this investigation assuming we will get better results with more.

As we can see from the figure 4.1, a significant improvement between out smallest model and out largest one. However it appears we hit diminishing returns after a certain number of filters as we presumably hit the limit of the features we can draw from our small 32x32 data.

From the qualification report in fig. 4.2 we also see an interesting pattern we hinted on earlier in this document. Dogs and Cats are the worst performing classes with confusing happening mostly between them but dogs get confused less with cats than the opposite.

This becomes clearer when we look at the classification reports as seen in figure 4.2

As you might notice from the figures above we stop seeing overall improvements at 48-48-96-96 with small improvement in some classes, but the extra cost of the larger networks makes it a good compromise for further experiments.

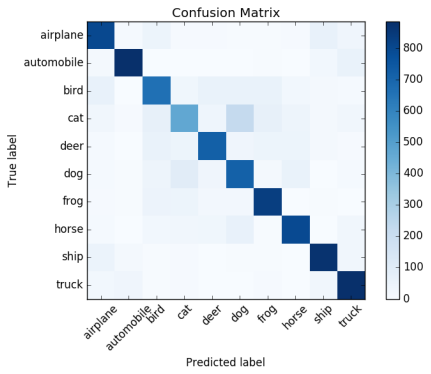
4.1.1.3 Effects on Size of Weights

This investigation is much simpler. We look at the sizes of the weights we store for each network and try to correlate it the number of calculations and network complexity.

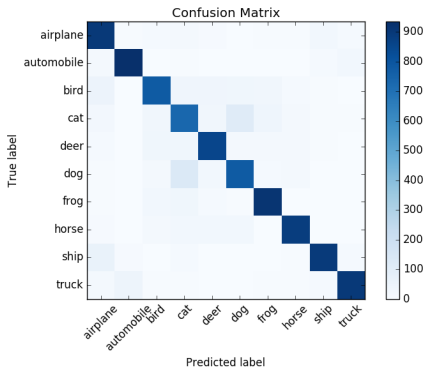
In figure 4.3 we can see how size increases with more features. This makes sense since our feature maps become larger, while we can also see the increase is not completely linear with the number of neurons. This is because in CNNs weights are shared.

4.1.1.4 Conclusion

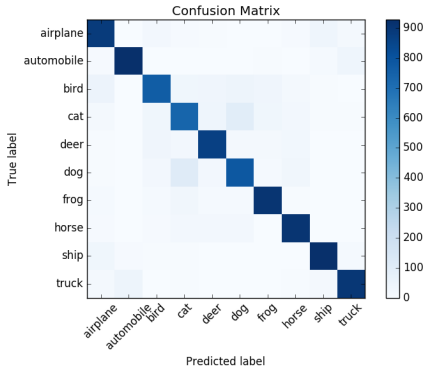
As we increase the number of our filters, we see an increase in classification performance. This comes at the expense of the simplicity of our model which impacts the speed of training, the size of the model and works as a trade-off with our classification performance. Because of the huge cost of models larger than the 48 and the small gains we will investigate using 48 filters in situations where we are testing other hyper parameters.



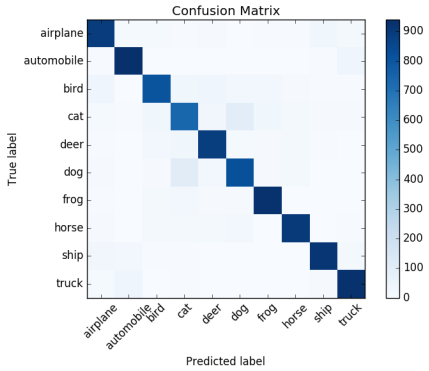
(a) 8-8-14-14



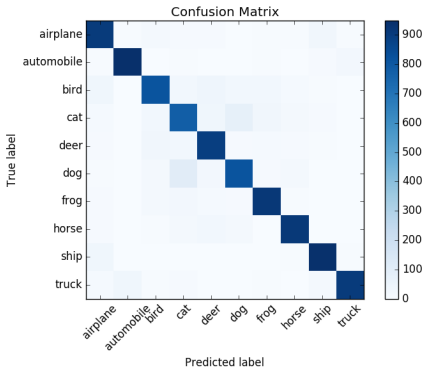
(b) 24-24-32-32



(c) 32-32-64-64



(d) 48-48-96-96



(e) 128-128-256-256

Figure 4.1: Confusion Matrices varied by filter number

4.1.2 Size of Filters

For this series of experiments we will report on the results of varying our 48-48-96-96 and 8-8-14-14 with a set of filter sizes:

- 3x3
- 4x4

8-8-14-14:

	precision	recall	f1-score	support
0	0.78	0.72	0.75	1000
1	0.84	0.90	0.87	1000
2	0.67	0.55	0.61	1000
3	0.55	0.45	0.50	1000
4	0.64	0.72	0.68	1000
5	0.63	0.65	0.64	1000
6	0.73	0.83	0.77	1000
7	0.74	0.81	0.77	1000
8	0.80	0.84	0.82	1000
9	0.86	0.82	0.84	1000
avg / total	0.73	0.73	0.73	10000

(a) 8-8-14-14

24-24-48-48:

	precision	recall	f1-score	support
0	0.82	0.90	0.86	1000
1	0.93	0.94	0.93	1000
2	0.84	0.78	0.81	1000
3	0.71	0.74	0.72	1000
4	0.85	0.86	0.86	1000
5	0.79	0.78	0.79	1000
6	0.88	0.91	0.90	1000
7	0.91	0.89	0.90	1000
8	0.93	0.90	0.91	1000
9	0.94	0.90	0.92	1000
avg / total	0.86	0.86	0.86	10000

(b) 24-24-32-32

32-32-64-64:

	precision	recall	f1-score	support
0	0.84	0.89	0.86	1000
1	0.94	0.93	0.93	1000
2	0.83	0.77	0.80	1000
3	0.74	0.74	0.74	1000
4	0.85	0.87	0.86	1000
5	0.80	0.79	0.79	1000
6	0.89	0.91	0.90	1000
7	0.89	0.91	0.90	1000
8	0.91	0.93	0.92	1000
9	0.92	0.90	0.91	1000
avg / total	0.86	0.86	0.86	10000

(c) 32-32-64-64

48-48-96-96:

	precision	recall	f1-score	support
0	0.87	0.89	0.88	1000
1	0.92	0.94	0.93	1000
2	0.87	0.81	0.84	1000
3	0.76	0.74	0.75	1000
4	0.86	0.89	0.88	1000
5	0.83	0.82	0.83	1000
6	0.91	0.93	0.92	1000
7	0.92	0.90	0.91	1000
8	0.92	0.92	0.92	1000
9	0.90	0.93	0.92	1000
avg / total	0.88	0.88	0.88	10000

(d) 48-48-96-96

128-128-256-256:

	precision	recall	f1-score	support
0	0.87	0.91	0.89	1000
1	0.95	0.95	0.95	1000
2	0.87	0.82	0.84	1000
3	0.77	0.78	0.78	1000
4	0.85	0.89	0.87	1000
5	0.85	0.82	0.83	1000
6	0.90	0.92	0.91	1000
7	0.93	0.92	0.92	1000
8	0.92	0.94	0.93	1000
9	0.96	0.91	0.93	1000
avg / total	0.89	0.89	0.88	10000

(e) 128-128-256-256

Figure 4.2: Classification Reports for experiments

- 5x5

As before we are keeping all other hyper parameters the same.

4.1.2.1 Effects on Training Speed

For this we found training speed was not really affected by using larger filters in fact it was faster in some cases. We originally run this study on most of our studies but for brevity we will only look at the two extremes combinations.

4.1.2.2 Effects on Performance

We expected to see better performance with larger filters that were less numerous on account of potentially capturing larger features. Instead what we saw in fig. 4.4

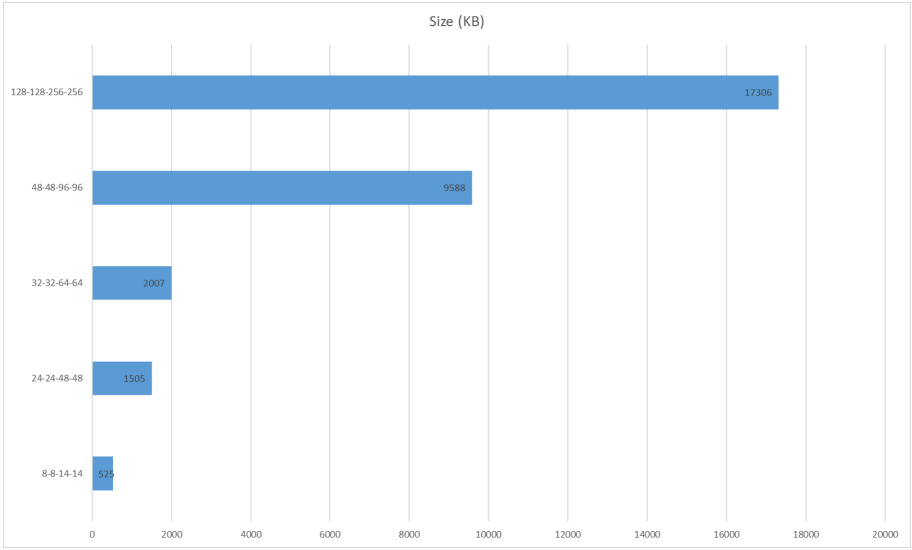


Figure 4.3: Bar Chart of Size of Stored Weights

We can see from the confusion matrices above that larger filters affect results negatively. The difference is so significant it can clearly be seen on the confusion matrices for smaller numbers of filters which is the opposite than what we initially expected.

The other pattern we observe is filter numbers still matter with larger filters we see an improvement in the 48 model. However they never match the smaller filters though the results do converge somewhat.

We believe this might be worse due to the nature of our training images. Remember the images are quite small at 3x3. This also means that any feature in those images is also going to be very small. As a result many small filters seem to be the way to go here.

Due to how significant the difference is we will not be adding the classification report but we note a range of accuracy of 0.60 to 82 on the smaller network.

4.1.2.3 Effects on Size of Weights

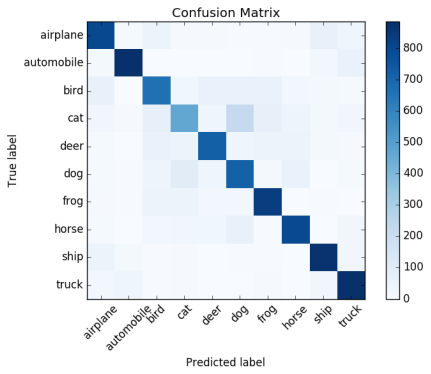
4.1.2.4 Conclusion

Larger filters are not really advisable in this dataset as they give no real practical benefits over 3x3 filters in performance, and training speed. Size of weights does not mitigate the loss in classification performance.

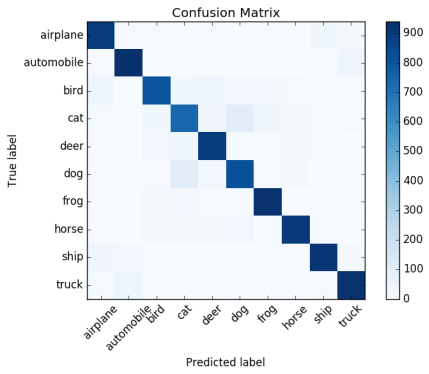
4.1.3 Augmentation

For this section we will only use the 48 network again. As a consistent good performer, and having seen the performance of 3x3 networks with 2 layers between max pooling in this dataset we use it as our reference.

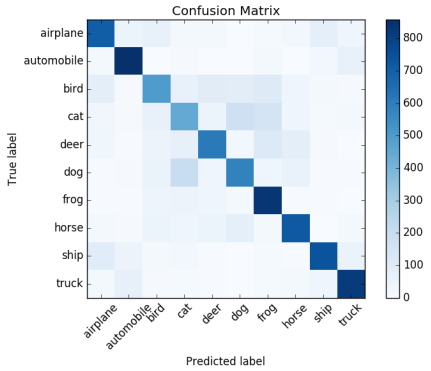
For our augmentation we decided to use a rotation range of 0.1, a horizontal and vertical shift



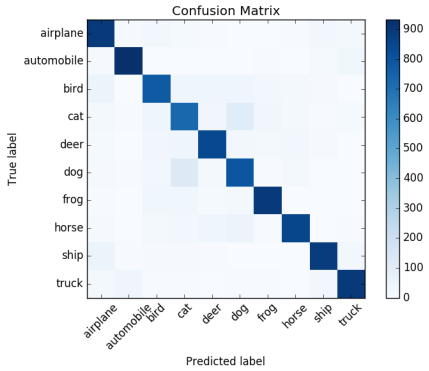
(a) 8-8-14-14 3x3



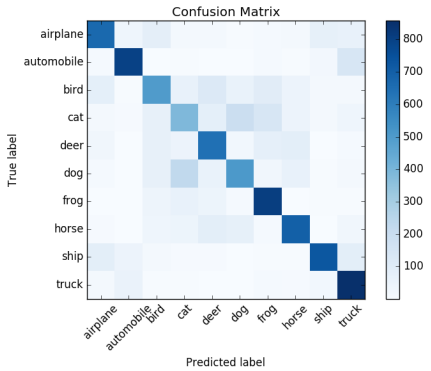
(b) 48-48-96-96 3x3



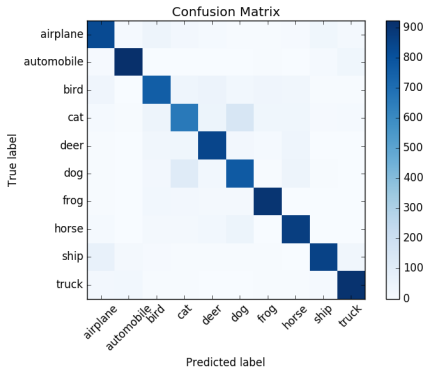
(a) 8-8-14-14 4x4



(b) 48-48-96-96 4x4



(a) 8-8-14-14 5x5



(b) 48-48-96-96 5x5

Figure 4.4: Confusion Matrices varied by filter size

range of 0.8 and a random horizontal flip here. From the intelligence we have in our data and smaller experiments these should work well.

4.1.3.1 Effects on Training Speed

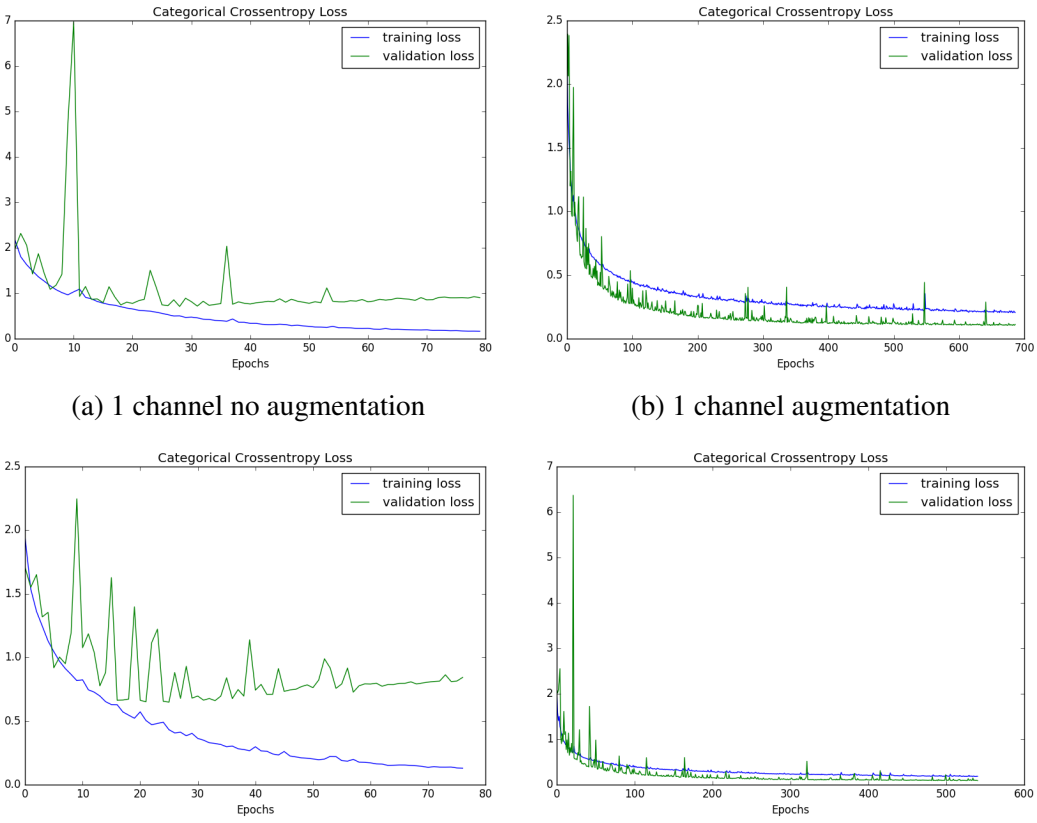
Here we saw very significant drops in speed. This is largely because augmentations need to run on each batch and run on a single CPU thread. This means that we are bottlenecked by the CPU as the

GPU idles waiting for images to train on. One way we observe this is that models are restricted to a minimum of 12s per epoch with it on with no decrease in this time despite trying out parameters that helped in the past.

This could be alleviated if Keras used a parallel image augmentation mechanism since this problem can be solved in a divide a conquer algorithm. We lose about 50-80 percent per epoch to this, impact also depends on the number of channels. Because of this we'll be looking at the benefits both in colour and greyscale.

4.1.3.2 Effects on Overfitting

We would like to see how quickly our model overfits. If our Augmentation works we expect to see a significant reduction in overfitting that should in theory increase our classification performance



(a) 1 channel no augmentation

(b) 1 channel augmentation

(c) 3 channels no augmentation

(d) 3 channel augmentation

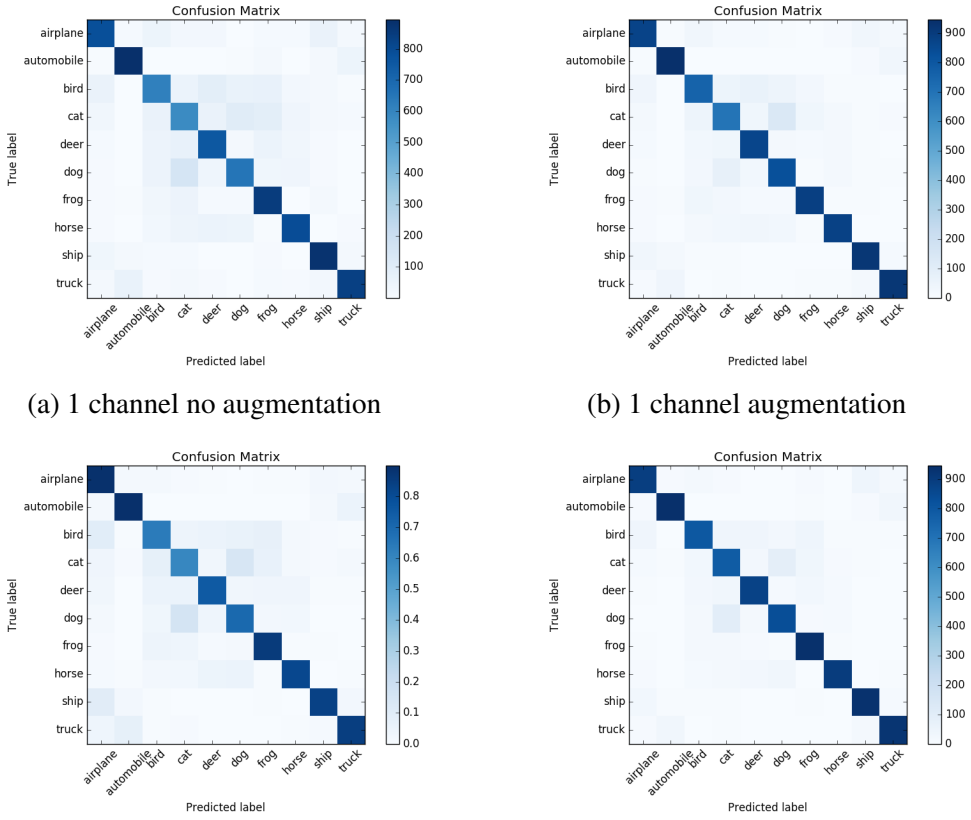
Figure 4.5: Plot of Training History in Terms of Loss

As we can see in the figure, we are seeing a surprisingly significant improvement in our cross-validation results. With a patience of 50, we end up terminating at close to 80 epochs without augmentation. This should reflect very negatively in our generalised classification performance.

An observant reader might also notice local minima for validation loss that is consistent with what we'd expect to see and the main reason we use Early Stopping.

4.1.3.3 Effects on Performance

Following our earlier results we are eager to see whether there is any impact and if so what is it with not using augmentation in terms of results. We've seen it leads to very fast overfitting but does this translate to results?



(a) 1 channel no augmentation

(b) 1 channel augmentation

(c) 3 channels no augmentation

(d) 3 channel augmentation

Figure 4.6: Confusion Matrices for Augmentation Experiment

We can see just from the confusion matrix that performance takes a big hit without our augmentation. Indeed looking at our f1 score in the reports we can see 0.76 and 0.86 between the two options in greyscale. A very significant difference.

Looking very briefly at older logs from past experiments we also notice a pattern that larger networks tend to benefit more from augmentation. To corroborate our results we re-run the experiment and produced 4.7

4.1.4 Conclusions

Data augmentation helps significantly to reduce overfitting and improve performance in our models. The performance cost is very significant with the current implementation of it but results benefit enough that it would be foolish not to use it.

If also appears that the larger the network the more important good augmentation becomes.

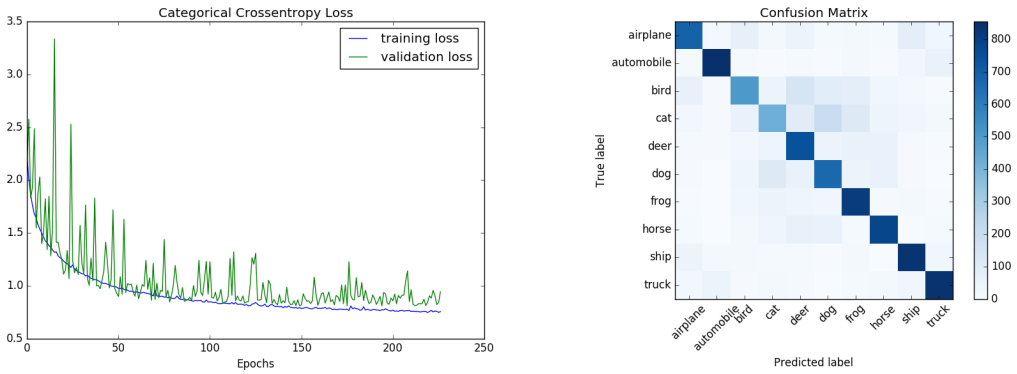


Figure 4.7: Small model with 8-8-14-14 3x3 filters no augmentation

This makes some sense as with more features extracted, our network would be increasingly likely to overfit so avoiding this is important.

4.1.5 Number of Colour Channels

4.1.5.1 Effects on Training Speed

Going from our augmentation results we are eager to observer how colour affects the training speed. We use the 48 as usual as our reference. Conversion takes some extra time especially on our Mode 4 conversion method. What we see however is a big boost in speed. This makes sense since we reduce inputs to a third which means there is much less processing to do.

The effect is more pronounced when augmentation is used where the performance gap widens. Presumably this is because there is much less data to convert which reduces CPU cost for the augmentation code.

4.1.5.2 Effects on Performance

Here we see a small reduction in classification performance by switching to greyscale. It's usually of the order of 2-3 percent. A noticeable decline but not to the extent to justify the extra performance hit to run with 4 channels and augmentation. We can see from 4.8

Ultimately for fast experiments it should not matter as long as it is noted and then once parameters are optimised re-run with colour to observe the better result.

4.1.5.3 Effects on Size of Weights

Minimal increases in the size of the weights was observed here. This is because with a CNN weights are shared by neurons, for the colour that's especially handy as it makes a 3x increase in inputs have a very negligible impact in size.

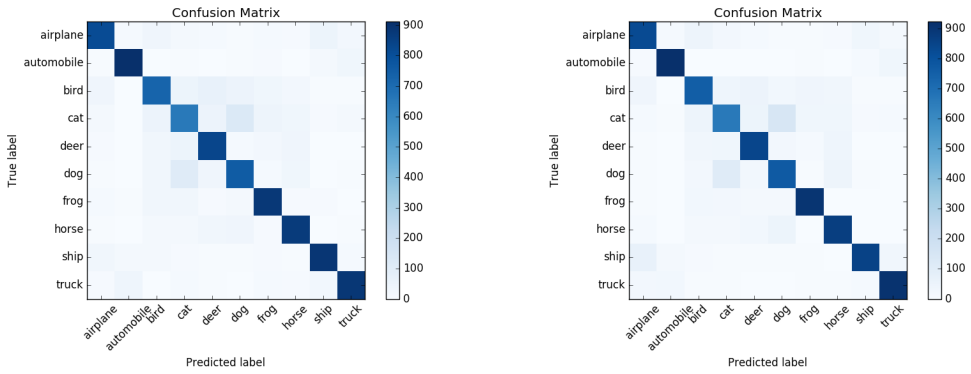
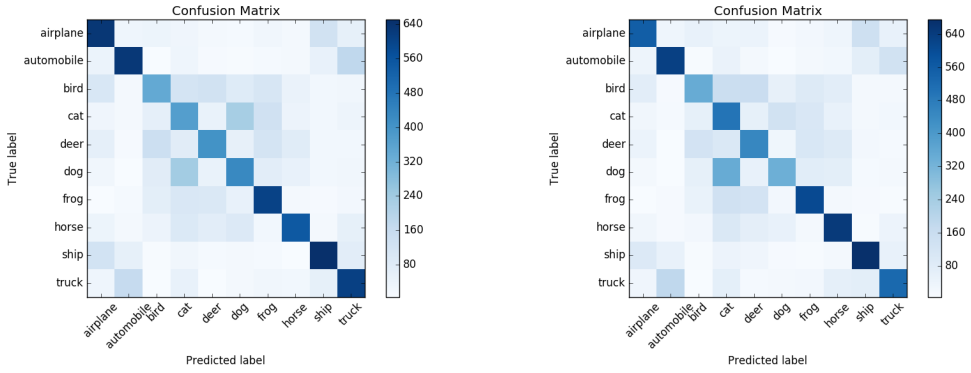


Figure 4.8: Colour to grey confusion matrix comparison

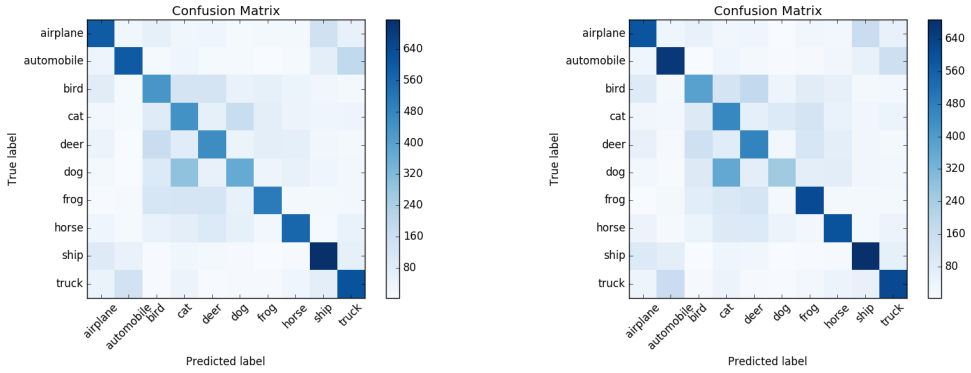
4.1.6 Multi Layer Perceptron: Figures in Matrices

As an early test we obtained a number of results from MLPs. Here we have collated some of the most interesting ones showing the effects of depth of MLPs in terms of training and classification performance



(a) 256-128

(b) 256-256-128-128



(c) 512-512-256-256-128-128

(d) 1024-1024-512-512-256-256-128-128

Figure 4.9: Confusion Matrices varied neurons

we can see this illustrated in figure 4.9 how deeper MLPs with no augmentation improve

slighly but ultimately overfit too quickly this can be seen more clearly in 4.10 We couldn't get augmentation working when we ported MLP's back to the new test suite so we could not test augmentation with MLPs.

We could not exceed 0.54 f1 score with this solution.

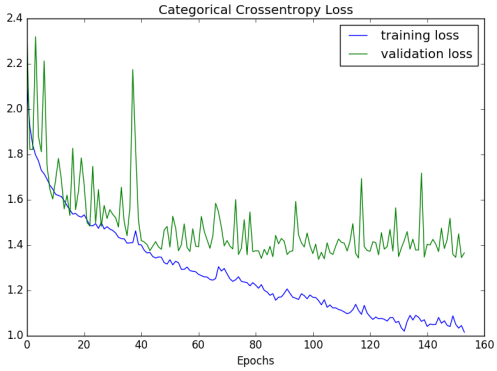


Figure 4.10: 512-512-256-256-12-128 MLP History

4.2 Miscellaneous Discussion

4.2.1 Accounts of Early Experiments

Here we will discuss the earliest parts of our experimentation. These were done before the experimental infrastructure was complete so we don't have complete figures recorded for them. These accounts are based on logs and diary entries taken at the time. Rapid prototyping was taking place at the time so results might not be consistent. As a result this section is included in a pure discussion format and is aimed to offer some insight to early decisions taken in this project.

4.2.1.1 Early Experiments

Early on we experimented with decay, momentum, activation functions and learning rate. We ended up finding that defaults or near defaults for these values worked well for our training solution so we left them as is for the final results in an effort to reduce the hyperparameters we reported on.

Towards the middle of our studies we implemented the greyscale mode. That took a lot of tweaking to get right and we noticed differences between the different modes however at the time we lacked the correct diagnostics to record the data properly.

4.2.2 Conclusion

From our experiments we learned a number of things about CNN design for this set. First lesson was that larger models can get higher results but suffer from diminishing returns and overfitting.

A successful model for this dataset needs a good augmentation strategy, more aggressive augmentation works to an effect but it is important to try to maintain as many features as possible to maintain performance. Features are small so one needs to moreso careful.

Colour gives a small bump but experimenting in greyscale can give significant speed improvements which makes optimisation go much quicker since we can get results faster. Ideally we can only switch to colour for final results.

Larger filters are generally not very good for this dataset. Features are quite fine so 3x3 seems to work quite well from all our experiments and benefits of other filters seem lacklustre from our experiences.

Deeper networks can do well in this dataset and deal better with more aggressive augmentation but the size of the CNN seems to be more important. Very deep networks are hard to train with random initialisation so require different strategies to take advantage of. We did not have time to properly test implications for this.

From looking at these results we can also conclude that for some problems, neural networks might be reasonable ready to be sold, with accuracies exceeding 88 percent for a small student project.

Chapter 5

Critical Evaluation

5.1 Identification of requirements

We believe we made a good effort at identifying requirements for this project. All the issues we identified at the start was addressed and the project was brought to a conclusion. However we feel that perhaps we may have misjudged the scope of the project.

We were originally hoping to have the ANN part to only be 90 percent of the project with the rest being dedicated to creating practical demo applications with it. In the end 100 percent ended up being the CNN analysis

We got some of the hooks for third part interaction which was out initial goal so at the very least we were partially successful here.

Ultimately the project was designed to be decomposable to different sub projects that I feel worked well enough in modularizing requirements and each stage had a correct set of requirements. We failed in terms of planning and some features were implemented later than we needed them. The overhead of running tests meant that some things needed to be done early

5.2 Design

Design was evolutionary so it was more about the design of the processes. I feel the process worked well when it was adhered to but there was a feeling of lack of direction which contributed to keep the project somewhat vague until every aspect was worked out.

It is a difficult topic, which made sizing quite unreliable and literature is fairly scarce and directed at researchers.

5.3 Choice of Tools and Language

We lamented long and hard over the choice of framework. We'd have originally preferred to use Tensor Flow but Keras worked very well. We intended to use Python from the start so it working out i was a good surprise.

Ultimately I don't feel there was anything that could be improved given external factors and the timing of the project. The landscape is much different today than it was when the project was started which is common with alpha software.

5.4 How Well were the needs met

The software provided all the features we wanted from our investigation in the end. There were small things that could be added but ultimately I feel we implemented just enough comfort for the final scope of the project.

Ultimately the original scope was larger than what was expected from this project given the time and complexity of the subject matter. There is enough material to continue development in new areas. Visualisation was a big disappointment since we wished to include it in the report initially but plans had to be changed.

5.5 Regrets

Some of the more ambitious plans relied on some circumstances coming into play that did not, we suffered a major hardware failure near the start of the project that delayed progress immensely, after this restoration ate into all our contingency time and some of the equipment we wished to acquire was also delayed.

We ultimately should have asked for department resources sooner as those helped tide us over duting the mid-project demo for instance. Biggest thing I would have changed was to ask for HPC access from the start since we could have done much more if we had access to these resources.

5.6 The Future of the Project

The future of this project would at least include all original aims. We would like to have the visualisation and interface features as well as the extra statistic tools.

Later on the big project that this was created to prepare for can start. We would like to train this model on a new dataset we have been working on which consists of a reduced set of 500,000 fan drawn images, the ultimate goal is multi-label, hierarchical, associative tag inference and classification for drawn imagery where we can identify characters, origins, and themes as provided in labels.

This technology can then be integrated in our intelligent web crawler to filter incoming images.

Appendices

Appendix A

Third-Party Code and Libraries

For this dissertation I have used the following libraries as stated in the design documentation:

1.1 Keras

Neural network library allowing quick model prototyping. [3] and simplifies some GPU operations which would otherwise have to be implemented.

1.2 SciKit Learn

Visualisation and statistical tools for machine learning. Can be found here

Appendix B

Ethics Submission

2.1 Submission Number 4111

AU Status
Undergraduate or PG Taught

Your aber.ac.uk email address
thn2@aber.ac.uk

Full Name
Theodoros Nikopoulos Exintaris

Please enter the name of the person responsible for reviewing your assessment.
Rayer Zwiggelaar

Please enter the aber.ac.uk email address of the person responsible for reviewing your application
rrz@aber.ac.uk

Supervisor or Institute Director of Research Department
cs

Module code (Only enter if you have been asked to do so)
CS39440

Proposed Study Title
MMP: Deep Learning for Deep Neural Networks

Proposed Start Date
20-01-2016

Proposed Completion Date
04-05-2016

Are you conducting a quantitative or qualitative research project?
Mixed Methods

Does your research require external ethical approval under the Health Research Authority?
No

Does your research involve animals?
No

Are you completing this form for your own research?
Yes

Does your research involve human participants?
No

Institute
IMPACS

Please provide a brief summary of your project (150 word max)
Investigation of Deep Neural networks for computer vision (classification) problems.

Where appropriate, do you have consent for the publication, reproduction or use of any unpublished material?
Yes

Will appropriate measures be put in place for the secure and confidential storage of data?
Yes

Does the research pose more than minimal and predictable risk to the researcher?
No

Will you be travelling, as a foreign national, in to any areas that the UK Foreign and Commonwealth Office advise against travel to?
No

Please include any further relevant information for this section here:

If you are to be working alone with vulnerable people or children, you may need a DBS (CRB) check. Tick to confirm that you will ensure you comply with this requirement should you identify that you require one.
Yes

Declaration: Please tick to confirm that you have completed this form to the best of your knowledge and that you will inform your department should the proposal significantly change.
Yes

Please include any further relevant information for this section here:

Appendix C

Code Examples

3.1 Colour Conversion

This code enables conversion between RGB channel intensities into greyscale luminance using different conversion methods based on their respective colourspace recommendations in a variety of modes. A starting discussion on the effects of different methods can be found here

```
import numpy as np
from math import sqrt

def convert_set_to_greyscale(cifar_set, method=0, gamma=1.0):
  converted_set = np.empty((cifar_set.shape[0], 1,
     cifar_set.shape[2], cifar_set.shape[3]), 'float32')
  for image_index, image in enumerate(cifar_set):
    for row_index, row in enumerate(image[0]):
      for pixel_index, pixel in enumerate(row):
      grey = 0.0
      if method == 0: # Rec.709 luminance
        grey = (0.2126 * cifar_set[image_index, 0, row_index,
           pixel_index]) + \
        (0.7152 * cifar_set[image_index, 1, row_index, pixel_index])
           + \
        (0.0722 * cifar_set[image_index, 2, row_index, pixel_index])
      elif method == 1: # NTSC/W3C luminance
        grey = (0.299 * cifar_set[image_index, 0, row_index,
           pixel_index]) + \
        (0.587 * cifar_set[image_index, 1, row_index, pixel_index]) +
           \
        (0.114 * cifar_set[image_index, 2, row_index, pixel_index])
      elif method == 2:
        grey = sqrt(((0.299 * cifar_set[image_index, 0, row_index,
           pixel_index]) ** 2) +
        ((0.587 * cifar_set[image_index, 1, row_index, pixel_index])
           ** 2) +
        ((0.114 * cifar_set[image_index, 2, row_index, pixel_index])
           ** 2))
      elif method == 3:
        grey = sqrt(((0.2126 * cifar_set[image_index, 0, row_index,
```

```
            pixel_index]) ** 2) +
          ((0.7152 * cifar_set[image_index, 1, row_index, pixel_index])
            ** 2) +
          ((0.0722 * cifar_set[image_index, 2, row_index, pixel_index])
            ** 2))
        elif method == 4: # Simple mean of RGB
          grey = ((cifar_set[image_index, 0, row_index, pixel_index]) +
          (cifar_set[image_index, 1, row_index, pixel_index]) +
          (cifar_set[image_index, 2, row_index, pixel_index])) / 3
        else:
          print 'Error: This is not a valid conversion mode.\n
            Reverting to colour.'
        return cifar_set.astype('float32') / 255

      converted_set[image_index, 0, row_index, pixel_index] =
        np.float32(grey/255)
    print 'Converted ', len(converted_set), ' images to greyscale.'
    return converted_set
```

Annotated Bibliography

[1] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*, 1st ed. Springer, Oct. 2007. [Online]. Available: http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0387310738 0387310738.

A book about the application of Pattern Recognition and Machine Learning

[2] O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee, "Choosing multiple parameters for support vector machines," *Machine learning*, vol. 46, no. 1-3, pp. 131–159, 2002.

[3] F. Chollet, "Keras," https://github.com/fchollet/keras, 2015, accessed March 2016.

Keras citation as recommended by author.

[4] D. C. Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber, "Deep, big, simple neural nets for handwritten digit recognition," *Neural computation*, vol. 22, no. 12, pp. 3207–3220, 2010.

A useful paper about the effect of multiple layers for MNIST classification.

[5] M. A. Hearst, S. T. Dumais, E. Osman, J. Platt, and B. Scholkopf, "Support vector machines," *Intelligent Systems and their Applications, IEEE*, vol. 13, no. 4, pp. 18–28, 1998.

[6] R. N. Keiron O'Shea, "An introduction to convolutional neural networks," *arXivaij*, vol. 1511.08458, 2015.

Helpful primer into convolutional neural networks

[7] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

A discussion on the Gradient Descent optimisation algorithm.

[8] Y. LeCun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*. IEEE, 2010, pp. 253–256.

The paper that first introduced Convolutional Neural Networks.

[9] T. B. Ludermir, A. Yamazaki, and C. Zanchettin, "An Optimization Methodology for Neural Network Weights and Architectures," *Neural Networks, IEEE Transactions on*, vol. 17, no. 6, pp. 1452–1459, 2006. [Online]. Available: http://dx.doi.org/10.1109/tnn.2006.881047

This document discusses the use of Weights in Deep Neural Network architectures

[10] W. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biology*, vol. 5, no. 4, pp. 115–133, Dec. 1943. [Online]. Available: http://dx.doi.org/10.1007/bf02478259

Seminal work regarding creating artificial neurons which, even though the techniques described in the book have become out of date, still forms the basis of modern ANNs

[11] M. Moreira and E. Fiesler, "Neural networks with adaptive learning rate and momentum terms," *Technique Report 95*, vol. 4, 1995.

Discussion of the use of momentum and learning rate in neural networks.

[12] J. Shotton, T. Sharp, A. Kipman, A. Fitzgibbon, M. Finocchio, A. Blake, M. Cook, and R. Moore, "Real-time human pose recognition in parts from single depth images," *Communications of the ACM*, vol. 56, no. 1, pp. 116–124, 2013.

An paper, describing the use of synthetic data for use in machine learning.

[13] Various, "Cifar," http://www.cs.toronto.edu/~kriz/cifar.html, Feb. 2016, accessed February 2016.

Site for the CIFAR-10 and CIFAR-100 datasets.

[14] ——, "Mnist," http://yann.lecun.com/exdb/mnist/, Feb. 2016, accessed February 2016.

Site for the MNIST dataset.

[15] ——, "Tensor flow," https://www.tensorflow.org/, Feb. 2016, accessed February 2016.

Site for Google's Tensor Flow machine learning framework