Critical Comparison of the Classification Ability of Deep Convolutional Neural Network Frameworks with Support Vector Machine Techniques in the Image Classification Process

Robert Kelly
Technological University Dublin

Follow this and additional works at: https://arrow.tudublin.ie/scschcomdis

Part of the Computer Engineering Commons

Recommended Citation

This work is licensed under a Creative Commons Attribution-Noncommercial-Share Alike 3.0 License
Critical Comparison of the Classification Ability of Deep Convolutional Neural Network Frameworks with Support Vector Machine Techniques in the Image Classification Process

Robert T. Kelly
D14126766

A dissertation submitted in partial fulfilment of the requirements of Dublin Institute of Technology
for the degree of
M.Sc. in Computing (Data Analytics)
25/01/2017
Declaration

I certify that this dissertation which I now submit for examination for the award of MSc in Computing (Data Analytics), is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

This dissertation was prepared according to the regulations for postgraduate study of the Dublin Institute of Technology and has not been submitted in whole or part for an award in any other Institute or University.

The work reported on in this dissertation conforms to the principles and requirements of the Institute’s guidelines for ethics in research.

Student Name: Robert Kelly
Signed: .............................................
Date: 25 January 2017
Abstract

Recently, a number of new image classification models have been developed to diversify the number of options available to prospective machine learning classifiers, such as Deep Learning. This is particularly important in the field of medical image classification as a misdiagnosis could have a severe impact on the patient. However, an assessment on the level to which a deep learning based Convolutional Neural Network can outperform a Support Vector Machine has not been discussed.

In this project, the use of CNN and SVM classifiers is used on a dataset of approx. 55,000 images. This dataset was used to assess the classification potential of each methodology, in terms of training, implementation, and the ability to engineer parameters and features for successful classifications on a very large dataset. The use of CNN approaches is further broken down into the use of different frameworks, in this case Theano and Torch implementations. These are then compared to an SVM classifier by confusion matrix, training time and ease of use to assess which has the higher classification potential.

Here it is seen that the Theano model outperforms the Torch model slightly for this task, by roughly 3% in the accuracy of the confusion matrix. The SVM meanwhile is shown to be very limited in its ability to classify such a large dataset. Furthermore, the SVM is shown to be limited in its ability to recognise the classes corresponding to the different levels of disease severity, achieving a classification accuracy of only 75% for the whole sample.

These results suggest that the application of Deep Learning techniques currently have a very large advantage over SVM approaches in both accuracy and data handling, that to not natively avail of the computational power of Deep Learning.

Key Words: Deep Learning, Neural Networks, Support Vector Machines, Medical Image Classification, Theano, Torch
Acknowledgements

This was a very long work in progress and it is thanks to the perseverance and understanding of a number of people that it has been eventually completed.

First, to John McAuley for enormous support and encouragement throughout the process, without whom I would have been unlikely to finish this work, and who was given the unenviable task of keeping this on track at times when it was completely up in the air.

Second, I would like to thank the staff of the DIT School of Computer Science, particularly Luca Longo, Sarah Jane Delaney and Andrea Curley for boundless support and understanding at a very busy time in my life.

Third, to my colleagues at Deutsche Bank, who were very supportive to the completion of this work at the expense of having to pick up the slack I left on the team while completing this.

Finally, to my family, my partner Shane-Liz Andaloc and my daughter Jasmine Andaloc-Kelly. Having a baby during the completion of this project, was both one of the most stressful and the most rewarding experience of my life. This work would not have been possible without the unbelievable patience and understanding of my partner, and I am endlessly grateful for this, and am finally able to take up my share of the baby changing responsibilities.
# Table of Contents

Declaration .............................................................................................................................. ii
Abstract .................................................................................................................................. iii
Acknowledgements ............................................................................................................... iv
Table of Figures ...................................................................................................................... vii
Table of Tables ...................................................................................................................... viii
1. Introduction ....................................................................................................................... 9
   1.1. Background ................................................................................................................... 9
   1.2. Research Question .................................................................................................... 10
   1.3. Hypothesis and Research Objectives ....................................................................... 10
   1.4. Research Methodologies .......................................................................................... 11
   1.5. Thesis outline .......................................................................................................... 11
2. Literature Review and related work .................................................................................. 13
   2.1 Diabetic Retinopathy Background .......................................................................... 13
   2.2. Machine Learning Classifiers ................................................................................. 14
   2.3. Support Vector Machines ......................................................................................... 15
   2.4. Convolutional Neural Networks ............................................................................. 18
       2.4.1. Convolutional Layers ...................................................................................... 18
       2.4.2. Pooling Layers .............................................................................................. 19
       2.4.3. Fully Connected Layers ................................................................................. 20
   2.5. Model Performance Measurements ......................................................................... 20
   2.6. Deep learning and GPU computing ....................................................................... 23
   2.7. Summary .................................................................................................................. 24
3. Design and Implementation ............................................................................................... 25
   3.1. Data .......................................................................................................................... 25
   3.2. SVM Configuration ................................................................................................. 26
   3.3. CNN Configuration ................................................................................................. 26
   3.4. Summary .................................................................................................................. 27
4. Implementation and Results .............................................................................................. 28
   4.1. Image Pre-Processing .............................................................................................. 28
   4.2. Development of SVM ............................................................................................ 31
   4.3. Development of CNN ............................................................................................ 33
       4.3.1. Theano Implementation ............................................................................... 33
       4.3.2. Torch Implementation ............................................................................... 37
Table of Figures

Figure 1: Project workflow diagram .................................................................................................................. 12
Figure 2: Support Vector Machine Overview .................................................................................................. 17
Figure 3: Convolutional Neural Network Overview .......................................................................................... 20
Figure 4: Machine learning performance measurement ....................................................................................... 22
Figure 5: Source training images .......................................................................................................................... 29
Figure 6: Border Removal ...................................................................................................................................... 29
Figure 7: Collage of Transformed images ............................................................................................................. 30
Figure 8: Theano Network architecture ................................................................................................................ 34
Figure 9: Torch network architecture ................................................................................................................... 37
Figure 10: Montage of pre-processed images and the resulting greyscale images ............................................. 40
Figure 11: Results of the AHE pre-processing stage ............................................................................................. 40
Figure 12: Results of the wavelet transformation step of the feature engineering stage ................................. 41
Figure 13: Results of the application of fuzzy c-means clustering to the sample images .................................. 42
Table of Tables

Table 1: Theano Blend Network Architecture .......................................................... 36
Table 2: Summary of the various models being investigated and the parameter and feature engineering tasks involved with their development. ....................................................................... 44
Table 3: Theano CNN Confusion Matrix ..................................................................... 48
Table 4: Torch CNN Confusion Matrix ......................................................................... 49
Table 5: Training times of both CNN implementations .............................................. 50
1. Introduction

In this section, the background scope for the study will be introduced, the research question will be presented, the hypothesis and research objectives will be stated and the data source will be specified. Finally, an overview of this thesis and its contents will be presented.

1.1. Background

Many people consider dynamic parallelism to be the future of computing as future microprocessor development will likely focus on the addition of cores to the processor system. However, it is difficult to fully integrate all CPU cores as a fully parallel system. The architecture of a graphics processing unit (GPU) allows for parallelisation as these features are necessary to render and perform computationally difficult visualisations (Merrill & Grimshaw, 2011; Owens et al., 2008).

Since its induction, a number of applications have benefited greatly from the application of Deep Learning within the field of data analytics. One such example is the field of medical imaging, which was one of the first “early adopters” of GPU programming. As patient numbers, and hospital waiting times, continue to increase, automating the diagnostic process could free up time for medical researchers to deal with more patients in the same period of time. This would also allow patients to receive a diagnosis at an expedited pace.

While a number of papers have sought to understand the performance gains possible through Deep Learning and GPU computing, information on the time saved carrying out classification of large medical image datasets is scant (Pratx & Xing, 2011; Schmidhuber, 2015)

This study intends to investigate the classification of images using modern Deep Learning based approaches and quantitatively assess to what extent they can outperform more traditional CPU based techniques. While previous works (Bandyopadhyay & Sahni, 2011; Cheng & Wang, 2011; Schmidhuber, 2015; Tarditi, Puri, & Oglesby, 2006) have identified a correlation between the addition of GPU technology to an algorithm, and an increase in performance, it has not been completely assessed to what level there will be an increase in image classification using high resolution images.

Image classification is a well-known quantity, and it is possible to use a number of techniques to implement it. The addition of Deep Learning techniques allows for the implementation of Convoluted Neural Networks (CNN), which have been shown to allow for
increased speed and fidelity in classification procedures in other studies. In contrast to other machine learning techniques, CNN’s do not require an overly large training time, therefore the model production can be expedited. To compare and contrast the performance increases, traditional neural networks and support vector machines will be used for comparison, as they have also been shown to have strong abilities to classify images.

1.2. Research Question

The research question that this study seeks to address is: To what extent can deep learning based neural networks statistically out-perform Support Vector Machines for image classification and under what circumstances could other Support Vector Machines be considered more useful?

1.3. Hypothesis and Research Objectives

**H0:** This is the null hypothesis. The application of Deep Learning will result in no performance increase over CPU approaches and will result in similar training times.

**H1:** The addition of Deep Learning and GPU computing techniques by way of Deep Convoluted Neural Networks will allow for a decrease in misclassification rate over a traditional neural network classification or Support Vector Machine (SVM).

**H2:** Training time will be greatly reduced when using Deep Learning compared to a traditional CPU based approach

This project will involve carrying out secondary research, using quantitative approaches, to empirically classify a best practice method of image classification using deductive reasoning. Data will be generated using the suggested datasets, split between a training a test set. Using these datasets, a comparative study between deep learning and traditional image classification tests will be carried out.
1.4. Research Methodologies

The data for this study is a large medical image dataset, focusing on the instances of diabetic retinopathy in approx. 30,000 training images, along with 55,000 testing images. This dataset was available from EyePACS, which is a diagnostics company specialising in this particular condition. This data was presented in several batches of 8GB files due to the large size, and combined later. Also provided was a master file, with the correct classifications given for each image in the testing image dataset, which is assumed to be 100% correct for this purpose. The models used for testing will consist of several iterations of various machine learning models, an SVM developed in the Python Scikit Learn package and two CNN’s developed in both Python and Lua using the Theano and Torch frameworks.

1.5. Document outline

This thesis will consist of constructing models for both convolutional neural networks and support vector machines. The neural network models will be based on the Theano and Torch frameworks, which are built in python and Lua respectively. These will be critically compared to each other, being used to classify a reduced size dataset to understand their features and strengths and weaknesses, based both upon the experimental results and supported by the relevant literature. The best performing model will be selected to progress to a classification of the entire dataset, as due to time constraints both cannot be run in full. The results of this full classification procedure will be compared to the results of the SVM classification and both methods will be compared under the criteria of:

1. Adaptability of the frameworks
2. Ease of use
3. Training times
4. Confusion matrixes

The experimental process will follow the workflow presented below in Figure 1.1. This will be referenced throughout where appropriate.
Figure 1: Project workflow diagram. This diagram highlights the various stages in experimental design used throughout. Also highlighted are the various decision points present in the work.
2. Literature Review and related work

In this chapter an overview of the medical problem contained in the dataset images, and an introduction to the different levels of classification of this condition. Next, an introduction to machine learning, and the specific machine learning techniques that will be used to address the research question and answer the hypotheses will also be presented, namely Support Vector Machines and Convolutional Neural Networks. This will give a reasonable overview of how the problem will be approached in the project.

2.1 Diabetic Retinopathy Background

Diabetic retinopathy also known as diabetic eye disease, is when damage occurs to the retina due to diabetes. It’s a systemic disease, which affects up to 80 percent of all patients who have had diabetes for 20 years or more. Despite these intimidating statistics, research indicates that at least 90% of these new cases could be reduced if there were proper and vigilant treatment and monitoring of the eyes. The longer a person has diabetes, the higher his or her chances of developing diabetic retinopathy (Priya & Aruna, 2010).

There are five major level of clinical DR severity. Many patients have no clinically observable DR early after DM diagnosis, yet there are known structural and physiologic changes in the retina including slowing of retinal blood flow, increased leukocyte adhesion, thickening of basement membranes, and loss of retinal pericytes. The earliest clinically apparent stage of DR is mild non-proliferative diabetic retinopathy(NPDR) characterized by the development of micro aneurysms. The disease can progress to moderate NPDR where additional DR lesions develop, including venous calibre changes and intra-retinal microvascular abnormalities. The severity and extent of these lesions in increased in severe NPDR, and retinal blood supply becomes increasingly compromised. As a consequence, the non-perfused areas of the retina send signals stimulating new blood vessel growth, leading to proliferative diabetic retinopathy(PDR). The new blood vessels are abnormal, friable, and can bleed easily often causing severe visual loss. Diabetic macular edema (DME) occurs when there is swelling of the retina due to leaking of fluid from blood vessels within the macula, and can occur during any stage of DR.
The progression from no retinopathy to PDR can take 2 decades or more, and this slow rate enables DR to be identified and treated at an early stage. Development and progression of DR is related to duration and control of diabetes. DR in its early form is often asymptomatic, but amenable to treatment. The Diabetic Retinopathy Study and the Early Treatment of Diabetic Retinopathy Study (ETDRS) showed the treatment with laser photocoagulation can more than halve the risk of developing visual loss from PDR (Pratt, Coenen, Broadbent, Harding, & Zheng, 2016; Priya & Aruna, 2010; Yu, Liu, Valdez, Gwinn, & Khoury, 2010).

2.2. Machine Learning Classifiers

One of the central medical image classification issues currently, is that manual recognition is often used, with a doctor, or another in a similarly specialised role, being required to view the image and give an opinion on the disease state of the patient. Automated classification of images would be both more cost efficient and far faster, however an issue with current computerised classification methods is that it’s very difficult to write the program that could recognise a three dimensional object for example. This issue can be compounded when new viewpoints or lightings are added to this.

The machine learning approach however is, instead of writing programs for a particular problem by hand, examples are collected which specify the correct outputs for a particular input. The particular machine learning algorithm being used for the task, then produces a program that can accomplish the task of generating the correct outputs for new input examples. In this way, the program generated by a linear algorithmic process may look completely different than one developed manually by a programmer, it may contain huge amounts of information about a seemingly trivial task, for example how to calculate the weights of a specific piece of the program can be a huge amount of the calculation in the algorithm. The ideal in this case is for the program to function as close as possible on new cases, called the testing set, then it did for the original input, called the training set. Once this training is executed, it should also be possible to retrain the model if the underlying data chances to allow it to adapt to the new requirements. As the level of available computational power has increased drastically, so has the ability to develop complex machine learning tasks simply and more cost effectively than paying someone to develop the program for us.
Machine learning applications have shown themselves very adept at solving some complex problems, likely more adept than a human could possibly be. Some examples of the things that learning algorithms are most adept at revolve around the topic of recognizing patterns, for example objects in real scenes, or the identities or expressions of people's faces, or spoken words. Anomaly detection is another core area in which learning algorithms excel. Anomaly detection includes things such as fraud, so, for example, an unusual sequence of credit card transactions would be considered an anomaly. Similarly, another example of an anomaly would be an unusual pattern of sensor readings in a nuclear power plant. Trusting a supervised learning process to identify and mitigate these issues alone would not be an idea scenario. The ideal scenario would be for an unsupervised process to look at the ones that blow up, and see what caused them to blow up. Then a technician could review this data and attempt to implement changes that would prevent this from occurring.

Within the field of object recognition, there are a number of classifiers that have shown themselves very adept at using a training set as a base to build a very accurate classifier. As mentioned previously, this is particularly encouraging within the field of medical image diagnostics. With the strain placed on the health services in recent years, freeing up the time of specialised professionals via automating classification of images would be very beneficial and may allow for a vastly reduced overhead in patient waiting times. In the next section, a range of different machine learning classifiers, currently used for object recognition, will be discussed.

2.3. Support Vector Machines

Support vector machines are a type of supervised learning technique that are based upon the principals of statistical learning theory. As machine learning needs have increased in areas such as character and handwriting digit and text recognition in the field of image classification, the demand for a low overhead machine learning technique to handle this need has also increased. SVM’s are simple in comparison to other, more programatically difficult classifiers, however as they are non-parametric, they can be used to give a boost to the robustness compared to ANN’s or other non-parametric classifiers(Zhan & Shen, 2005).
The benefit of using an SVM over another classification technique is that it is relatively easy to obtain an acceptable result in minimal time and with very little programming overhead (Priya & Aruna, 2010). Support vector machines are some of the most theoretically superior classification methodologies and give very high levels of accuracy in high dimensionality datasets. An SVM classification tasks involves training and testing data consisting of some sort of data instances. Each of the instances of the training set must contain a target value, and several attributes, also known as Class Labels and features. The goal of this classification task is to produce the optimal hyperplane, which is the best possible model which can predict the value of the target for all the data instances in the testing set, while only been given the attributes.

SVM’s function by use of a kernel function φ, via nonlinear projection of training data in the input space to a feature space of higher, to possibly infinite, dimensions. Then, the SVM detects the linear separating hyperplane which has the maximal margin in this higher dimensional space. This process is what enables the classification procedure in remote sensing datasets, which are usually nonlinearly separable in the input space. In many classification instances, high dimension feature spaces result in overfitting of the input space. Conversely, in SVMs, overfitting is mitigated through the principle of structural risk minimization. The risk of misclassification can be minimised by maximizing the margin between the decision boundary and the data points themselves. In practice however, the minimisation of a cost factor replaces this; the cost function describes both the complexity of the classifier and the degree to which marginal points can be misclassified. The trade-off between these factors is managed through a margin of error parameter which is further tuned through cross validation procedures. An example of a binary SVM is shown in Fig 1.

SVM’s offer many advantages in the field of machine learning. Some examples of this are

1. Uniqueness of the solution, as due to the theory behind SVM’s this is guaranteed to be the global minimum of the optimization problem the classifier corresponds to
2. The solution is certain to have good generalisation properties
3. SVM’s have a solid base, being based on statistical learning theories and also on optimisation theory
4. Common formulation for the class separable and the class non-separable problems as well as for linear and non-linear problems (through kernel trick)

16
Figure 2: Support Vector Machine Overview. The Support Vector Machine seeks to create the optimal Hyperplane that separates all vectors in the image correctly, shown here as H₃.

These properties make SVM’s a very attractive prospect for application to a number of machine learning problems (Mane, Hingane, Matkar, & Shirsat, 2015; Venkatesh & Ramamurthy, 2014.; Yu et al., 2010). SVM’s allow for the construction of various classifiers based on a different choice of dot products, therefore the influence of the functions that can be implemented by specific learning machine can be studied in a unified frame work, making SVM’s very applicable to problems requiring the generating of multiple learning machines (AR, 2011). Since SVM’s are developed from SLT, largely on the structural risk minimisation principal, they are guaranteed to have high generalisation ability. Some studies have suggested that decision rules constructed by the SVM algorithm do not reflect any incapability of the learning machine, but rather regularities of the data (Y. Lin et al., 2010).
2.4. Convolutional Neural Networks

Convolutional Neural Networks (CNN’s) are hierarchical neural networks whose convolutional layers alternate with subsampling layers, reminiscent of simple and complex cells in the primary visual cortex. CNNs vary in how convolutional and subsampling layers are realized and how the nets are trained. The first layer is the image processing layer. This is an optional pre-processing layer of predefined filters that are kept fixed during training. Thus additional information besides the raw input image can be provided to the network, such as edges and gradients. In particular, it can be seen that a contrast-extracting layer helps to improve the recognition rate for certain datasets and implementations (Schmidhuber, 2015).

There are three types of layers in a Convolutional Neural Network:

3. Fully-Connected Layers

2.4.1. Convolutional Layers

The main layers in this approach are the convolutional layers. These layers are parametrized by the size and the number of the maps, kernel sizes, skipping factors, and the connection table. Convolutional layers are comprised of filters and feature maps.

i. Filters

The filters are the equivalent to neurons of the layer. These filters take in weights as an input and output some value. The input size of the filter is a fixed square, known as a patch or a receptive field. If the convolutional layer is an input layer, the format of the input patch will be pixel values. If it is deeper in the network architecture, then the convolutional layer will take its input from a feature map from the previous layer.

ii. Feature Maps

The feature map is the output produced by one filter being applied to the previous layer. A given filter is drawn, by moving a single pixel at a time, across the entire previous layer. Each
position in the layer results in an activation of the neuron, and output is collected in the feature map. If the receptive field is moved one pixel from activation to activation, then the field will overlap with the previous activation by field width – 1 of the input values.

iii. Zero Padding

The distance which the filter moves across the input from the previous layer each activation is referred to as the stride of the filter. If the size of the previous layer is not evenly divisible by the size of the filters receptive field, and also the size of the stride, it is then possible for the receptive field to attempt to read off the edge of the input feature map. In this case, techniques like zero padding can be used to invent mock inputs for the receptive field to read.

Each layer consists of M maps of equal size (Mx, My). A kernel of size (Kx, Ky) is shifted over the valid region of the input image, meaning that the kernel has to be completely inside the image. The number of pixels that the filter skips in the X and Y directions between subsequent convolutions are defined by skipping factors Sx and Sy. The size of the output map is then defined as:

\[
M_n x = M_{n-1} x - K_n x S_n x + 1 + 1; M_n y = M_{n-1} y - K_n y S_n y + 1 + 1 \tag{1}
\]

where index n indicates the layer. Each map in layer Ln is connected to M_{n-1} maps, at most, in layer Ln–1. Neurons in each given map share weights but consist of different receptive fields.

Other layers that can be involved in building a CNN are pooling layers and fully connected layers. Sometimes these layers are excluded altogether to simply skip nearby pixels prior to convolution, instead of pooling or averaging (Behnke, 2003).

2.4.2. Pooling Layers

The pooling layers’ function by down sampling the feature map from the previous layer. Pooling layers follow one or more convolutional layers in the CNN sequence and are intended to consolidate the features learned and expressed by the feature map from the previous layer. As such, pooling can be considered a technique to generalize feature representations, generally resulting in a reduction of the overfitting of the training data by the model.
The pooling layers also consist of a receptive field; however, it is often much smaller than the convolutional layer. The stride Also, the stride that the receptive field is moved for each activation is often equal to the size of the receptive field to avoid any overlap. The pooling layers are often quite simple, taking the average or the maximum of the input value in order to create its own feature map.

It has been shown that max-pooling can lead to faster convergence select superior invariant features, and improve generalization (Boureau, Ponce, & LeCun, 2010). The output of the max-pooling layer is given by the maximum activation over non-overlapping rectangular regions of size (Kx, Ky). Max-pooling enables position invariance over larger local regions and down samples the input image by a factor of Kx and Ky along each direction (Cirecsan, Meier, Masci, Gambardella, & Schmidhuber, 2011).

2.4.3. Fully Connected Layers

Fully connected layers are the normal flat feed-forward neural network layer. These layers may have a non-linear activation function or a softmax activation in order to output probabilities of class predictions. Fully connected layers are used at the end of the network after feature extraction and consolidation has been performed by the convolutional and pooling layers. They are used to create final non-linear combinations of features and for making predictions by the network. An overview of the use of a CNN is shown in Figure 2.

![Figure 3: Convolutional Neural Network Overview](image)

*Figure 3: Convolutional Neural Network Overview*. This system takes an input, passes it through a series of convolutional, pooling and linear layers, retrieving an output at the end corresponding to the classification task that it was given.
2.5. Model Performance Measurements

Model performance in classification tasks can be measured using the calculation of the 4 different performance measures of the classifier. These are:

1. Accuracy/Classification rate
2. Precision
3. Sensitivity/Recall
4. Specificity

Accuracy refers to the probability that the classification task is performed correctly. Accuracy is measured using the formula:

\[ X = \frac{TP + TN}{TP + TN + FP + FN}, \]

where X is the precision of the classifier, TP is the true positives, TN is the true negatives, FP is the false positives and FN is the false negatives.

Precision refers to the probability that a diagnostic test is performed correctly, when some classes of images come consecutively. It is a degree of the accuracy measure and is given by the formula:

\[ X = \frac{TP}{TP + FP}, \]

where X is the precision.

Sensitivity is a true positive fraction and is the probability that the classification returns a positive result and is given by:

\[ X = \frac{TP}{TP + FN}, \]

where X is the sensitivity.

Specificity is the true negative fraction and refers to the probability that the classification will return negative and is given by:

\[ X = \frac{TN}{TN + FP}, \]

where X is the specificity (Keerthi, Shevade, Bhattacharyya, & Murthy, 2001).
Figure 4: Machine learning performance measurement. This shows how true positives and true negatives are calculated. While also showing false positives and false negatives. The desired outcomes in the classification are true positives and true negatives. Also shown is how precision and recall are calculated.
2.6. Deep learning and GPU computing

Deep convolutional neural networks have recently been shown to have great power at analysing and classifying data sets, ranging from handwritten digits (MNIST) (LeCun, Bottou, Bengio, & Haffner, 1998), to characters and faces (Strigl, Kofler, & Podlipnig, 2010). Deep neural networks are at their best when big and deep, however in a CPU-based system training these deep neural networks can take weeks, or even months. While these deep neural networks offer significant advantages, they also provide significant limitations in that high data transfer latency limits the ability of multi-CPU and multithreading architectures from expediting the process. In recent years fast parallel neural net code for graphics cards has attempted to overcome this limitation, and has provided results that show image processing techniques can be an order of magnitude faster than the CPU counterpart (Strigl et al., 2010; Uetz & Behnke, 2009). While many of these studies have shown correlation between the application of GPU techniques, the majority of studies are based around high throughput classification of sequences of small images. The scale of improvement that GPU based DNN’s offer in the correct classification of high resolution medical imaging for disease state recognition remains unclear.

In the past, the implementation of GPU computing was a specialised task, requiring a deep understanding of hardware architecture, as problems had to be implemented using graphical API’s such as DirectX and then convert it into a graphical pipeline friendly format. However, with the recent development of CUDA, Nvidia’s GPU programming shell, this process has been simplified immensely. The CUDA programming model has three main key abstractions, hierarchy of thread groups, barrier synchronisation, and shared memories. These abstractions provide fine-grained data parallelism together with task and thread parallelism, forming a type of coarse-grain parallelism (Owens et al., 2007). This means programmers are required to partition problems into independently executed blocks of threads that are executed in parallel. Each of these threads are able to cooperate when solving each task, and can also be scheduled for execution on any of the available processor cores, either sequentially or concurrently, meaning at specific core is not required to run any program block, greatly speeding the execution of the programme (Vuduc, Chandramowlishwaran, Choi, Guney, & Shringarpure, 2010).

With regards to image classification this “speed up” scales greatly with the size of the dataset. For example, in the field of medical image analysis, a 4-D (3D image over time) CT
dataset can require up to 10 GB of memory storage, as it can be in the resolution of 512x512x512x20. For a dataset this large even using high cost massively parallel CPU computing, this process could take several hours. However, using the same GPU acceleration as the ultrasound image, this could be calculated within 10 or 15 minutes (Owens et al., 2008; Pratx & Xing, 2011).

Traditionally, a high percentage of classification procedures are carried out using Support Vector machines (SVM’s), and much of the research surrounding the field of supervised machine learning involves these techniques (Catanzaro, Sundaram, & Keutzer, 2008a). However, similar research to quantitatively assess the application and cost-benefit of Deep Learning approaches in CNN’s is lacking (Razavian, Azizpour, Sullivan, & Carlsson, 2014).

2.7. Summary

This chapter provided the context for the research, image classification in medical diagnosis. Then several automated approached to image classification were reviewed. The literature explains how image processing is currently being used in the field of medical image diagnosis and what the cutting edge technologies are. A basic explanation was also given of neural networks and support vector machines, which will be used to answer the research question and support or refute the hypotheses given. In the next chapter, an overview of the design of the experimentation will be given.
3. Design and Implementation

In this chapter, an overview of the experimental design will be given, along with the specifications of the hardware being used and the data source and contents will be given. The aim of the experiment will also be specified.

All experimentation will be carried out using a desktop PC with an Intel i7 5820 processor, a NVidia GTX970 graphics card and 16 gigabytes of on-board RAM.

3.1. Data

The data must be collected and downloaded from the source and cross validated, to ensure the model has enough information to correctly classify the images. The classification procedure will attempt to classify the stages of diabetic retinopathy from images taken of eyes. Data is provided by EyePACS, which is a free platform for retinal screening. The data exploration task will seek to identify the key areas to focus on during research. This includes the identification of features for classification and the feature extraction methodology to be used. Depending on the features selected; colour, texture, shape etc., a different feature extraction technique will be required.

The aim of this project is to solve a complex image processing task, using both CNN and SVM methodologies, taking advantage of Deep Learning techniques where possible, and to investigate both the differences in performance among the different methods, but also to critically assess the relative strengths and weaknesses of the different methodologies. The experimentation will be carried out using a freely assessable EyePACS dataset, consisting of ~30,000 training images of either varying level diabetic retinopathy patients, or NULL cases. This dataset also provides 55,000 test cases, for use after training the model and a master results set, which will provide certainty about the model accuracy. These designs will aim to be as similar as possible to critically assess the strengths of the design, especially in the case of the CNN, however differences will be present despite this as the different frameworks have different requirements in the design process. This process will be contrasted with a support vector machine (SVM) classifier. This type of process was selected due to its status as a more “classic” approach, however it is limited in its ability to handle complex images, therefore the time taken to train the classifier is traditionally large. The process of classification will use a linear SVM. Given an input image, the system first extracts dense local descriptors, which is coded either using local coordinate coding. The codes of the descriptors are then passed to
weighted pooling to form a vector for representing the image. Finally, the feature vector is fed to a linear SVM for classification (Catanzaro, Sundaram, & Keutzer, 2008; Lin et al., 2010).

Both of these techniques will be evaluated in the same way, and using the resultant information and the time taking to proceed through the steps we will be able to accept or reject the null hypothesis, as per the experimental overview shown in Figure 1.

3.2. SVM Configuration

The SVM design methodology used the SVM package from the sklearn module in python. This module allowed the classification to proceed without needing too many lines of code. However, it would be difficult to carry out the full classification of the dataset using the SVM technique as it is a resource hungry process and the size and scale of the images, and the dataset itself, suggests that a vast amount of feature and dimensionality reduction would be required before classification could proceed. The method would combine principal component analysis (PCA) along with a reduced size dataset, which was generated from the source images using the supplementary code shown in Appendix 1.

3.3. CNN Configuration

The CNN methodologies proposed here consisted of convolutional neural networks trained with various libraries until a “best performer” was established, using the large colour images, various data augmentation techniques, dynamic resampling for class imbalance between both diseased and non-diseased, which also makes considerations for the different disease states and also a “per patient” feature blending strategy which makes the most of the availability of having two sets of images per patient, which is taken advantage of in the system. The solution to the problem proposed is to use a simple average of blends, consisting of features from two different deep convolutional networks and three sets of weights for each network, forming a robust and powerful classifier. The first implementation of the CNN will begin with the training of the classifier using stochastic gradient decent, using a small weight decay <0.005%. The literature has suggested that this is the correct number to use to ensure the model learns correctly (Krizhevsky, Sutskever, & Hinton, 2012; Liu, Fang, Zhao, Wang, & Zhang,
The CNN methodologies proposed here will consist of convolutional neural networks trained with various libraries until a “best performer” is established, using the large colour images, various data augmentation techniques, dynamic resampling for class imbalance between both diseased and non-diseased, which will also make considerations for the different disease states and also a “per patient” feature blending strategy which makes the most of the availability of having two sets of images per patient, again which will be taken advantage of in the system. The solution to the problem proposed will to use a simple average of blends, consisting of features from two different deep convolutional networks and three sets of weights for each network, forming a robust and powerful classifier.

3.4. Summary

This chapter gave an overview of the CNN and SVM experimental process and gave a short description on where the data being used came from. A brief introduction was given to the model design which will be expanded upon in the next chapter. The next chapter will assess how these methods were implemented and the results of the model designs that have been implemented, along with any feature or parameter engineering that was required.
4. Implementation and Results

In this chapter the implementation and model design of the CNN and SVM models will be shown, along with some findings from the design process that lead to the final models used. A discussion of the different CNN frameworks being tested will be given and an overview of the image processing being imposed on the data prior to model training.

4.1. Image Pre-Processing

The images in this dataset are too large to be conveniently trained with convolutional networks and even the high end consumer grade hardware that was available to run the process. Therefore, to correctly build a functioning network, a reduction in image size, and therefore performance overhead was required. Since the original images are fairly large (approx. 3000x2000 pixels on average) and most contained a fairly significant black border, the first stage of the development process was to downscale all the images by a factor of five (without interpolation) and trying to remove most of these black borders. The original images are shown in Figure 5, while the resultant images of this processing step are shown in Figure 6. The images were further reduced in size, to squares of 128, 265 and 512 pixels respectively. This gave an even spread of images across multiple file sizes. After this had been completed, background subtraction was used to remove the large black border present in the images.
Figure 5: Source training images. This image shows the state of the source images prior to the pre-processing tasks, including size discrepancies and a large black border being present.

Figure 6: Border Removal. The images from Figure 5, however the black border has been cropped out in this example, and the images have been downscaled by a factor of 5.

To further improve the quality of the resultant training, the images were subjected to further transformations to convince the model that there were many similar images present in
the dataset, despite seeming different at face value. The transformations that were carried out are shown below.

These transformations were as follows:

1. The images were cropped with certain probability
2. The images were adjusted for colour balance
3. Adjusting the brightness of the images
4. Adjusting the contrast of the images
5. Flipping some of the images (50% chance)
6. Rotating images by x degrees, somewhere between 0 and 360
7. Equal cropping of X and Y axis

Most of these were implemented from the start. During training random samples are picked from the training set and transformed before queueing them for input to the network. The augmentations were done by spawning different threads on the CPU such that there was almost no delay in waiting for another batch of samples. A collage of the resultant images from this stage is shown in Figure 7.

![Collage of Transformed images](image)

*Figure 7: Collage of Transformed images.* This figure shows a collage of 512 x 512 pixel images, which have been cropped and rotated to generate a good field of view.

A resizing transformation was carried out after the cropping and before the other transformations, as it can make some of other operations computationally intensive. This process can be carried out in two different ways:
1. Rescale, sparing the original aspect ratio and subsequently carrying out a centre crop on the resulting image

2. Normal bilinear rescaling, however this will destroy the aspect ratio originally present

The method chosen also depends on the model. Early designs focused more on method 1, however during development it was seen that the second method offered some improvements in model performance, therefore the design was revisited. Method 2 was used going forward from here as the best performing models took advantage of this and the addition of another hyperparameter was deemed to be suboptimal as there was a risk of information loss with the centre crops from the first method.

During the training the input is normalised by subtracting the total mean and dividing by the total standard deviation estimated on a few hundred samples before training. After this process was completed, the images were available for testing on the various different models that were highlighted as options to complete this task, broken up between CNN and SVM models.

4.2. Development of SVM

With regards to the SVM methodology, the proposed work was adapted to combat the inherent resource heavy dependency of the SVM classifier. Early approaches sought to classify across the entire dataset, similar to the CNN approach. However, this became unfeasible due to the scope of the used dataset as the vast array of images. Therefore, to correctly build an SVM classifier, a vast reduction in dimensionality was required. To accomplish this, some feature and parameter engineering was required. From the original dataset of 55,000 images, only 3000 could be loaded into an SVM and trained correctly, even with the dimensionality reduction allowed by the engineering tasks. The proposed method consisted of a feature reduction stage and a classification stage using SVM.

For the classification stage we need to extract significant information of each image, encode it as efficiently as possible, and compare one encoded image with a data set using a model encoder based on similarly. Therefore, we have used principal component analysis in order to find the significant features (principal components) of the distribution of the image data set.
The basis of principal component analysis is to find the vectors that best account for the distribution of fundus images within the entire image space. This analysis is as follows: Given a training data set of fundus images I1, I2, ..., IM, where every image Ii will be represented as a vector Γi. Then the average fundus image vector Ψ will be calculated as follows:

$$\Psi = \frac{1}{M} \sum_{i=1}^{M} \Gamma_i$$

where each fundus image differs from the average by the vector Γ, i.e. it subtracts the mean fundus image:

$$\phi_i = \Gamma_i - \Psi$$

This set of very large vectors is then subject to principal components analysis, which seeks a set of M orthonormal vectors un and the eigenvalues (scalars) λk, respectively, which best describe the distribution of the data. The eigenvectors uk and eigenvalues λk are obtained from the covariance matrix C. The λk are the elements in the diagonal of the matrix C.

$$C = 1\ M\ M\ n = 1\ \phi n\Phi^T = AAT$$

where the matrix,

$$A = [\phi1\phi2...\phiM].$$

The associated eigenvalues allow us to rank the eigenvectors according to their usefulness in characterizing the variation among the objects (fundus images).

For training the SVM classifier, the Kernel-Adatron technique using a Gaussian kernel was used. To obtain the optimal values for the Gaussian kernel (s) and C we experimented with different SVM classifiers using a range of values. Tenfold cross-validation was applied to find the best classifier based on validation error. The performance of the selected SVMs was quantified based on its sensitivity, specificity and the overall accuracy. In the first experiment, with no restrictions on the Lagrange multipliers (hard margin), we achieved an overall accuracy of 88.6% with 86.2% sensitivity and 90.1% specificity for s=0.3.
4.3. Development of CNN

The CNN developments all followed a similar path, the training and test sets came as a subset of the training dataset available as there was a master solution file available with the data. Therefore, any development could be exactly quantified based on ease of use, success rate, time taken to run and development overhead. As there are a great many frameworks available, 2 were selected as potential candidates for testing, Torch and Theano. These models would be generated on a miniaturised dataset, and critically compared to each other, to generate the optimal model for this work.

4.3.1. Theano Implementation

The first architecture used was the Theano framework. The network architecture of the Theano designed convolutional network is shown in Figure 8.
The networks were trained using rectifier linear units (ReLU), which were used over other activation functions, such as tanh’s or sigmoids, as they can be more easily developed by thresholding the matrix of activations at zero, without suffering from saturation. In some cases, ReLU’s have been shown to greatly accelerate the convergence of gradient descent compared to the other functions, due to the linear form of the function. Unfortunately, ReLU units can be fragile during the training process and can “die”. For example, a large gradient flowing through a ReLU neuron could cause the weights to update in such a way that the neuron will never activate on any data point again. Thus, from that point on, the gradient of that unit will always be zero and can never alter again. With a learning rate set too high, this can be an exponential problem, with a large percentage of the network being dead, causing substantial classification accuracy and implementation difficulties. To combat this, a solution is to use leaky ReLUs, which is one method of overcoming this limitation and fix the “dying ReLU” problem. Leaky RELU’s operate similarly to regular rectifiers, except Instead of the function being zero when $x$
< 0, a leaky ReLU will instead have a small negative slope (usually approx. 0.01). For the purposes of nonlinearity leaky rectifier units are used here following each convolutional and fully connected (dense) layer. The networks were trained with nesterov acceleration with fixed schedule.

For the networks run on the 256 and 128 pixel images, training is stopped immediately after 200 epochs. L2 weight decay is added at a small value, approx. 0.0005, as there are a large number of training examples in the dataset. The problem then becomes a simple regression issue, with mean squared error objective and threshold at (0.5, 1.5, 2.5, 3.5) to obtain integer levels for computing the kappa scores.

As the classes within the dataset are extremely unbalanced, it was difficult to determine the optimal strategy for handling the neural network generation, however the following solution was devised. Initial sampling was carried out across all classes, such that every class was represented equally on average. Gradually, the oversampling of the rarely instances was wound down to give a better view of how the dataset actually appeared. The weights for the resampling between the different disease levels, 0-4 at a given epoch, t, are given as follows:

\[ w_i = r t - 1 w_0 + (1 - r t - 1) w_f \]

where \( r \) was set at 0.975, \( w_0 \) were approximately 1.36, 14.4, 6.64, 40.2 and 49.6 and \( w_f \) were set at 1, 2, 2, 2 and 2. These were all values which were found to work well for initial convergence and final classification. It is probable that an even higher classification accuracy could be achieved by using a dynamically weighted objective function within the theano library, however this was ruled out due to a difficult implementation and unfamiliarity with the minutia of the system architecture.

In this model, the best solution that could be devised was to use 10% of the reduced dataset as a validation set, equating to roughly 590 images from a pool of 5900. The original strategy was to implement and train the network entirely from scratch, however in practice this was deemed too difficult for this particular issue. Instead a smaller network was first trained on the 128 pixel images, which was then used as a base to use the trained weights to initialize partly formed networks of intermediate size which could be implemented on the 256 pixel images. The weights from this were then used to repeat this procedure for the final networks that on the 512 pixel images. The layers of the model were constructed as follows, the 128 px images were used to build layers 1 – 11 and 20 – 25, the 256 px images were used for layers 1 – 15 and 20 – 25, (with the weights of layers 1 – 11 being initialised from the previous weights)
and the 512 px images being used for all layers, with weight initialising being the same for the 1 – 11 layers.

Various transformations were carried out at all times during the running of the network. Translation, stretching, rotation, flipping as well as colour augmentation were all dynamically implemented when needed. Each image channel (RGB) was scaled and centred to have zero mean and unit variance over the training set. The output sizes that were generated for the data augmentation pipeline were 112/224/448 pixels for the 128/256/512 pixel input images.

The feature extraction processes were then carried out at the last pooling layer of the convolutional neural networks. Since the feature extraction process was so vital to the success of the training procedure, it is important to increase the quality of the features at this point, to prevent extraction of low quality features. This process was repeated approximately 50 times, with varying s, per image and then mean and standard deviation of each feature is calculated and used as an input for the blending network.

The mean ($\mu$) and standard deviation ($\sigma$) of the RMSPool layer output were next extracted for 50 pseudo random transformations for three sets of weights, these being best validation score, best kappa and final weights, for both network a and network b. For each eye, corresponding to each patient, the following were used as the input blending features:

$$x = (\mu_{\text{this eye}}, \mu_{\text{other eye}}, \sigma_{\text{this eye}}, \sigma_{\text{other eye}}, \delta_{\text{right}})$$

where $\delta_{\text{right}} \in \{0, 1\}$ was selected as an indicator variable for right eyes. All these features were standardised, and therefore have no mean and unit variance throughout and can therefore be used to train the fully connected network, whose architecture is shown below in Table 1. This is a well understood blend architecture for use in CNN design (Liu, Fang, Zhao, Wang, & Zhang, 2015; Sharif Razavian, Azizpour, Sullivan, & Carlsson, 2014).

| Table 1: Theano Blend Network Architecture |
|------------------|------------------|
| Input            | 8193             |
| Dense            | 32               |
| Maxout           | 16               |
| Dense            | 32               |
| Maxout           | 16               |

This is a well understood blend architecture for use in CNN design (Liu, Fang, Zhao, Wang, & Zhang, 2015; Sharif Razavian, Azizpour, Sullivan, & Carlsson, 2014).
4.3.2. Torch Implementation

The first methodology used the Torch and cuDNN libraries for all of the computational work. All work was carried out on the same hardware across all of the different architecture designs using the pre-processed images at the percentage of 10% of the total, by altering the random sample generator script to select this percentage of images. The architecture of the torch model is shown in Figure 9

<table>
<thead>
<tr>
<th>Num</th>
<th>Batch</th>
<th>Channels</th>
<th>Width</th>
<th>Height</th>
<th>filter</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>64</td>
<td>448</td>
<td>448</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Conv</td>
<td>64</td>
<td>32</td>
<td>224</td>
<td>224</td>
</tr>
<tr>
<td>3</td>
<td>Conv</td>
<td>64</td>
<td>32</td>
<td>224</td>
<td>224</td>
</tr>
<tr>
<td>4</td>
<td>MaxPool</td>
<td>64</td>
<td>111</td>
<td>111</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>Conv</td>
<td>64</td>
<td>64</td>
<td>56</td>
<td>56</td>
</tr>
<tr>
<td>6</td>
<td>Conv</td>
<td>64</td>
<td>64</td>
<td>56</td>
<td>56</td>
</tr>
<tr>
<td>7</td>
<td>Conv</td>
<td>64</td>
<td>64</td>
<td>56</td>
<td>56</td>
</tr>
<tr>
<td>8</td>
<td>MaxPool</td>
<td>64</td>
<td>27</td>
<td>27</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>Conv</td>
<td>64</td>
<td>128</td>
<td>27</td>
<td>27</td>
</tr>
<tr>
<td>10</td>
<td>Conv</td>
<td>64</td>
<td>128</td>
<td>27</td>
<td>27</td>
</tr>
<tr>
<td>11</td>
<td>Conv</td>
<td>64</td>
<td>128</td>
<td>27</td>
<td>27</td>
</tr>
<tr>
<td>12</td>
<td>MaxPool</td>
<td>64</td>
<td>13</td>
<td>13</td>
<td>3</td>
</tr>
<tr>
<td>13</td>
<td>Conv</td>
<td>64</td>
<td>256</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>14</td>
<td>Conv</td>
<td>64</td>
<td>256</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>15</td>
<td>Conv</td>
<td>64</td>
<td>256</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>16</td>
<td>MaxPool</td>
<td>64</td>
<td>6</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>17</td>
<td>Conv</td>
<td>64</td>
<td>512</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>18</td>
<td>Conv</td>
<td>64</td>
<td>512</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>19</td>
<td>RMSPool</td>
<td>64</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>20</td>
<td>Dropout</td>
<td>32</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>Dense</td>
<td>64</td>
<td>1024</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>Maxout</td>
<td>64</td>
<td>512</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>Dropout</td>
<td>32</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>Dense</td>
<td>64</td>
<td>1024</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>Maxout</td>
<td>64</td>
<td>512</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 9: Torch network architecture. Shown here is the architecture and positions of the various layers in the Torch CNN

The torch model was trained using a clipped MSE error function. This was the standard MSE process, with the output of the model clipped to [0, 4] before evaluation. This was an experiment carried out only for this model as the literature suggested that allowing the model
to generate scores lower than zero and greater than 4, would allow for an increased output range, which would assist in separating the different diseased classes. This model also used sigmoid functions, as there was a combination of 4 different sigmoid functions used as the loss. Furthermore, this was used as an opportunity to alter some of the image processing parameters, to assess if the application of GPU computing could increase the speed of the processing calls. This was changed between the Torch and Theano methods as an error exists when attempting to call the WarpAffline function on Theano. The following details were shared by all of our models:

1. Predictions were generated one eye at a time
2. Data transformations were carried out via random affine transformation. It was possible to do these calculations in the GPU, using a dedicated function from NVidia, called WarpAffine. As the GPU is computationally far more efficient than the CPU this increased the speed of these calculations greatly resulted in increased GPU memory usage, but was faster than performing the operations on the CPU. In a typical setup, images were randomly cropped to 85-95%, horizontally flipped, rotated between 0 and 360 degrees and then scaled to the desired model input size
3. Channel-wise global contrast normalization was applied to normalize image color
4. PreLU weight initialization. This was found to help train models with large numbers of layers
5. Leaky rectifier non-linearities (0.1 leak factor)

In both model cases, the design sought to exploit the fact that each patient, in both training and test datasets, had a left and right eye image present. Therefore, a simple linear model was generated to pool the predictions from both eyes. An early 10% early stopping validation set was reused to fit this model. Several attempts were made unsuccessfully to train a neural network architecture that used both eyes as input. It is possible that further experience with modelling would have allowed this to proceed, however it was abandoned early due to excessive time constraints with both research and implementation.

In the torch model, a similar 10% validation set was used. There were small differences in the layer construction between this and the Theano model as seen in the network architectures in Figs (), however images were used to build layers in a similar way. This model was the first to use PreLU weight initialisation, which was then added into the Theano model,
as it was found to help train models that have a large number of layers, and is supported by the literature in this regard (He, Zhang, Ren, & Sun, 2015).

4.4. SVM Implementation

Image pre-processing steps were more advanced for the SVM implementation as a number of feature engineering tasks were required, as the SVM does not have the power of the CNN in terms of easily identifying region of interest (ROI). In this case a number of steps were used, such as gray scale conversion, adaptive histogram equalisation, discrete wavelet transform, PCA and fuzzy C-means clustering for the segmentation task. All image preprocessing tasks at this stage are carried out using the ImageJ processing tool.

4.4.1. Grey Scale Conversion

As all the images are presented in an RGB format, these must be first converted to a grey scale, which is commonly carried out by matching the luminance of the colour image. A greyscale image is an image that only encodes intensity information. The grey scale conversion was carried out using ImageJ’s luminance module. An example image is shown in Figure 10, created using the montage feature of ImageJ.
4.4.2 Adaptive Histogram Equalisation (AHE)

AHE is a technique used to improve the quality of contrast in images, by enhancing the local contrast features of the image. The objective of this method is to define a point transformation, with the assumption that intensity values within the local window are a stoical representation of the distribution of intensity values within the whole image. The point transformation distribution is localised around the mean intensity of the window, covering the entire intensity range present in the image. The result of this AHE is that the dark area in the image that was badly illuminated is brighter in the output, while keeping the areas that were already brightly illuminated the same. The result of this process is shown in Figure 11.
4.4.3. Wavelet Transformation

A transformation task changes the representation of the signal without altering the information content of the image. The wavelet transformation is a multi-transformation technique in which different frequencies are used with different resolutions. A discrete wavelet transformation has been seen to yield a fast computation of wavelet transforms, which is easy to implement and reduces the resources required (Priya & Aruna, 2010; Vekket & Shukla, 2009). The wavelet transform breaks the signal into a set of base functions, called wavelets. These wavelets are obtained from a single wavelet prototype known as the mother wavelet. In this case, the mother wavelet is used to generate all basis functions based on the desired characteristics, in this case the approximation coefficients matrix and details coefficients matrix of input X, where X is a given input image. In this case the haar wavelet is used, and the usage of this transform reduces the size of each image by half. The resultant transformations are shown below in Figure 12.

![Figure 12: Results of the wavelet transformation step of the feature engineering stage](image)

4.4.4. Principal Component Analysis

PCA was used next to make a classifier system more effective, as discussed previously. Prior to the classification step, a dimensionality reduction stage was required. The PCA task is based on the assumption that most information about classes is contained in the directions along which the variations are the largest. There is an assumption made for feature extraction and dimensionality reduction by PCA, that most information of the observation vectors is contained
in the subspace spanned by the first \( m \) principal axes, where \( m < p \), where \( p \) is the dimension of the dataset and \( m \) is the principal axis.

4.4.5. Fuzzy \( c \)-Means Clustering

FCM is a clustering method that allows one price of data to belong with multiple clusters. Upon investigation of the literature, it was decided to focus on a \( c \) means method as it has been shown in medical diagnostic systems that this can outperform a harder \( k \) means method. In this case, the clustering is based around detecting blood vessels that can be used to grade the severity of the disease. The clustering is carried out by detecting the blood vessels in the eye images and grouping these to one category as pixels which are turned on, and all other pixels, which are turned off. The results of this process are shown below in Figure 13.

![Figure 13: Results of the application of fuzzy c-means clustering to the sample images](image)

4.4.6. Feature Extraction

After these stages have been completed, the feature extraction stage is carried out, which extracts information from the image to use as inputs into the SVM. These features include

1. Area of on pixels: The area of white pixels remaining in the image
2. Mean: the average distribution of all pixels divided by the original values
3. Standard Deviation: The square of every pixel of all individual samples is collected and then an average across these \( N \) samples is calculated

4.4.7. SVM Classification
After the processing steps have been completed, the SVM algorithm is used to produce the classification parameters for the calculated features that have been derived from the processing stages. The algorithm should break down the images into their respective classes based on level of disease state. In this case, the image is considered to be classified correctly if the fundus of the image is actually abnormal and has been screened as abnormal. The goal of the SVM here is to find the optimal hyperplane that separates the clusters of vectors in such a way that the cases with one characteristic are on one side of the plane and those without are on the other. In this case, diseased vs. non-diseased, then further separated based on the severity of the condition. The SVM makes use of a function called a kernel function, which maps the data into a different feature space where the hyperplane can be used to do the separation task.

4.5. Summary

In this section the implementation of the image pre-processing steps, along with the development and implementation of the various models was discussed. Various feature and parameter engineering steps required were covered in detail and the stepwise process for how each classification would proceed was also discussed. As shown below in Table 2, there were various different stages involved in all the different methodologies before the models were fully developed, showing the breadth of options available for each technique and the level to which they were implemented in this case.
Table 2: Summary of the various models being investigated and the parameter and feature engineering tasks involved with their development.

<table>
<thead>
<tr>
<th>Techniques</th>
<th>Frameworks</th>
<th>Parameter Engineering</th>
<th>Feature Engineering</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN</td>
<td>Theano</td>
<td>Decoding alterations</td>
<td>None; features were extracted at last pooling layer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Rectifier unit design</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Batch Normalisation</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Layer modelling</td>
<td></td>
</tr>
<tr>
<td>Torch</td>
<td>Clipped MSE Function</td>
<td>None; features were extracted at last pooling layer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Random Affine Transformations</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Pre-Lu Weight initialisation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SVM</td>
<td>Scikit-Learn</td>
<td>Reduction in image data to ensure model functionality</td>
<td>Grey-scale conversion</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Adaptive Histogram Equalisation</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Discrete Wavelet Transformation</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Principal Component Analysis</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fuzzy C-means Clustering</td>
<td></td>
</tr>
</tbody>
</table>
5. Evaluation and Analysis

In this chapter the results of the model designs will be described and compared, giving justification as to what models were used in the end, and why other models were excluded. The results of the models were evaluated using the following criteria:

5. Adaptability of the frameworks
6. Ease of use
7. Training times
8. Confusion matrixes

Within these evaluation methods, however, confusion matrixes were given heavy weight in the final selection process as they offered a simple way to assess model performance.

5.1. CNN Evaluation

CNN design was a long process as there is a large amount of available libraries from which to develop a model. In this project some of the more common libraries were investigated, namely theano and torch. Each of these libraries offers a wide range of applicable features and thorough testing of all would leave little between them in terms of quality of build and applicability. However, for the purposes of this experiment, one was necessary for use and there were defined use-cases which helped narrow down the list.

To narrow down the list of possible libraries/frameworks to use, models were required to be designed and run on each of them on a small subset of the data using the random sample extractor code found in the appendix. This code can select a random number of images, specified in the code as any percentage of the total data. As there was reason to believe that each of the models would take a long time to train, selection a small amount, approx. 5% of all the images available, was deemed an acceptable first stage of the selection process.

The data set for this competition is difficult to manage, insofar as the images themselves contain features important to the classification, for example micro aneurysms, which are relatively small with respect to the field of view, i.e. the retina. With regard to the performance of the classification itself, results suggested that the use of large images was essential to the
successful generation of a model. The torch model was the “testing environment” for altering the size of the images to see if it improved the ability to classify the images. Using the image sizes provided from the conversion process, 128, 256 and 512 pixels respectively, a great increase in performance was shown. The results of the classification went from ~70% for the 256 images to ~80% for the 512 pixel images, however this was also accompanied by a vast increase in testing time.

The leaky rate of the rectifier was also shown here to have a great impact on the classification rate of the model, as the leaky rate was increase, the overall classification accuracy decreased. For example, using the standard leaky rate (0.01) of the classifier, the overall score was approx. 84%, however when the rectifiers were changed to using “very” leaky settings (0.33) and the model retrained, the success rate fell to 83%. This difference is quite marginal, but has a great impact on both the selection of this model as the standard and in a real life scenario, incorrectly classifying a patient’s diabetes is a very large problem. After investigating other causes of the fall in success, it was concluded that either a change in the rectifier or the training procedure had been the root of the issue and further experimentation used the 0.01 rate leaky rectifiers (Behnke, 2003; Maas, Hannun, & Ng, 2013).

With regards to the Theano model, the architecture of this model did not vary greatly from the outset, and only minor additions and alterations were implemented. Most of the improvements in this model came from optimising the learning process. The most accurate model came from a simple log mean ensemble, created from 2 or 3 existing models. Similar to the torch model, the method most focused on was a combination of the left and right eyes, as it was deduced that taking advantage of having two different eyes was the optimal strategy, as if one was classified as having a diseased state, the other should too. Therefore, any instances of one having the diseased state and the other having it absent could be simply deemed as incorrect. This method was implemented by using a merge of the outputs of the first convolutional or pooling layer for each of the two eyes. In theory it should then be possible for the convolutional layer to be able to detect similar patterns in the left and right eye. However, it was determined that this was reducing the input space too much after merging and instead a different method, using a merging of outputs A and B from the output of some layer and these were then replaced with AB and BA by stacking on the channel, instead of replacing with AB. This allowed the network to still have access to low level representations but retained all the input space.
Improvements with this methodology came from altering the decoding. Initial builds simply used the class with the highest probability from the softmax output as the predictor. However, this methodology was inherently limited by the fact that it does not take magnitude of the other probabilities into account. Therefore, the decoding process had to be optimised to generate the optimal prediction. As this was a completely foreign area, direction was taken from a recent study that represented ranking in a similar way (Galluppi & Furber, 2011). In this method, the probabilities from the output are converted to one value by weighing each probability by the class label \{0, 1, 2, 3, 4\}. Next the optimal probabilities are ranked from high to low, and the goal is to identify the optimal boundaries, \([x_0, x_1, x_2, x_3, \ldots]\) etc. For example, the images with a weighted probability in \([0, x_0]\) the label 0 is assigned, etc. This process assisted in increasing the classification accuracy, but not by too much, an increase of approx. 1\%. This process likely could have been fine tuned to generate improved results, however there is a risk of badly overfitting the model by doing too much of the classification work for the model.

A number of other methods were tried to improve performance but generated no extreme improvements over the standard results. Examples of these attempted improvements are shown below

- Batch Normalisation, this process allowed for higher learning rates, however in this implementation it was found to not speed up training to a very large degree, however there is sure to be some way of improving the training with more elegant implementations.
- An attempt was made to remove the pooling layers and replace all the 3//2 (pool size 3x3 and stride 2x2) max pooling layers with a 3//2 convolutional layer instead. This had a negative impact on the training time and also reduced the performance generated.
- Another idea was to replace the 3//2 max pooling layer with a 2//2, as a lower pool size should have allowed the model to differentiate finer details, however it generated no improvement in model performance. This could be pushed further with regards to feature modelling as the features in this study include things such as micro-aneurysms, which are generally located in a particular area and improvements in locating these would surely lead to a better model performance.
- Multiple other options were attempted, such as adding additional convolutional layers, adding more pooling layers, replacing the first 7//2 convolutional and 3//2 pooling layer by two 5//2 convolutional layers, etc. However, these changes didn’t improve the model
to any significant degree and were abandoned as there was a time constraint on each attempt due to the long training times.

5.1.1. Model Accuracy Measures

To determine which was the optimal approach for the CNN implementations, the optimal design of each methodology was assessed using a confusion matrix, along with a number of literature and ease of implementation based optimisations for this task and finally the recorded time of each epoch and the complete model was assessed, however not expected to be largely different between models. Shown below in Table 3, is a confusion matrix for the optimal Theano implementation. This model presented with an accuracy of 90.25%, which was a very high result. The sensitivity of this model at class 0 was also particularly high, at 97%, suggesting that this model was especially good at recognising when one did not have the disease in the image in question, however the specificity was less impressive at only 79% within the same level, suggesting that the model was less accurate at predicting a true negative in this class. Within the next class, class 1, which was the first disease level, the model sensitivity was especially bad at 55% suggesting there was a minute difference between class 0 and class 1, confusing the model somewhat. However, within this sample the prevalence of class I patients was only 11%, compared with 71% from class 0, meaning that incorrect classification at this level had additional weight. Class 2 and class 4 maintain high sensitivity and specificity, however class 3 also has a low sensitivity, suggesting the model had difficulty recognising class 3 patients.

Table 3: Theano CNN Confusion Matrix, showing the predicted and expected classifications for each disease level from the reduced randomised dataset used for the CNN optimal design selection

<table>
<thead>
<tr>
<th>Prediction</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4123 31 23 2 2</td>
</tr>
<tr>
<td>1</td>
<td>47 377 0 14 2</td>
</tr>
<tr>
<td>2</td>
<td>56 57 709 33 3</td>
</tr>
<tr>
<td>3</td>
<td>6 0 1 109 0</td>
</tr>
<tr>
<td>4</td>
<td>9 0 1 8 92</td>
</tr>
</tbody>
</table>

Accuracy 0.9025

P-Value [Acc > NIR] < 2.2e-16
Shown below in Table 4, is the confusion matrix for the optimal torch implementation of the CNN. This implementation was not shown to have a large degree of difference in accuracy to that of the Theano implementation, with an accuracy rating of 87%. Again, similar to the Theano model, class 0 had a high sensitivity rating, higher even in this case. However, the torch models were shown to have more difficulty recognising the first disease class, with a sensitivity of only 50%, in contrast to the 56% in the Theano case. This model had more difficulty classifying stage 2 and 4 classes positively, compared to the Theano models, however could positively predict class 3 patients with more success, at 61%. The specificity of both models was roughly equal, with the specificity of class 0 being slightly lower in this case. There was slightly different weighting of each class in this implementation, with the prevalence of class 0 being only 66%, in contrast to the Theano’s 71%, however this should not have had an extreme impact on the results.

**Table 4: Torch CNN Confusion Matrix**, showing the predicted and expected classifications for each disease level from the reduced randomised dataset used for the CNN optimal design selection.

<table>
<thead>
<tr>
<th>Prediction</th>
<th>Reference</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3907</td>
<td>233</td>
<td>38</td>
<td>14</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>23</td>
<td>532</td>
<td>38</td>
<td>10</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>26</td>
<td>69</td>
<td>750</td>
<td>38</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>12</td>
<td>0</td>
<td>116</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>7</td>
<td>2</td>
<td>12</td>
<td>102</td>
<td></td>
</tr>
</tbody>
</table>

Accuracy: 0.8754  
P-Value [Acc > NIR]: < 2.2e-16

<table>
<thead>
<tr>
<th>Class: 0</th>
<th>Class: 1</th>
<th>Class: 2</th>
<th>Class: 3</th>
<th>Class: 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensitivity</td>
<td>0.9861</td>
<td>0.50921</td>
<td>0.9058</td>
<td>0.61053</td>
</tr>
<tr>
<td>Specificity</td>
<td>0.7548</td>
<td>0.98489</td>
<td>0.9732</td>
<td>0.99670</td>
</tr>
<tr>
<td>Prevalence</td>
<td>0.6661</td>
<td>0.14341</td>
<td>0.1392</td>
<td>0.03194</td>
</tr>
</tbody>
</table>

One of the criterion being used to select an optimal implementation to use the time taken to train the models comparatively. As there was a large number of epochs required to
train each model, running time was a large limiting factor for each design. The torch model was faster per epoch than the Theano model, with each epoch taking 8.5 minutes, and also consisted of less epochs to train the model itself, with only 150 epochs being required. This gave the model a total running time of 1,275 minutes, equating to slightly over 21 hours of training time. Conversely, the Theano model required 11.75 minutes per epoch, across a larger number of total epochs, giving a total of 2,350 minutes of training time, which was almost 39 hours. This is summarised below in Table 5. This was a very large difference in model training times.

Table 5: Training times of both CNN implementations, given in minutes per epoch and combined for a total time.

<table>
<thead>
<tr>
<th>Model</th>
<th>Number of Epochs</th>
<th>Time Per Epoch (min)</th>
<th>Total Training Time (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Torch</td>
<td>150</td>
<td>8.5</td>
<td>1,275</td>
</tr>
<tr>
<td>Theano</td>
<td>200</td>
<td>11.75</td>
<td>2,350</td>
</tr>
</tbody>
</table>

Had time been the only criterion, it is likely that the torch model would have been selected, however it also had a lesser total performance to the Theano model. There were also a number of framework differences in pros and cons between Theano and Torch that impacted the decision to use one over the other. In the case of Theano, it consists of a very low level raw framework, which can make it very difficult to work with, which was found in this implementation before using some of the wrapper libraries, in this case Lasagne. This made the design much easier. Theano is also a much “bulkier” program and has limited support for any pre-trained models. Conversely it is very well supported in the academic community and there is a large amount of documentation available, along with very good integration with Python and the numpy module, which was essential to the tasks at hand.

In contrast to this, Torch is a framework that is written in Lua, so while it is easier to read the library code for a non-specialist, torch isn’t as academically friendly as it is not designed around python, as Lua has a more native integration with Java. This means that Torch is more naturally geared towards software development, and can be seen by it being the tool of choice for companies such as Facebook. In the case of this experiment, torch was seen to be much more difficult to implement as there is not nearly as much support available for non-commercial torch users, being more industry focused.
5.1.2. CNN Model Selection

For the selection of the model, it was therefore decided to use Theano as the framework of choice as it offered superior performance and ease of implementation, while having a slower training time. Furthermore, torch required a strong familiarity with Lua and a python based approach was deemed easier to fully implement. Finally, despite numerous attempts, it was never possible to get the Torch based models to match this performance and it is possible that the design of the model itself was simply not as effective as the Theano models. The Theano design used a more expressive model and fit it better to the whole dataset.

When the Theano model was scaled up to the entirety, it generated the results shown in Table 6. The results stayed roughly the same, with the accuracy of the model raising slightly to 90.64%. Training time experienced a significant increase as a result of training 10x as many images, however the model ran without incident. The full confusion matrix can be seen in Appendix 1.

Table 6: Confusion matrix of the full Theano CNN implementation, showing results of classifications and descriptive statistics.

<table>
<thead>
<tr>
<th>Prediction</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>35123</td>
</tr>
<tr>
<td>1</td>
<td>1220</td>
</tr>
<tr>
<td>2</td>
<td>290</td>
</tr>
<tr>
<td>3</td>
<td>49</td>
</tr>
<tr>
<td>4</td>
<td>26</td>
</tr>
</tbody>
</table>

Accuracy: 0.9064
P-Value [Acc > NIR]: < 2.2e-16

<table>
<thead>
<tr>
<th>Sensitivity</th>
<th>Class: 0</th>
<th>Class: 1</th>
<th>Class: 2</th>
<th>Class: 3</th>
<th>Class: 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9336</td>
<td>0.5823</td>
<td>0.9262</td>
<td>0.65809</td>
<td>0.92199</td>
<td></td>
</tr>
<tr>
<td>Specificity</td>
<td>0.8112</td>
<td>0.98794</td>
<td>0.9770</td>
<td>0.99830</td>
<td>0.99683</td>
</tr>
<tr>
<td>Prevalence</td>
<td>0.6909</td>
<td>0.11914</td>
<td>0.1370</td>
<td>0.03194</td>
<td>0.02105</td>
</tr>
</tbody>
</table>

5.2. SVM Evaluation
The SVM method could not be applied to so large a dataset, as the model did not have the robustness to accommodate it. To circumvent this liability, a smaller model was trained using the SVM architecture, which could not reasonable exceed 3000 images. This meant that the SVM model was very sensitive to the less common disease classes as there were naturally fewer of these in the entire dataset and so small a subsample could lead to some of the classes being excluded almost entirely. Therefore, the sample generated used the same percentages of each class that existed in the master solution file, and the random file generator was altered to select random members of each class in approximately equal percentages present in the original master file. This could only be an approximation to prevent inducing too much bias into the model.

Shown below in Table 7 is the confusion matrix generated for the SVM model. As shown, the SVM model performed poorly in this experiment. As the sample consisted of far fewer samples, each incorrectly classified sample had a far greater weight than the corresponding CNN model. The accuracy of the SVM model was only 75%, far lower than the CNN’s 90%. It is unclear as to whether or not this score would have been changed had it been possible to run a complete sample, however it is likely that there were un-optimised design decisions made that impacted the score. Due to the fact that there was a certain amount of feature engineering present in the SVM design process that was not the case in the CNN design, there was a possibility to add human error into the model as unfamiliarity with the process would have a far more significant impact on the SVM model over the CNN, where the features are largely extracted anonymously.

**Table 7: SVM Confusion matrix**, showing a lower accuracy rating and fewer image instances than the CNN model

<table>
<thead>
<tr>
<th>Prediction</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1770</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>94</td>
</tr>
<tr>
<td>3</td>
<td>55</td>
</tr>
<tr>
<td>4</td>
<td>38</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class: 0</th>
<th>Class: 1</th>
<th>Class: 2</th>
<th>Class: 3</th>
<th>Class: 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1770</td>
<td>98</td>
<td>106</td>
<td>169</td>
<td>156</td>
</tr>
<tr>
<td>4</td>
<td>187</td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>94</td>
<td>6</td>
<td>289</td>
<td>23</td>
<td>11</td>
</tr>
<tr>
<td>55</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>11</td>
</tr>
<tr>
<td>38</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>6</td>
</tr>
</tbody>
</table>

**Accuracy** 0.7526

**P-Value [Acc > NIR]** < 2.2e-16
Despite the fact that there was a reduced accuracy rating in the SVM model, there are a number of positive traits to the model. For example, the sensitivity of class 0 in this model was very similar to that of the CNN, suggesting they had the same ability to recognise a disease free image successfully. Furthermore, the sensitivity of the class 1 was actually higher in the SVM model at 63%, compared to 58% in the CNN model. This suggests that the SVM is better at recognising the bound between the first two classes than the CNN model. Due to the fact that the SVM is often used as a binary classifier, it makes sense that it would be adept at recognising this difference well. The issue arises when trying to recognise the other classes, which it is shown to have very poor sensitivity, probably due somewhat to the absence of a significant number of training cases. However, this could also suggest that this model is less adept than the CNN at recognising similar features than the CNN. This similarly follows what the literature suggests about CNN’s, the convolutional layers make it very adept at recognising very small differences between features (Behnke, 2003; Uetz & Behnke, 2009), a power that the SVM lacks.

Another difference in the models was in the time taken to run the training, in the case of the CNN, the training took considerably longer than the small sample implementation, taking many days to train fully. The final times taken for this model are shown in Table 8.

<table>
<thead>
<tr>
<th>Model</th>
<th>Number of Epochs</th>
<th>Time Per Epoch (min)</th>
<th>Total Training Time (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Theano</td>
<td>200</td>
<td>38.5</td>
<td>7,700</td>
</tr>
</tbody>
</table>

This total equated to roughly 5 days of training time, meaning that there was little opportunity to run this multiple times to change parameters. In contrast to this the SVM took only a number of hours to complete its training, however as it consists of less than 10% of the training instances of the CNN, it is difficult to compare them naturally. The SVM does not train in epochs in the same way as the CNN model, with this model taking a total of approx. 465 minutes to complete the classification process, which is almost 8 hours of total runtime.
5.3. Summary

In this section an overview of the how the models were evaluated was given, with figures supporting the evaluations. A thorough discussion of the merits of the CNN models followed, along with the criteria in use for establishing the optimal model. The reasons for the selection of Theano as the optimal model were discussed at length and then the comparative study turned to comparing this Theano model with the designed SVM model. The merits of both approaches were discussed, before settling for the CNN as the optimal technique due to superior results.
6. Conclusion

The research question this work sought to address was: To what extent can deep learning based neural networks statistically out-perform other machine learning techniques for image classification and under what circumstances could other machine learning techniques be considered more useful?

The objectives of the experiment were therefore to design a reasonably optimised CNN and SVM model to use as a comparative study to seek to disprove the hypotheses of the work, given as:

**H0:** The application of Deep Learning techniques will result in no performance increase over SVM approaches and it will generate no decrease in training time.

**H1:** The addition of Deep Learning and GPU computing techniques by way of Deep Convoluted Neural Networks will allow for a decrease in misclassification rate over a Support Vector Machine (SVM).

**H2:** Training time will be greatly reduced when using Deep Learning compared to a traditional CPU based approach.

The results from the model were then used as a base to address these hypotheses and to propose both further work and discussion of the merits of both approaches.

6.1. Research Objectives

The objective that this work sought to address is to critically assess whether or not an SVM could be built that would be able to match the precision and power of a deep learning system, in this case a deep learning based CNN. As seen in the implementation and results, and evaluation sections, it was possible to train an SVM to accomplish the image classifications, however at a far reduced rate. As an SVM is a resource hungry operation, it was limited in its ability to handle the extreme requirements of classifying a 55,000 strong dataset into 5 distinct classes. It was observed that in this case, the SVM was vastly outperformed by the CNN.
In the case of hypothesis H0, which is our null hypothesis, the objective was to prove that the application of a deep learning technique would improve performance over an SVM. In this case, an increased performance was judged to have been achieved by how limited the SVM was at classifying over the entire dataset. As the CNN approach was able to classify all images correctly, which the SVM required a reduced sample simply to run, irrespective of classification performance.

The first hypothesis posed was that the addition of Deep Learning and GPU computing techniques by way of Deep Convoluted Neural Networks will allow for a decrease in misclassification rate over a Support Vector Machine. The results of this are shown in the confusion matrixes developed for each implementation, shown in Tables 6 and 7. These results show that the SVM classification accuracy was significantly reduced compared to the CNN implementation. However, the SVM was not without positives in this regard. The sensitivity of the classification of class 1 images was superior in the SVM implementation, which suggests that while the more precise classifications of the less common levels was not quite as strong, the SVM was quite adept at correctly identifying early disease states, but struggled at minute class differences. The second hypothesis could not, therefore, be completely supported as there was an instance in which the SVM had some advantage. For the most part, however, the CNN had a superior accuracy, sensitivity and specificity across the majority of classes in the results. Therefore, the hypothesis is supported for the large part.

The 2nd hypothesis was Training time will be greatly reduced when using Deep Learning compared to a traditional CPU based approach. This was a more difficult hypothesis to test based on the fact that the SVM implementation was cut short by necessity. In Table X is shown the time taken to fully train the CNN network, at 7,700 minutes. Conversely the SVM took a shorter period of time, at 465 minutes. While it was impossible to make the network larger than this for an individual run, classifying all images in these short bursts would have taken approximately 8,600 minutes if we took it in bursts of 3,000 images to a total of 55,000. This would mean that the SVM was slower than the CNN in total, however it is unlikely it would have taken this amount if the full dataset could have been classified at once as the program would not have to rerun functions repeatedly during initialisation of the program and it would become more optimised as it ran. This is why it can be seen that between Table 5, and Table 8 for the CNN implementation, the time taken to run the full network is not that same time it would have taken the whole process to be run in batches.
It is likely in this case then that the CPU based approach used for this process was significantly faster when comparing the CNN results in Table 5 to the 465-minute result of the 3,000 image classification of the SVM, even bearing in mind that the CNN short models consisted of nearly double the images. However, despite this the 2nd hypothesis cannot be rejected immediately as the two different methodologies are difficult to directly compare in this manner. The solution to this issue to simple remove access to the GPU, either within the code or physically so that the model will only use the CPU. This is accomplished in framework wrappers such as Keras by feeding a JSON file into the classifier that shuts off access to Tensor, however in this case the GPU was simply disabled. However, it quickly became apparent that the process was orders of magnitude slower, with many hours being required for each epoch even on a reduced dataset. Time constraints meant that no results could be generated from this model as it could not be completed in time (training the small dataset would have taken several weeks), however it was seen that the addition of the GPU sped the process up considerably, which is supported by the literature (Catanzaro, Sundaram, & Keutzer, 2008; Ciregan, Meier, & Schmidhuber, 2012; Owens et al., 2008).

There were a number of positive take away’s from this work and also some negative points. It seemed that the CNN models were designed better, due to the volume of supporting documentation available for them recently. As CNN’s, and other Deep Learning approaches have become more and more prolific, the number of solid introductory examples has grown exponentially, therefore it was far easier to find simple examples of ways to get started with a CNN and to evolve it further, than it was with the SVM approach. Coupled with the number of image classification problems being solved in competition formats with CNN meant that it was simply an easier process to design well.

Conversely the SVM was very difficult to implement, perhaps contribution to its lower accuracy score. As deep learning has become such a popular idea in recent years, techniques that cannot leverage it as easily will perhaps be left behind in favour of those that can for this type of application. This meant that for the purposes of this work, it was difficult to find a starting point as this was not a familiar technique before starting this work.

6.2. Future Work

There are a number of ways that this work could be pushed further, some of which were already tried within the work and abandoned due to various constraints. However, there were
many other approaches that were options for this experimental design but could not be readily or quickly implemented. One example was to use specialist networks in the classification of the CNN. Specialist networks are ensemble models that, instead of being formed from a large model that encompasses all of the classes, they are made up of several models that recognise only one class. While models generally perform better as one large model. However, recently the concept of knowledge transfer is being applied to these models. In this case, a master model is trained and then knowledge transfer is used to pass this information to a number of smaller specialist networks. This idea has been shown to have had some success in increasing classification accuracy for image datasets (Romano & Schucker, 2015.). In this case, however, I decided that this would add too many layers of complexity to a time sensitive process and furthermore the small size of some classes may have rendered the effort wasted. This process could be expanded on in future.

Another method that warranted some thought was the possibility of using a hybrid CNN-SVM approach. Approaches such as this are becoming more popular as due to the binary classification power of the SVM, a large amount of the computational work of the CNN can be avoided, which would have the benefit of reducing the classification time of the model. This procedure has been shown to have had some success at recognising handwritten digits more successfully than each model alone, by using the CNN as an automated feature extractor and then using the SVM as a classifier (Niu & Suen, 2012). While this would not have been a workable approach in this experiment, a model was devised by which the SVM would be used as a binary classifier to assess whether or not the image was of a disease state and exclude all others, then the CNN would be used to classify individual levels. This is another method which could be expanded upon further in future.

Another interesting direction to take a further experiment would be to assess the performance gain or decrease from a comparison of the CNN model to a multilayer perceptron (MLP). This is another type of artificial neural network that takes advantage of backpropagation, which allows for the training of networks with many layers. MLP’s and CNN’s operate under the same general principal, however MLP’s do not contain pooling layers like the CNN. Instead the MPL is trainable via the backpropagation method. This method operates by propagating input vectors forward through the network, until it reaches the output layer. The output is then compared to the desired output and the loss function is propagated backwards. Backpropagation uses these errors to calculate the gradient descent. This gradient is then used to update the weights and the process repeats. This has been shown to be a powerful
technique, particularly due to the rise of deep learning technique (Athanasopoulos, Dimou, Mezaris, & Kompatsiaris, 2011; M. Lin, Chen, & Yan, 2013). Despite this, analysis of the performance differences between the two techniques has not be directly compared (Owens et al., 2008; Schmidhuber, 2015).

This paper has proposed that the current power of Deep Learning techniques far outmatches techniques that do not take advantage of it for these image recognition tasks, such as SVM’s. While SVM’s remain powerful and robust classifiers, it is likely that to continue being the tool of choice for image classification tasks, they will have to leverage the power of Deep Learning or will likely be replaced by more powerful models (Athanasopoulos et al., 2011; Tang, 2013).
Bibliography


Appendix 1: Confusion Matrixes and Classification Results
Theano CNN Complete Confusion Matrix and Descriptive Statistics

Confusion Matrix:

<table>
<thead>
<tr>
<th>Prediction</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>35123</td>
</tr>
<tr>
<td>1</td>
<td>1220</td>
</tr>
<tr>
<td>2</td>
<td>290</td>
</tr>
<tr>
<td>3</td>
<td>49</td>
</tr>
<tr>
<td>4</td>
<td>26</td>
</tr>
</tbody>
</table>

Overall Statistics:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>0.9064</td>
</tr>
<tr>
<td>95% CI</td>
<td>(0.9039, 0.9089)</td>
</tr>
<tr>
<td>No Information Rate</td>
<td>0.6909</td>
</tr>
<tr>
<td>P-Value [Acc &gt; NIR]</td>
<td>&lt; 2.2e-16</td>
</tr>
<tr>
<td>Kappa</td>
<td>0.7968</td>
</tr>
<tr>
<td>McNemar's Test P-Value</td>
<td>&lt; 2.2e-16</td>
</tr>
</tbody>
</table>

Statistics by Class:

<table>
<thead>
<tr>
<th></th>
<th>Class: 0</th>
<th>Class: 1</th>
<th>Class: 2</th>
<th>Class: 3</th>
<th>Class: 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensitivity</td>
<td>0.9336</td>
<td>0.5023</td>
<td>0.9262</td>
<td>0.65809</td>
<td>0.92199</td>
</tr>
<tr>
<td>Specificity</td>
<td>0.8112</td>
<td>0.98794</td>
<td>0.9770</td>
<td>0.99830</td>
<td>0.99683</td>
</tr>
<tr>
<td>Pos Pred Value</td>
<td>0.9209</td>
<td>0.84875</td>
<td>0.8648</td>
<td>0.92751</td>
<td>0.86235</td>
</tr>
<tr>
<td>Neg Pred Value</td>
<td>0.9567</td>
<td>0.93596</td>
<td>0.9881</td>
<td>0.98883</td>
<td>0.99832</td>
</tr>
<tr>
<td>Prevalence</td>
<td>0.6909</td>
<td>0.11914</td>
<td>0.1370</td>
<td>0.03194</td>
<td>0.02105</td>
</tr>
<tr>
<td>Detection Rate</td>
<td>0.6795</td>
<td>0.05960</td>
<td>0.1269</td>
<td>0.02102</td>
<td>0.01941</td>
</tr>
<tr>
<td>Detection Prevalence</td>
<td>0.7379</td>
<td>0.07022</td>
<td>0.1467</td>
<td>0.02266</td>
<td>0.02251</td>
</tr>
<tr>
<td>Balanced Accuracy</td>
<td>0.8974</td>
<td>0.74409</td>
<td>0.9516</td>
<td>0.82820</td>
<td>0.95941</td>
</tr>
</tbody>
</table>
CNN classification results
Results shortened to make addition to document possible

<table>
<thead>
<tr>
<th>Image</th>
<th>Actual</th>
<th>Predicted</th>
</tr>
</thead>
<tbody>
<tr>
<td>1_left</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1_right</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2_left</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2_right</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3_left</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3_right</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4_left</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4_right</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>5_left</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5_right</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6_left</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6_right</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7_left</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7_right</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8_left</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>8_right</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>9_left</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>9_right</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>11_left</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11_right</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>12_left</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>12_right</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>44322_left</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>44322_right</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>44324_left</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>44324_right</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>44326_left</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>44326_right</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>44329_left</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>44329_right</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>44332_left</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>44332_right</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>44333_left</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>44333_right</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>44335_left</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>44335_right</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>44336_left</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>44336_right</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
SVM Complete Confusion Matrix and Descriptive Statistics

Confusion Matrix:

<table>
<thead>
<tr>
<th>Prediction</th>
<th>Reference</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>1770</td>
<td>98</td>
<td>106</td>
<td>169</td>
<td>156</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>187</td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>94</td>
<td>6</td>
<td>289</td>
<td>23</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>55</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>38</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>6</td>
</tr>
</tbody>
</table>

Overall Statistics:

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>0.7526</td>
</tr>
<tr>
<td>95% CI</td>
<td>(0.7367, 0.7679)</td>
</tr>
<tr>
<td>No Information Rate</td>
<td>0.6539</td>
</tr>
<tr>
<td>P-Value [Acc &gt; NIR]</td>
<td>&lt; 2.2e-16</td>
</tr>
<tr>
<td>Kappa</td>
<td>0.4869</td>
</tr>
<tr>
<td>McNemar's Test P-Value</td>
<td>NA</td>
</tr>
</tbody>
</table>

Statistics by Class:

<table>
<thead>
<tr>
<th></th>
<th>Class: 0</th>
<th>Class: 1</th>
<th>Class: 2</th>
<th>Class: 3</th>
<th>Class: 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensitivity</td>
<td>0.9026</td>
<td>0.63823</td>
<td>0.70316</td>
<td>0.026178</td>
<td>0.041958</td>
</tr>
<tr>
<td>Specificity</td>
<td>0.5424</td>
<td>0.99409</td>
<td>0.9770</td>
<td>0.94822</td>
<td>0.982143</td>
</tr>
<tr>
<td>Pos Pred Value</td>
<td>0.7884</td>
<td>0.92118</td>
<td>0.68322</td>
<td>0.070423</td>
<td>0.105263</td>
</tr>
<tr>
<td>Neg Pred Value</td>
<td>0.7467</td>
<td>0.96209</td>
<td>0.95264</td>
<td>0.936475</td>
<td>0.953433</td>
</tr>
<tr>
<td>Prevalence</td>
<td>0.6539</td>
<td>0.09770</td>
<td>0.13705</td>
<td>0.063688</td>
<td>0.047683</td>
</tr>
<tr>
<td>Detection Rate</td>
<td>0.5902</td>
<td>0.06235</td>
<td>0.09637</td>
<td>0.001667</td>
<td>0.002001</td>
</tr>
<tr>
<td>Detection Prevalence</td>
<td>0.7486</td>
<td>0.06769</td>
<td>0.14105</td>
<td>0.023675</td>
<td>0.019006</td>
</tr>
<tr>
<td>Balanced Accuracy</td>
<td>0.7225</td>
<td>0.81616</td>
<td>0.82569</td>
<td>0.501337</td>
<td>0.512050</td>
</tr>
</tbody>
</table>
### SVM Classification Results

Results shortened to make addition to document possible

<table>
<thead>
<tr>
<th>Image</th>
<th>Actual</th>
<th>Predicted</th>
</tr>
</thead>
<tbody>
<tr>
<td>9987_left</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9946_left</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>993_right</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>9923_right</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9922_right</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9890_right</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9880_right</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>9877_left</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9867_left</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>9842_left</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9837_right</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9836_right</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>9834_left</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>982_left</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9791_right</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>9771_left</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>9735_right</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9727_right</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>97_right</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9654_left</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>9645_right</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9634_left</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9596_right</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9588_right</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9582_left</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>9576_left</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>9407_right</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>9385_right</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>938_right</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>9361_left</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9360_right</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9338_left</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9332_right</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9330_left</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>9328_right</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>9304_right</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>9303_left</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>9296_left</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Appendix 2: Python Model Codes

Image Conversion Code

code snippet
cropped = img.crop(bbox)
resized = cropped.resize([crop_size, crop_size])
return resized

def get_convert_fname(fname, extension, directory, convert_directory):
    return fname.replace('jpeg', extension).replace(directory, convert_directory)

def process(args):
    fun, arg = args
    directory, convert_directory, fname, crop_size, extension = arg
    convert_fname = get_convert_fname(fname, extension, directory, convert_directory)
    if not os.path.exists(convert_fname):
        img = fun(fname, crop_size)
        save(img, convert_fname)

def save(img, fname):
    img.save(fname, quality=97)

@click.command()
@click.option('--directory', default='data/train', show_default=True,
    help="Directory with original images.")
@click.option('--convert_directory', default='data/train_res', show_default=True,
    help="Where to save converted images.")
@click.option('--test', is_flag=True, default=False, show_default=True,
    help="Convert images one by one and examine them on screen.")
@click.option('--crop_size', default=256, show_default=True,
    help="Size of converted images.")
@click.option('--extension', default='tiff', show_default=True,
    help="Filetype of converted images.")
def main(directory, convert_directory, test, crop_size, extension):
    try:
        os.mkdir(convert_directory)
    except OSError:
        pass

file_names = [os.path.join(dp, f) for dp, dn, fn in os.walk(directory)
              for f in fn if f.endswith('jpeg') or f.endswith('tiff')]
file_names = sorted(file_names)

if test:
    name = data.get_names(file_names)
    y = data.get_labels(name)
    for f, level in zip(file_names, y):
        if level == 1:
            try:
                img = convert(f, crop_size)
                img.show()
                Image.open(f).show()
                real_raw_input = vars(__builtins__).get('raw_input',input)
                real_raw_input('enter for next')
                except KeyboardInterrupt:
                    exit(0)

    print("Resizing images in {} to {}, this can be a long process.".
".format(directory, convert_directory))

n = len(file_names)
# the process is carried out in batches
b_size = 500
num_batch = n // b_size + 1
pool = Pool(Num_Proc)

args = []

for files in filenames:
    args.append((convert, (directory, convert_directory, f, crop_size, extension)))

for i in range(num_batch):
    print("batch number {:>2} / {}".format(i + 1, num_batch))
    pool.map(process, args[i * batchsize: (i + 1) * b_size])

pool.close()

print('done')

if __name__ == '__main__':
    main()

Also viewable on personal Github at

Theano (Lasagne) Neural Net Training Script

"""Script for fitting neural net to a training model."""
import click
import numpy as np

import data
import util
from nn import create_net

@click.command()
@click.option('--cnf', default='configs/c_512_4x4_32.py', show_default=True,
               help='Path to the config module.')
@click.option('--weights_from', default=None, show_default=True,
               help='Path to the weights file')
def main(cnf, weights_from):

    config = util.load_module(cnf).config

    if weights_from is None:
        weights_from = config.weights_file
    else:
        weights_from = str(weights_from)

    files = data.get_image_files(config.get('train_dir'))
    file_names = data.get_names(files)
    file_labels = data.get_labels(names).astype(np.float32)

    net = create_net(config)

    try:
        net.load_params_from(weights_from)
        print("using weights from {}".format(weights_from))
    except IOError:
        print("error loading weights from file")

    print("fitting model ...")
    net.fit(file_names, file_labels)

if __name__ == '__main__':
    main()
Theano Transform File

```python
from __future__ import division
import time
import click
import numpy as np
import nn
import data
import tta
import util

config = util.load_module(cnf).config

runs = {}
if train:
    runs['train'] = config.get('train_dir')
if test or test_dir:
    runs['test'] = test_dir or config.get('test_dir')

net = nn.create_net(config)

if weights_from is None:
    net.load_params_from(config.weights_file)
    print("weights loaded from \{}").format(config.weights_file))
else:
    weights_from = str(weights_from)
    net.load_params_from(weights_from)
    print("weights loaded from \{}\).format(weights_from))

if n_iter > 1:
    tfs, color_vecs = tta.build_quasirandom_transforms(
        n_iter, skip=skip, color_sigma=config.cnf['sigma'],
        **config.cnf['aug_params'])
else:
    tfs, color_vecs = tta.build_quasirandom_transforms(
        n_iter, skip=skip, color_sigma=0.0,
        **data.no_augmentation_params)

for run, directory in sorted(runs.items(), reverse=True):
    print("extracting features for files in \{}\).format(directory))
    tic = time.time()
    image_files = data.get_image_files(directory)

    Xs, Xs2 = None, None
    for i, (tf, color_vec) in enumerate(zip(tfs, color_vecs), start=1):
        print("{} transform iter \{}\).format(run, i))
        X = net.transform(image_files, transform=tf, color_vec=color_vec)
        if Xs is None:
            Xs = X
            Xs2 = X**2
        else:
            Xs += X
            Xs2 += X**2
```

print('took {:6.1f} seconds'.format(time.time() - tic))
if i % 5 == 0 or n_iter < 5:
    std = np.sqrt((Xs2 - Xs**2 / i) / (i - 1))
    config.save_features(Xs / i, i, skip=skip,
        test=True if run == 'test' else False)
    config.save_std(std, i, skip=skip,
        test=True if run == 'test' else False)
    print('saved {} iterations'.format(i))

if __name__ == '__main__':
    transform()
Theano Blend Script (Part)

""" This program is used to blend features extracted with the Conv Nets and make the prediction"
from __future__ import division, print_function
from datetime import datetime
from glob import glob

import click
import numpy as np
import pandas as pd
import theano
from lasagne import init
from lasagne.updates import adam
from lasagne.nonlinearities import rectify
from lasagne.layers import DenseLayer, InputLayer, FeaturePoolLayer
from nolearn.lasagne import BatchIterator
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import StandardScaler
import yaml
import data
import nn
import util

class blend_net(nn.Net):
    def __init__(self, files, labels):
        pass
    def split(self, X, y):
        return X, y

class Resample_iterator(BatchIterator):
    def __init__(self, batch_size, resample_prob=0.2, shuffle_prob=0.5):
        pass
    def __iter__(self):
        pass
Xb = self.X[sl]
if self.y is not None:
    yb = self.y[sl]
else:
    yb = None
yield self.transform(Xb, yb)

def get_estimation(n_features, files, labels, eval_size=0.1):
    layers = [
        (InputLayer, {'shape': (None, n_features)}),
        (DenseLayer, {'num_units': N_HIDDEN_1, 'nonlinearity': rectify,
                      'W': init.Orthogonal('relu'),
                      'b': init.Constant(0.01)}),
        (FeaturePoolLayer, {'pool_size': 2}),
        (DenseLayer, {'num_units': N_HIDDEN_2, 'nonlinearity': rectify,
                      'W': init.Orthogonal('relu'),
                      'b': init.Constant(0.01)}),
        (FeaturePoolLayer, {'pool_size': 2}),
        (DenseLayer, {'num_units': 1, 'nonlinearity': None}),
    ]
    args = dict(
        update=adam,
        update_learning_rate=theano.shared(util.float32(START_LR)),
        batch_iterator_train=Resample_iterator(BATCH_SIZE),
        batch_iterator_test=BatchIterator(BATCH_SIZE),
        objective=nn.get_objective(l1=L1, l2=L2),
        eval_size=eval_size,
        custom_score=('kappa', util.kappa) if eval_size > 0.0 else None,
        on_epoch_finished=[nn.Schedule('update_learning_rate', SCHEDULE),
        ],
        regression=True,
        max_epochs=N_ITER,
        verbose=1,
    )
    net = blend_net(layers, **args)
    net.set_split(files, labels)
    return net

Also viewable on personal Github at

SVM Model Generation Code

```python
# Run SVM on retinal images
from sklearn import svm
from matplotlib.mlab import PCA as mlabPCA
import numpy as np
import os
from scipy.misc import imsave,imread
from sklearn.grid_search import GridSearchCV
from datetime import datetime
import cPickle
from sklearn.cross_validation import ShuffleSplit

# Define the local directory used
def load_subset(images):
    images_labels = []
    path_to_images = images + images # Image directory here
    labels_file = images + '.txt' # Path to label file
    images_labels = []
    with open(labels_file, 'r') as f:
        dict_labels = dict([line.strip().split() for line in f.readlines()])
        # List files in this directory
        files = os.listdir(path_to_images)
        # Create image holding structure
        images = np.zeros((len(files), 256*256*3), dtype=np.uint8)
        img_labels = np.zeros(len(files), dtype=np.uint8)
        for fid, file in enumerate(files):
            image = imread(path_to_images + '/' + file)
            if image.shape == (256, 256, 3):
                images[fid] = image.flatten()
                img_labels[fid] = int(dict_labels[file])

    return images, img_labels,

pca = RandomizedPCA(n_components=10, whiten=True)

param_grid = {'C': [1e3, 5e3, 1e4, 5e4, 1e5],
              'gamma': [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.1]}
clf = GridSearchCV(SVC(kernel='rbf', class_weight='auto'), param_grid)

def kappa(labels, predictions):
    image_labels = np.asarray(labels)
    image_predictions = np.asarray(predictions)

    model_ratings = np.matrix((image_labels, image_predictions)).T
    categories = int(np.amax(model_ratings)) + 1
    subjects = model_ratings.size / 2

    # Build weight matrix
    weighted = np.empty((categories, categories))
    for i in range(categories):
        for j in range(categories):
            weighted[i, j] = abs(i - j) ** 2
```

76
# Build observed matrix
observed = np.zeros((categories, categories))
distributions = np.zeros((categories, 2))
for k in range(subjects):
    observed[ratings[k, 0], ratings[k, 1]] += 1
    distributions[ratings[k, 0], 0] += 1
    distributions[ratings[k, 1], 1] += 1

# Normalize observed and distribution arrays
observed = observed / subjects
distributions = distributions / subjects

# Build expected array
expected = np.empty((categories, categories))
for i in range(categories):
    for j in range(categories):
        expected[i, j] = distributions[i, 0] * distributions[j, 1]

# Calculate kappa
kappa = 1.0 - (sum(sum(weighted * observed)) / sum(sum(weighted * expected)) )
return kappa

train_images, train_labels, train_files = load_subset('train')

param_grid = {'C':[1], 'gamma':[0.0001],}
clf = GridSearchCV(svm.SVC(kernel='rbf', class_weight='auto'), param_grid, n_jobs=1, cv=ShuffleSplit(test_size=0.20, n_iter=1, random_state=0, n=len(train_images))
clf = clf.fit(train_images, train_labels)

with open('', 'w') as f: # Storage location
    cPickle.dump(clf, f)
print 'Model saved as ' # File Name

Full SVM code can be found on personal GitHub at

Supplementary codes

```python
import random
import shutil

# Define variables
total_files = 100
sample_size = 5.45

# Add image Directory
orig_directory = ""
sample_directory = "/"

# Initialize list of files
files = []

# Generate list of files
for i in range(1, total_files + 1):
    files.append(str(i) + "\.tif")

# Select random sample
selected_files = random.sample(files, sample_size)

# Move selected files to sample directory
for file in selected_files:
    shutil.move(orig_directory + file[0], sample_directory + file[0])
    shutil.move(orig_directory + file[1], sample_directory + file[1])
```
Appendix 3: Lua Train Example Code

```lua
require 'optim'
require 'xlua'

--[[
  a number of steps involved in this process
  1. initialisation of SGD optimization state and also learning rate schedule
  2. Creation of loggers.
  3. train - this function begins the training loop, i.e. load data, train model, save model etc.
  4. trainBatch - Used by the train function to train a single batch after the data is loaded.
]]--

-- Setup a reused optimization state
optimState = {
  learningRate = opt.LR,
  learningRateDecay = 0.0,
  momentum = opt.momentum,
  dampening = 0.0,
  nesterov = true,
  weightDecay = opt.weightDecay
}
if opt.optimState ~= 'none' then
  assert(paths.filep(opt.optimState), 'File not found: ' .. opt.optimState)
  print('Loading optimState from file: ' .. opt.optimState)
  optimState = torch.load(opt.optimState)
end

local optimator = nn.Optim(model, optimState)

-- Learning rate schedule. A new optimizer for will be built per epoch
-- Follows 55-epoch training.
--
-- Return values:
--  diff to apply to optimState,

local regime = {
  --nEpoch, LR, WD,
  --{ 2, 1e-3, 1e-4 },
  { 30, 3e-3, 0 },
  { 30, 1e-3, 0 },
  { 1000, 1e-4, 0 }
}
print( "training regime")
print( regime)

local function paramsForEpoch(epoch)
  if opt.LR ~= 0.0 then
    return { }
  end
end
```

79
if opt.regime and opt.regime ~= "none" then
  regime = dofile(opt.regime)
end

local cnt = 0
for _, row in ipairs(regime) do
  cnt = cnt + row[1]
  if epoch <= cnt then
  end
end

-- 2. Create loggers.
trainLogger = optim.Logger(paths.concat(opt.save, 'train.log'))
local batchNumber
local top1_epoch, loss_epoch
local tError
local confusion
local cnt = 0
local tickSize = math.floor(opt.epochSize/20)

-- 3. Train
function train(dataset)
  print('==> doing epoch on training data:')
  print("==> online epoch # " .. epoch)
  confusion = optim.ConfusionMatrix(dataset:getNumberClass())
  confusion:zero()
tError = 0
cnt = 0

  local params, newRegime = paramsForEpoch(epoch)
  print (params)
optimator:setParameters(params)
  if newRegime then
    optimator = nn.Optim(model, optimState)
  end
  batchNumber = 0
cutorch.synchronize()
  model:training()
  -- model:cuda() -- get it back on the right GPUs.
  local tm = torch.Timer()
top1_epoch = 0
loss_epoch = 0
for i=1,opt.epochSize do
  donkeys:addjob(
    function(tid)
      local tid = tid or __threadid
      local inputs, labels = dataset:getBatch(tid)
collectgarbage()
    end
  )
return sendTensor(inputs), sendTensor(labels), tid
end,

function (inputs, labels, threadid)
    trainBatch(inputs, labels)
    dataset:refillId(threadid)
end)
end
donkeys:synchronize()
cutorch.synchronize()

learningRate = learningRate * opt.lr

print(string.format('Epoch: [%d][TRAINING SUMMARY] Total Time(s): %.2f
    .. average loss (per batch): %.2f
    .. accuracy(%%): top-1 %.2f
    epoch, tm:time().real, loss_epoch, top1_epoch))
print('
')
sanitize(model)
collectgarbage()

confusion:updataValid(
print(confusion)
local kappa = computeKappa(confusion.mat)
print ("train kappa is ". kappa)
print()
local avgError = math.sqrt(tError/opt.epochSize/opt.batchSize)
print (avgError)
config:trainError = avgError
local msg = {
    ['Error (train set)'] = avgError .. " ", ..params.learningRate.. " ", ..kappa
print (msg)
trainLogger:add(msg)
end -
local inputsCPU = torch.FloatTensor()
local labelsCPU = torch.FloatTensor()

local inputs = torch.CudaTensor()
local labels = torch.CudaTensor()

local timer = torch.Timer()
local dataTimer = torch.Timer()

function trainBatch(inputsThread, labelsThread)
cutorch.synchronize()
collectgarbage()
local dataLoadingTime = dataTimer:time().real
  timer:reset()
  receiveTensor(inputsThread, inputsCPU)
  receiveTensor(labelsThread, labelsCPU)

  inputs:resize(inputsCPU:size()):copy(inputsCPU)
  labels:resize(labelsCPU:size()):copy(labelsCPU)

local err, outputs = optimator:optimize(
  optim.sgd,
  inputs,
  labels,
  criterion)

cutorch.synchronize()
batchNumber = batchNumber + 1
loss_epoch = loss_epoch + err
tError = tError + err*inputs:size(1)
recordConfusion(confusion, outputs, labelsCPU)

if cnt%tickSize == 1 then
  xlua.progress(cnt, opt.epochSize)
end
  cnt = cnt +1
  collectgarbage()

dataTimer:reset()
end

Full Lua Torch code available on Github at ()