University College London

Masters Thesis

Eye2Gene: Classifying inherited retinal disease with deep learning

Author:
Ross Ilott

Supervisors:
Prof. Daniel Alexander
Dr. Nikolas Pontikos

This report is submitted as part requirement for the MSc Degree in Computer Graphics, Vision & Imaging, at University College London. It is substantially the result of my own work except where explicitly indicated in the text. The report may be freely copied and distributed provided the source is explicitly acknowledged.

September, 2020

Abstract

Inherited Retinal Diseases (IRDs) are a group of genetic conditions which causes progressive and bilateral deterioration of the retina. IRDs are a leading cause of blindness, with it being estimated that 1 in 3000 people in the UK have an IRD. Some IRD patients may be blind from birth, while others may have their vision deteriorate gradually over time. Genetic diagnosis and identification of the genetic mutations causing IRDs is a prerequisite to management and treatment. However a large percentage of cases remain undiagnosed due to insufficient knowledge, expertise, expense, or access to appropriate facilities. In this thesis I have created a system which applies machine learning techniques to retinal scan images in order to assist doctors in their diagnosis. Using a convolutional neural network (CNN), I have shown that it is possible to achieve 100% accuracy on a small dataset and very high levels of accuracy (above 95%) on a larger dataset when predicting the name of the gene responsible for diseases seen in these images, while reducing the time and cost needed to make such a diagnosis compared to manual inspection of the data. Access to this system is provided by the way of a website, Eye2Gene, which was created by researchers at Moorfields Eye Hospital and modified to accept image inputs for processing by the CNN. Network and deployment code can be found here.

Acknowledgements

Thanks to everyone involved in this project, especially to Dr Nikolas Pontikos for his guidance and expertise, and Moorfields Eye Hospital for providing the resources and materials that made this project possible.

A non-exhaustive list of thanks also goes to:

- Prof Daniel Alexander
- Prof Andrew Webster
- Dr Kaoru Fujinami

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The Problem

Inherited retinal diseases (IRD) are a leading cause of blindness in the UK working age population [1] and are currently difficult to diagnose quickly and efficiently [2]. This is because there are relatively few IRD clinical experts and relatively few centres where these patients can be diagnosed and managed. Additionally diagnosis is a manual process that is subject to human error when not performed by an experienced expert. As a result this means that around 40% patients are undiagnosed or are incorrectly diagnosed [3],[4] and the financial and time cost are very high which can result in a delay to diagnosis.

1.2 Objectives

The aim of this project is to develop an automated system that can classify retinal scans of patients depending on the mutation of the gene that has caused the IRD they have. To begin with I will only be attempting to detect the difference between two genes, 'ABCA4' and 'USH2A', based on images of the retina taken using Autofluorescence (AF) imaging, with potential to include other gene types and image modalities if there is sufficient data available. This system would hopefully benefit people in the following ways:

- Increase the speed of diagnosis
  The analysis of the data will be performed by a computer program, which can process much faster and more deterministically than a human.
- Increase the accuracy of diagnosis
  There is no human element to the classification of the image, removing any human error from the system.
- Decrease the cost of diagnosis
  Running costs for computers are generally lower than for that of humans, with additional improvements in cloud technology facilitating even cheaper CPU/GPU time.

While harder to quantify, an automated system could also strive to reduce stress and make the process of disease diagnosis easier for patients, but this is more of a by-product of automation rather than a goal of this project.

The shape that this system will take is in the form of a website where the user can upload images of retinal scans, with a pre-trained deep learning model running behind it that classifies the image based upon the visible effects of the gene mutation present. The details of how this system works is the basis for this paper.

Chapter 2

Background

To understand what is trying to be achieved, one must first know a bit about the underlying biology and how it is represented in the data. The background section briefly covers these areas before moving on to the subject of deep learning, which constitutes the core of the project and so receives the most attention.

2.1 Genetics

Humans have 23 pairs of chromosomes, each part of the pair comes from a different parent. Chromosomes are made of DNA, which contain regions called genes, where each gene controls the generation of a protein. These genes are transcribed into RNA which is then translated into a protein. Proteins are fundamental elements used by the body to perform all kinds of functions. Mutations in genes that produce these proteins can cause disease[5].

Genetic mutations that occur in gamete cells are passed on to offspring, depending on the location and type of mutation in the DNA of the gamete cell, there can be various consequences. The mutation might cause a faulty protein to be produced, or stop a protein from being produced altogether. This can result in visual impairment in the offspring.

One way of working out which gene is causing a retinal disease in a patient is to sequence their DNA, and see if there are any mutations in the regions that are known to affect the eyes. The sequenced DNA is compared to the reference human genome[6] to find any discrepancies. This requires a blood or saliva sample to be taken from the patient and sent to a lab for sequencing.

Mutations in over 250 genes are associated with IRDs[7]. These genes are then screened for IRD-causing mutations. The two genes I will be attempting to classify are both fundamental to the cause of IRDs, with ABCA4 being responsible for Stargardt Disease (STGD)[8] and USH2A having been identified as associated to Usher syndrome[9].

2.2 Retinal Imaging

Another way of detecting IRDs is via images of the back of the eye (retinal/fundus imaging). This is an appealing method of detection as it can be performed non-invasively, quickly and doesn't require a skilled technician to process the results. There are different types of imaging available, but for the purposes of this project there are 3 main types worth commenting on - Autofluorescence (AF), Infrared (IR) and Optical Coherence Tomography (OCT).

2.2.1 Autofluorescence Imaging

AF imaging works by shining a particular wavelength of light into the retina (usually blue short-wavelength), and collecting the photons that are re-emitted by special fluorescent cells called retinal pigment epithelium (RPE). The RPE cells contain special pigment molecules called lipofuscin, which is what produces the light. Initially this emission of light from lipofuscin was considered noise, but eventually new methods of analysing the light revealed that the intensity of the emitted light can inform us about the health and presence of cells in the tissue[10].

Over time, the amount of lipofuscin in RPE cells can accumulate. The amount of accumulation can also be used to infer age as well as the state of other diseases and potential problems[11]. Places where there is excess accumulations will appear hyperfluorescent whereas areas where the RPE cells have died will appear darker and are called hypofluourescent, however the amount of lipofuscin isn't always the only contributor to the emitted light, with photooxidation of lipofuscin contributing to the amount as well[12].

2.2.2 Infrared Imaging

IR imaging operates in a similar way to AF imaging in that it uses particular wavelengths of light to determine properties of the tissue[13]. In this case infrared light is shone into the eye and the reflections recorded. IR is mostly used for inspecting blood flow in the retina. IR and AF imaging have similarities but they each highlight different tissues within the eye.

2.2.3 Optical Coherence Tomography

Optical Coherence Tomography[14] (OCT) is one of the most recent advances in retinal imaging and is used widely in the field today. It builds upon the principals of collecting reflected light using a technique called interferometry. The optical path length of received photons is recorded and allows rejection of any that have

scattered multiple times after reflection while accepting photons from areas of interest.

The main benefits of OCT include much higher resolution of images, it is non-invasive and uses non-harmful wavelengths of EM waves. Another interesting aspect of OCT is that one can construct 3D images using different wavelengths of light that penetrate deeper through certain tissues and therefore highlights the different cellular layers in the retina (of which there are approximately 8)[15]. Generally this is the preferred way of capturing high resolution retinal scans today[14].

2.3 Deep Learning

Deep learning is an area of artificial intelligence (AI) research that specifically looks at how computers can gain **deeper** insights into data comparable to human knowledge. Programs and research in this area specialise in learning to identify features or distinguish classes of objects with minimal human input.

2.3.1 Neural Networks

The term "neural network" came about as a consequence of efforts to create computational models of how the brain works[16]. A neural network in computing is not realistically how the brain works, but is inspired by it. The origins of this area of computer science started in the 1940s and went by the name of cybernetics, with some of the first models being developed such as the Perceptron[17], which was the first model which could learn weights for defined categories given inputs from each category and ADALINE[18]. These simple models were state of the art for their time, but they were not without their flaws, for example they were linear models which can famously not learn the XOR function. This drew a lot of criticism at the time and was partly responsible for the decline in popularity of this area of research.

In the 1980s neural networks increased in popularity again, but under a different name, connectionism. There were several key concepts that were developed during this movement, one of them being distributed representation[19] - the idea that inputs can be broken down into individual features, and neurons can learn each individual feature, e.g instead of a neuron for a red car, you would have a neuron for red, and a neuron for car. In this case the red neuron could then learn from anything red. Another big success in this age of neural networks was the use of the backpropagation algorithm to train neural networks, which has now become the primary algorithm used in training today[20][21]. The wave of research progressed into the 90s as well, with various advances being made such as the LSTM[22], but

by the end of the decade the performance of neural networks did not meet their ambitious expectations, resulting in another decline in their popularity.

The current resurgence of interest in neural networks began in the mid 2000s with a breakthrough in training (e.g greedy layer-wise pretraining [23]). This made it possible to train much deeper networks than was possible before, and a lot of attention was attracted to the concept of deeper networks. This increase in model size has been possible due to advances in technology, both hardware and software, to enable faster training and processing of networks with larger amounts of neurons. Faster CPUs and the use of GPUs have enabled use of larger and larger networks, with the general software used for distributed computing also increasing in quality. Dataset sizes to train these networks on have also rapidly increased in size. The amount of digital data available in the 21st century has enabled performance of neural networks to produce useful results, whereas in decades previously this was not possible. Having large amounts of images and data available means that less skill is required in crafting the network, and just feeding in more data allows the network to be more proficient.

This area of deep learning is currently providing some very useful results in several scientific areas, with applications in a wide range of fields spanning physics, medicine, security, economics.

Convolutional Neural Networks

The subtype of a neural network called a convolutional neural network (CNN) [24] is a type which is specialised at using grids of data as inputs. Images can be thought of as 2D grids, so it is possible to feed them in to this type of network as an input. They are particularly well suited to identifying and classifying objects in images, and are also used in many other fields like predicting how molecules interact [25], searching for exotic subatomic particles [26] and creating 3D mappings of the human brain [27]. I will be looking at the case where a CNN is being fed an image as input, and attempting to classify it according to visible effects of gene mutations seen within it.

A CNN is comprised of layers, each layer performs some kind of extraction of information or operation to the image, and then passes that information on to the next layer. The shape and size of the output of each layer normally decreases each time, eventually providing you with size of output you desire, e.g cat,dog. This is a grossly oversimplified explanation and the reality is much more complex. To start understanding how a CNN works, it might be prudent to begin with the components involved.

Convolution

One of the main operations performed on images by a CNN is convolution - the process of applying a filter or kernel to a square area of the image, and recording the value it produces. This convolution is normally applied as a layer of a network, with each neuron operating on a kernel sized area of the image. The purpose of this kind of operation is to extract information about features of the image, depending on the type of kernel you apply and the size, you can detect different properties of the image, e.g edge detection.

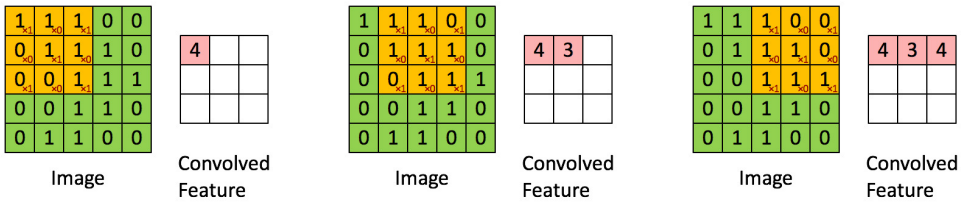

FIGURE 1: Information extraction using convolutional kernels[1]

Pooling

Pooling is an operation that can be performed to an output of a layer of the network to make the value invariant to small disturbances/noise in the input. This is normally applied after any convolutional layers have been applied. This is particularly useful for images because it avoids any large changes in the output of the network due to noise in the image in the form of translation or orientation. A common type of pooling layer is maximum pooling[28], which just returns the maximum value within an area of a convolved image. There are other methods of pooling such as weighted or unweighted average pooling and normalised pooling using the L2 norm.

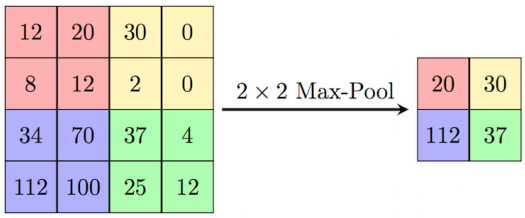

FIGURE 2: Max pooling operation[2]

[1] http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

Fully connected layer

The final layer of a CNN usually consists of a fully connected (FC) layer with each neuron being mapped to a desired output. Some networks include multiple fully connected layers at the end to increase the number of linear combinations that can be created, but evidence has found that these extra FC layers can be removed without compromising performance[29].

Dropout layer

A problem that a CNN can have is overfitting. This is when the network achieves high accuracy on its training data, but when given any previously unseen data performs very badly, i.e does not generalise. This can be due to training too specifically on some features of the input data. One method of reducing this overfitting effect is to introduce 'dropout'[30] layers that deactivate neurons randomly with a given probability. This random chance of deactivation happens with each iteration of training, creating some variation in the learning process.

ReLU

The Rectified Linear Unit (ReLU)[31] is a relatively recent improvement made to the activation function of neural networks. The use of this kind of unit introduces non-linearity to the network, and improves the speed that a network can learn compared to previous activation functions such as sigmoid and tanh (which were the default before) by reducing the vanishing gradient problem.

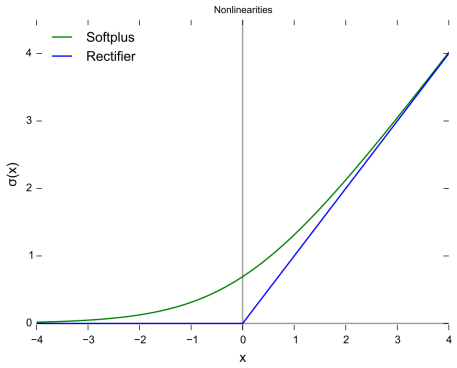

FIGURE 3: The ReLU activation, which only activates once the input is positive[3]

[2] https://computersciencewiki.org/index.php/Max-pooling_/_Pooling
[3] https://en.wikipedia.org/wiki/Rectifier_(neural_networks)

Chapter 3

Related Work

Now we move onto related works which are directly relevant to my project. This can be divided into two sections, the first comprising of similar publications which have the tackled the same problem of predicting genes based on retinal images, the second covering notable CNN architectures which are of interest and which might be applicable to my problem.

3.1 Papers

Two papers were found which apply deep learning techniques to identify genes in IRDs.

3.1.1 Fujinama-Yokokawa et al.

Fujinama-Yokokawa et al.[2] document a data-driven approach to identifying IRDs. A four-class classifier was trained on SD-OCT 178 images from 75 individuals, to distinguish between ABCA4, RP1L1, EYS and normals which produced a mean overall test accuracy of 90.9%.

Some issues were identified, such as overfitting on some types of gene due to lack of genetic diversity and the small size of the dataset, leaving room for improvement. A commercial software was used to train a neural network on the data, called MedicMind[32], which also provides the predictions when new data is input. The website for the product does not state how the network is constructed or trained, which means it's difficult to compare it's effectiveness / architecture to other networks I have been reviewing. That being said, it is hard to glean much information about CNNs from the paper as a whole, apart from the fact that classifying based upon gene is at least possible and can be done with high accuracy above 90% using software that is already available. Some preprocessing was done to the images such as cropping, which may be worth investigation.

3.1.2 Shah et al.

Using the same kind of imagery (SD-OCT), Shah et al[33] use two different CNN models to classify the severity of Stargardt disease (STGD) ranging between normal, mild and severe. They collected 749 images for their network to train and test on, 102 with a normal retina, and 647 with STGD. Instead of just classifying their images as either having STGD or not, they propose using their three grade system to allow more descriptive and useful analyses of the images. To increase the size of their dataset, they use a few different types of image augmentation. Firstly they apply rotations of up to 30 degrees, reflections in the y-axis and translations, in such a way as to "preserve clinical relevance". Secondly they chop the image into columns 32 pixels wide, called 'profile analysis', and feed each column into the network as a separate input, with some additional vertical translation. This second augmentation increases the dataset 30-fold.

One of their CNNs was VGG19[29], pretrained using the ImageNet weights, and using each OCT image cropped to a 256x256 size. Their second CNN was an architecture of their own devising, similar to LeNet[34], with approx. 3 million parameters. This network included batch normalisation[35] and dropout[30] layers.

Their results are extremely high, with a reported accuracy of 99.6% for VGG19 and 85.3% for their own architecture when used as a binary classifier to distinguish between normal and diseased images (when using aggregated weighted probabilities for whole OCT scans they report 98.0% accuracy). They also cite the size of their dataset as a limiting factor, but are hopeful that one day a publicly available standard test dataset will be available.

Some comments are also made about the previous paper by Fujinami-Yokokawa et al[2], where they suggest some of the error experienced may be due to the network learning differences in camera properties that were used to take the OCT images, and suggest using one single type of camera. They also say that MedicMind [32] uses the Inception-v3 architecture that has been reviewed further up in this document, I was not able to find any reference to this network on the MedicMind website.

3.2 Network Architectures

Neural network architecture is an extremely active area of research, with new creative and innovative ideas being regularly published. One of the breeding grounds for these new networks has been the ImageNet Large Scale Visual Recognition Challenge (ILSVRC)[36], whose competitors have consisted of many of today's most famous and successful CNNs.

3.2.1 AlexNet

AlexNet is a CNN that was designed by Alex Krizhevsky and competed in ILSVRC 2012 [37]. One variant of their network architecture won the top-5 error rate (where a successfull classification is when one of the network's top 5 predicted classes was the correct class) category by a large margin, opening the door to other CNNs that would come after it. The AlexNet used a combination of convolution pooling and fully connected layers, resulting in a network with 8 learned layers, but more importantly they were one of the first to use Rectified Linear Units (ReLU) as the activation function for each neuron, which considerably speeded up their training time. They contrast this ReLU activation to the traditional sigmoid and tanh methods, showing that their function reaches 25% training error in a shorter time. Even with this increase in training speed, it took them between 5 to 6 days to train the network on the ImageNet dataset of over 15 million images. The main problem they had with the architecture was overfitting, a problem which they overcame with a combination of data augmentation and dropout layers. These anti-overfitting measures reportedly resulted in double the number of iterations of training required to converge on the best performance. Overall the AlexNet showed that depth can be very powerful in improving CNN performance.

3.2.2 VGG16/19

A few years after ILSVRC2012 came ILSVRC2014, and with it several new architectures that once again pushed the performance of CNNs even further. One such network came from the Visual Geometry Group (VGG) group at Oxford University in the form of 'VGG16'[29]. This network focused on improving on the AlexNet architecture by increasing the number of convolutional layers, making the network deeper. This increase in layers while decreasing the convolutional layer's size to very small amounts (to keep the network small enough to be feasibly trained) meant that this network came top in several of the ILSVRC2014 categories. They also show in the paper that by stacking these small convolutional layers you can still capture larger features, as secondary conv. layers use the outputs of other conv. layers. Training speed for this deeper type of network was negatively affected, with reported training times on the ImageNet dataset being 2-3 weeks, depending on the variant.

Some interesting methods that were applied to the training process of VGG include modifying the learning rate of the network at different layers, and initialising the weights of deep variants with pretrained weights of simpler variants of the network at different layers. These might be aspects worth exploring in my experiments however they may be outweighed by other aspects, like data cleaning or augmentation.

VGG16/19, the number referring to the number of layers, produced the best results for the competition, and so warrants some investigation in terms of how they perform on my data set.

3.2.3 GoogLeNet/Inception-v1

GoogLeNet is the name of another network which was submitted to ILSVRC2014[36], and which outperformed VGG16 in some categories.

The paper[38] states that the performance gains were obtained while maintaining the same network complexity. The architecture itself is codenamed 'Inception', with one implementation with 22 layers being called 'GoogLeNet'.

The origins of this architecture are founded in LeNet-5[34], but with several improvements made. The use of series of fixed gabor filters in similar networks has been replaced here with learned filters, which are repeated many times, leading to the 22 layer design. The use of network-in-network[39] to increase representational power in 1x1 convolutions enables the layers to server 2 purposes - to decrease dimensionality, but to also remove computational bottlenecks by increasing width without performance penalty.

The impetus behind the development of this network was not due to any new datasets or technological progression, but due to new research and new ideas in the field such as the R-CNN algorithm by Girshick et al[40]. This network uses 12x fewer parameters than VGG16/19 from 2 years previous in ILSVRC2012, while achieving better accuracy by making use of an 'Inception module'. In addition the need to increase the computational efficiency of the network was also cited as a key motivator in keeping the number of layers and complexity of the network to a minimum.

[38] continues by talking about the high level considerations of dense and sparse implementations of CNNs. The traditional method of increasing the number of layers of the network and providing more training data can only get us so far before it becomes unfeasable to process the network on today's technology, not to mention the problem of overfitting. An interesting solution to this problem might be the use of sparse matrices in the computation of the layers. If a lot of the weights in a convolutional layer are zero, then a lot of computation time has been wasted - if this could be represented sparsely, then perhaps a lot of time could be saved. A problem with this solution is at the moment a lot of software libraries have become very efficient at dense matrix calculations, not sparse, so much so that it would actually be more inefficient to use sparse matrices in today's infrastructure.

The inception module itself consists of a kind of aggregation in-between convolutional layers. The outputs of a convolutional layer are grouped into these clusters, with different types of convolution being applied, as well as spatial reduction in the form of pooling where required to keep the computational cost low.

The network itself is then constructed of mostly inception modules, with pooling layers used to decrease the size of the grid. The team found that using inception modules only in the lower layers was better for memory efficiency reasons during training - although they admit that perhaps this shouldn't be required.

Another interesting structural change is the addition of something called an auxiliary classifier. This is where the output of an intermediate layer is applied to the output of one of the inception modules in the middle of the network and their loss during training is added to the final loss of the network (weighted by a special amount, in this case 0.3 was used). The effect of this is that discrimination is encouraged in the lower stages of the network, as a lot of discrimination is then picked up on at this stage.

Inception-v2/v3

In 2015 a new paper was published that documents a revision so to speak of the inception architecture[41]. These changes result in what the authors call 'Inception-v2' and 'Inception-v3'. They speak of a few high-level design principles:

- Avoid large decreases in representation size

  Compressing the size of the data at any single layer discards data that might be useful, so changes in the size of the representation should happen gradually throughout the network.

- Increase the activations in higher dimensional representations

  This increases the number of features that can be detected separately from each other, and also increases the speed at which the network can be trained.

- Reduce the size of the representation before convolving

  e.g a 3x3 convolution can still capture features of an image that has been reduced spatially without expecting much loss of information.

- Balance network width and depth

  Increasing both number of filters in a layer (width) and number of layers (depth) in parallel can contribute towards optimal performance.

An interesting modification that was made to the architecture was the decomposition of medium to large convolutional layers (12 to 20 in area) into 1xn and nx1 convolutional layers. This can be seen to be more efficient if you look at a 5x5 convolution - which can be decomposed into a This factorisation results in much decreased computational cost without loss of any information.

As a result of these modifications to an inception module, the new architecture 'Inception-v2' was proposed. It contains factorised convolutional layers based on

the above idea, combined with some traditional inception modules. Even though the network is 42 layers deep, it is only 2.5 times more computationally expensive than the original GoogLeNet, and still more efficient than VGG19/16.

With a few more modifications again, 'Inception-v3' was created. This network is basically Inception-v2 with some additions - namely auxiliary classifiers (used in GoogLeNet/Inception-v1) and batch normalisation[35]. Batch Normalisation is an interesting concept that tries to eliminate problems like vanishing gradients, and reduce the dependence of gradients on the scale or initial value of the parameters. These upgrades to the v2 version of the network resulted in another performance improvement on the ILSVRC2012 classification benchmark, with a top-1 error rate of just 4.2%.

Inception-v4

'Inception-v4' is the latest and final variant so far of the inception architecture. The best result on ILSVRC2012 was this time performed with an ensemble of 4 models, 3 residual and one Inception-v4. To begin with we should define what a residual network or ResNet is.

Resnet The basic concept behind ResNet, or, deeply residual learning[42], is the residual learning unit which at it's core is a shortcut connection from the input to the output, where it is combined with the output of the stacked layers.

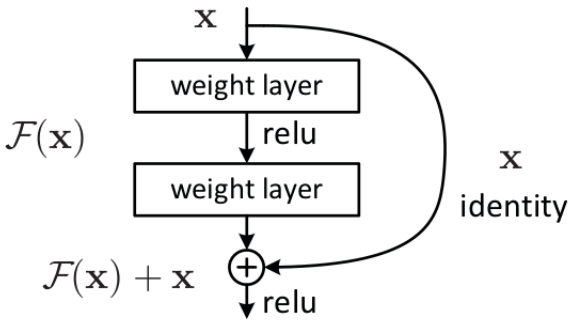

FIGURE 4: ResNet module

The aim of this modification is to avoid the degradation that occurs with very deep neural networks in the form of vanishing gradients or amplified gradients (small changes in high layers which get amplified greatly when the network has many layers). Using a modified version of the VGG family of networks, the paper[42] shows that their method improves the training error value. Interestingly

they experimented with very large networks of up to 1000 layers, and managed to achieve <0.1% training error (with 7.93% test error) on the CIFAR-10 dataset.

Inception-v4 attempts to take advantage of the unique properties of residual networks and integrate them into the inception module architecture. New variants of the inception module are used, that make use of residual mapping to combine the original input value into the output. This turned out to be a very successful technique, which once again produced another improvement on the top-1 and top-5 error rate of ILSVRC2012. The name given to this residual modified network is Inception-ResNet-v2 (there were 2 ResNet versions).

Chapter 4

Method

This section documents the data curation aspect and the development of a suitable network architecture. To begin with the available data needs to be considered. In particular its suitability for training and whether any further preprocessing is necessary. Once the data is ready I will discuss the process of choosing an appropriate network architecture, and possible augmentations and sensible values of hyperparameters.

4.1 Data preparation

The initial dataset (Dataset 1) I was provided with consisted of 567 retinal scans taken using autofluorescence imaging. These images were grouped by label, of which there are two: 'ABCA4' and 'USH2A', referring to the mutated gene responsible for the effect in the associated image. The dataset was further pre-split into training and testing portions with 425 and 142 images in each category respectively, representing a 75/25 split of training and testing data. This may not be an ideal ratio, and can be considered another hyper-parameter to be experimented with.

Label	Training	Testing	Total
ABCA4	276	85	361
USH2A	149	57	206
	425	**142**	**567**

Table 1: Number of images in original dataset

4.1.1 Sources

Dataset 1 was provided by Moorfields Eye Hospital (Prof Andrew Webster). Additional dataset of 87 images was provided by Dr Kaoru Fujinami (Dataset 2). Once this dataset was merged with Dataset 1, the totals were as follows:

Label	Total	Added	New Total
ABCA4	361	81	424
USH2A	206	6	212

TABLE 2: Number of images after addition of Dataset 2

A much larger dataset from Moorfields Eye Hospital was also provided at a later date (Dataset 3), containing many tens of thousands of images labelled by gene, of which there were 136 different classes including ABCA4 and USH2A. The majority of experiments had already been conducted at the point that Dataset 3 was found, so was not merged with Datasets 1 and 2. Experiments performed in later sections of this document are labelled with the dataset used. Some of these new classes may be of potential interest to extend the classifier into. Once the data had been sorted appropriately, the number of examples can be seen in the table below.

Label	Total
ABCA4	24632
USH2A	7536
RPGR	6840
BEST1	4792
CNGB3	4104
PRPH2	3805
CHM	3503
RS1	3473
RP1	2750
CNGA3	2589

TABLE 3: Number of images in Dataset 3, top 10 genes

4.1.2 Data Cleaning

Some thought is required before feeding these images into a network to be trained on. A lot of the previously shown architectures require specific sizes of image as inputs to their first layer. The size of Dataset 1 images are mostly 256x256, with some irregularities, which will need to be normalised before they can be used.

There appear to be two types of image in my data, each taken with a different field of view (narrow and wide). The effect of this camera setting means that the images with a wider field of view have a black ring around them, whereas the smaller field of view images do not. This could be a potential issue as a neural

network could start to associate this black ring with certain labels, or otherwise learn some correlation between them, giving incorrect results.

FIGURE 5: Examples of images captured with different field of view settings. *Left:* Small, *Right:* Large

Dataset 1 had already been inspected and low quality images had been removed, leaving little else to check for, Dataset 2 contained relatively few images, and so it was easy to spot any obviously invalid examples, which were removed.

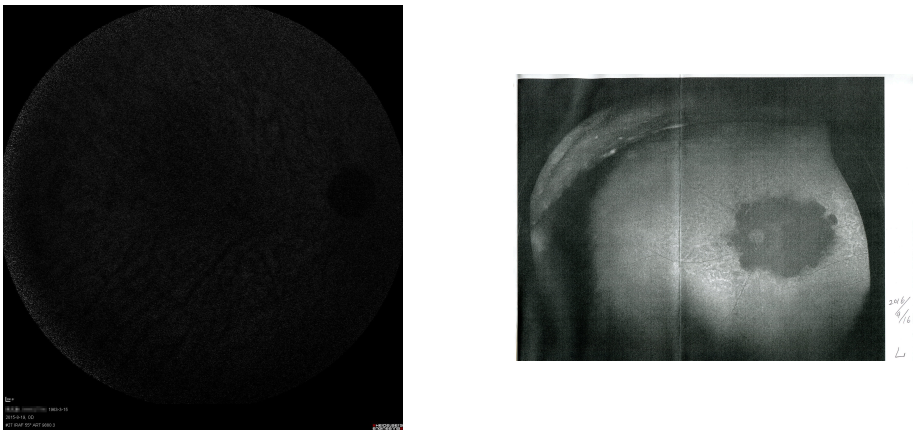

FIGURE 6: Examples of obviously bad images from Dataset 2

Dataset 3 on the other hand, contains enough images that it would be impractical to manually inspect each and every image. Some statistical analysis can be performed to identify possibly bad images and programmatically remove them. Looking at some of the images there were obvious candidates for removal, with various types of problems. Firstly, the dimensions of the image can be checked to see if they are equal. Setting a threshold of 100 for the difference between

the values of the two dimensions removes any images that are not mostly square. Some images may not be exactly square, which is tolerable and won't affect the information in the image.

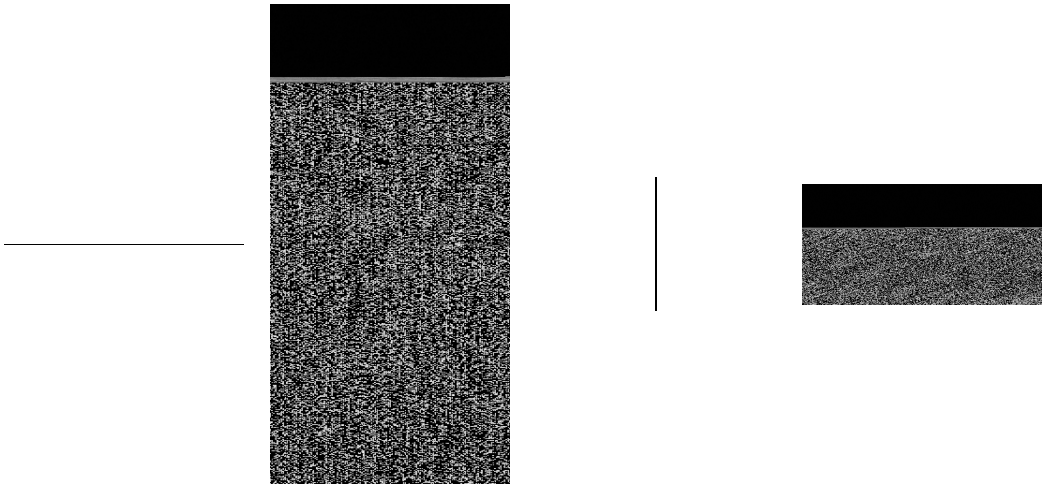

FIGURE 7: Examples of images with invalid shapes from Dataset 3

Secondly the sum of the pixel intensities can be checked, to see if there is a relationship that means more invalid images can be excluded. The distribution of intensity values of Dataset 3 images can be seen below:

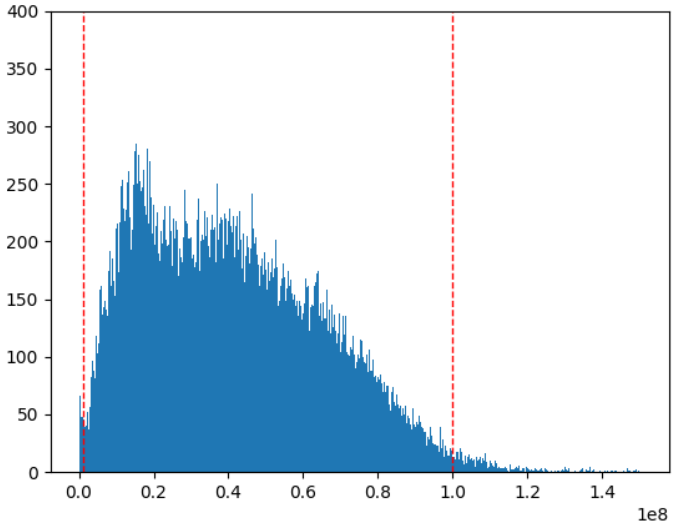

FIGURE 8: Histogram of sum of intensity values per image in Dataset 3 with thresholds of 1e+6 and 1e+8 displayed as dashed red lines

By applying a simple threshold to this histogram, it was possible to remove the majority of bad images. Good values for thresholds appeared to be 1e+6 for dark images and 1e+8 for bright images.

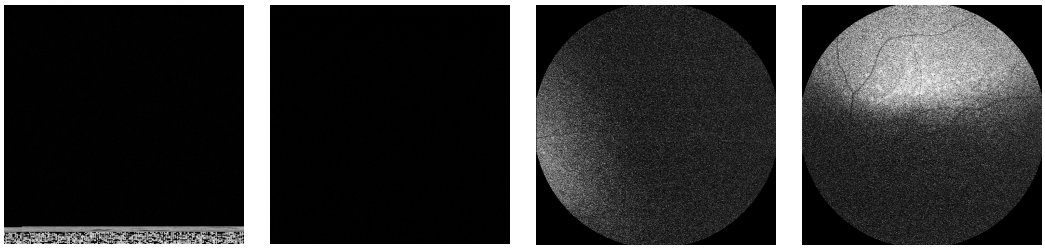

FIGURE 9: Examples of dark images from Dataset 3

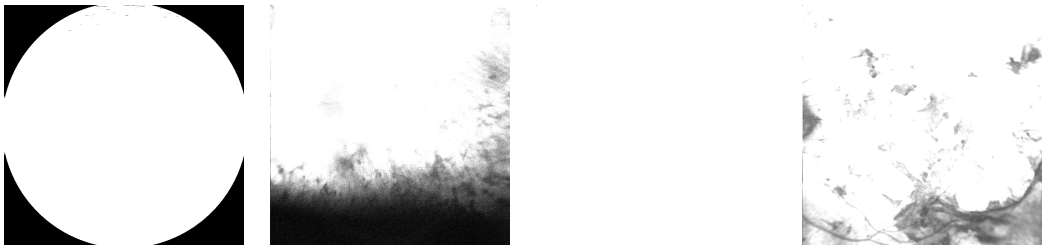

FIGURE 10: Examples of bright images from Dataset 3

Above are some visual examples of 'bad' images which are either too dark or too bright.

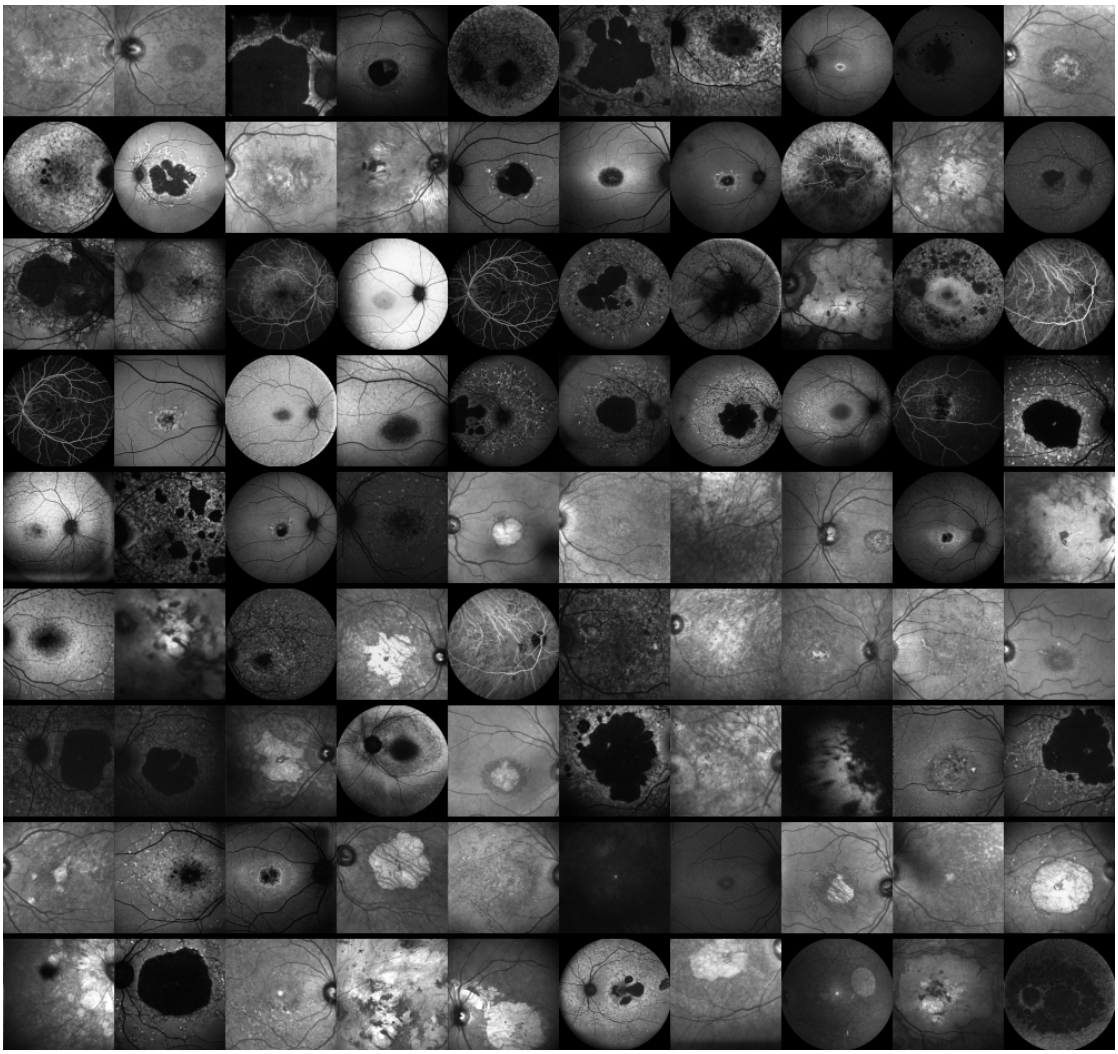

FIGURE 11: A montage of some random ABCA4 images which remained after the cleaning process

In Figure 11 we can see that there are no longer any invalid shaped or completely dark images. The revised totals of Dataset 3 can be seen in the below table.

Label	Total
ABCA4	18335
USH2A	5802

TABLE 4: Number of images in Dataset 3 after cleaning

4.1.3 Augmentation

Success in deep learning tasks has been in part due to increases in available data resulting in datasets of millions of images[43]. As Dataset 1 in this instance is comparatively much smaller, problems may be encountered with overfitting or low accuracy simply due to not having enough examples to learn from. To combat this effect a technique known as data augmentation can be used to increase the amount of available data points to learn from.

Data augmentation involves applying transformations to existing data to generate similar data that can be treated as new data points. There are various different types of transformation, and depending on the data some will be more relevant than others. For example on our dataset of retinal images it might not make sense to apply certain translation or rotation based transformations, as they would produce unrealistic results, but instead transformations in brightness and contrast.

In my experiments I attempted a combination of different augmentations chosen from the available options provided by Keras' ImageDataGenerator object. These mainly consisted of mirroring and rotational changes. Data also needs to be rescaled appropriately for neural networks, with pixel values rescaled to between 0 and 1, and in the case of Inception-based models -127 to 127. In some cases Keras comes with appropriate pre-processing functions which can be selected when instantiating the model, in other cases it needs to be done manually.

4.2 Architectures

Using the Keras framework it is very easy to download and instantiate well-known neural networks, so attempting to fit an instance of these was the easiest task to begin with. In addition to these architectures I can design my own in a similar fashion and fit it in the same way. The below table shows the architectures that were considered, and some of their attributes.

A variety of networks were considered, with these being some of the top performing in ILSVRC[36], and of varying depths and structures. All of these network implementations are included with Keras/Tensorflow. Also included in the table is a custom network of my own design (named Custom 1).

Name	Parameters (millions)	Depth	Ref.
Custom 1	3.7	5	
VGG16	138.3	23	[37]
InceptionV3	23.8	159	[41]
InceptionResnetV2	55.8	572	[44]

TABLE 5: Network Architecture information

VGG16, InceptionV3 and InceptionResnetV2 have been discussed in the background section of this document, so I will not go into detail again here, suffice to say they are all top performing networks that came about as a result of the ILSVRC throughout the last decade. VGG16 is a more straightforward design of convolution and pooling layers, whereas the newer Inception architecture implements more advanced features such as links back to previous layers and the 'Inception' module structure. It's worth discussing the Custom 1 network in more detail, below is a summary of a constructed Custom 1 network when built with Keras:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 254, 254, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 127, 127, 32)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 125, 125, 32)      9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 62, 62, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 60, 60, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 30, 30, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 57600)             0
_________________________________________________________________
dense (Dense)                (None, 64)                3686464
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 130
=================================================================
Total params: 3,715,234
Trainable params: 3,715,234
Non-trainable params: 0
_________________________________________________________________
```

FIGURE 12: Summary of Custom 1 network

Custom 1 was designed as a simple convolutional network with inspiration taken from VGG16. The top layers are composed of convolution and pooling pair

layers which are then flattened into one dimension, before passing through a single densely connected layer to allow all linear combinations to be used, and finally through another dense layer to produce the number of class outputs we want (in this case we have two classes). The dropout layer is positioned in the summary towards the end, but is in fact applied to all neurons throughout the network. You could in fact add many more convolutional and pooling layers, making the network very similar to the VGG family of networks, but Custom 1's aim is to see what results can be obtained with a relatively shallow network. This is to asssess what kind of effect network capacity has on learning.

4.3 Parameters

Once a network architecture has been selected, there are many different hyperparameters which need to be set. The tuning of these values can greatly affect the final accuracy of your network, so care needs to be taken when setting them. Finding good hyperparameters can be a bit of an art form, with no defined formula for finding the correct values, but many guidelines on how to achieve good accuracy on your data and some best practices to follow that will help you find good values.

During the training of each of my example networks, I varied these hyperparameters to see what the effect of these would be and recorded their effect on training accuracy.

4.3.1 Batch size

When an image has passed through a CNN, the output is compared against the desired output and the weights and biases are modified relative to the difference through backpropagation. The number of images which pass through the network before the weights are updated is known as the batch size. The larger your batch size, the more data the network sees before an update, meaning the update will have a better chance of moving along the error gradient in the right direction and improve your accuracy. Another way of thinking about this is that batch size is like the resolution of the error surface. A higher resolution surface means you'll follow it more closely, a larger value will allow bigger strides over the surface potentially allowing your network to converge quicker. Popular choices for this value are 32, 64 and 128, but any value can be used up to the size of your training dataset. Different values were selected and each network was trained with each value to compare their effect on training accuracy.

If your data is quite varied, a small batch size may not be large enough to fit all the different examples of data required to make a step in the correct direction.

Increasing the batch size in this case allows you to make weight updates using more data, giving you a better informed gradient descent direction.

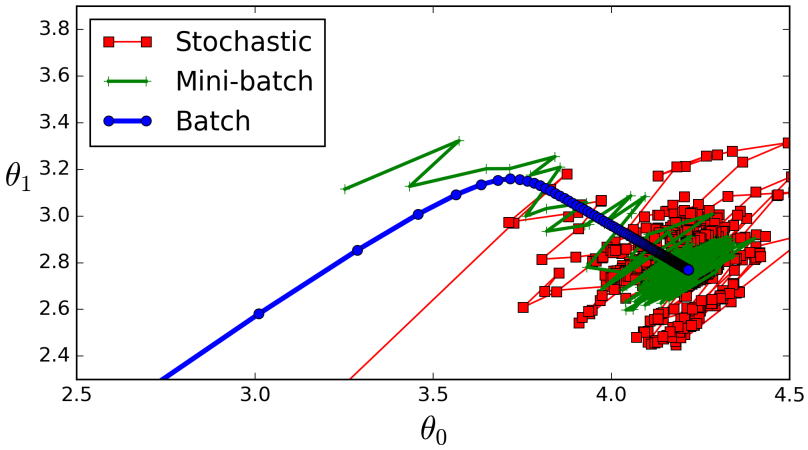

FIGURE 13: Effect of batch size on reaching global minimum[1]

The batch size also has a more practical implication, it can affect the speed that you can train. A larger batch size will require more resources as more images are loaded at once, so you could also be physically restricted on a maximum value depending on your hardware.

4.3.2 Epoch

An epoch has passed when the entire dataset has been processed by the network once. The number of epochs that the network runs for can be any length, ideally as few as possible to achieve your desired accuracy. In some initial experiments the accuracy of the network had mostly settled by around the 50th epoch, so that seemed like a good value to begin my experiments with. It is beneficial to keep the number of epochs as low as possible to reduce training time, but there is a risk that by doing so the network's accuracy will not have enough time to reach it's true potential.

[1] https://medium.com/@harshita.vemula/comparison-of-algorithms-for-linear-regression-105e405a4f15

4.3.3 Learning rate

The learning rate affects the size of the update after each iteration. It can be described as α in the following formula:

$$\theta_1 = \theta - \alpha \frac{\partial}{\partial \theta_1} f(\theta_1) \tag{4.1}$$

where θ is the current weight, θ_1 is the new weight, and $\frac{\partial}{\partial \theta_1} f(\theta_1)$ is the calculated gradient of the loss function. In this way the learning rate value controls the size of the step. Picking a learning rate too large can result in the algorithm never being able to reach the global minimum, constantly stepping over it, whereas conversely if the learning rate is too small, it can take a prohibitively expensive number of iterations to reach it.

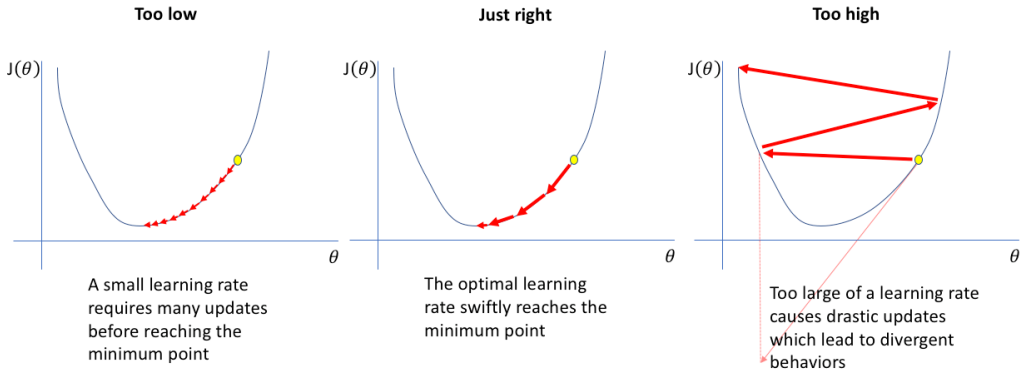

FIGURE 14: Learning rate step size effect during gradient descent[2]

To combat this problem many different methods of altering the learning rate dynamically have been developed. Mainly these consist of modifying the learning rate after each epoch of the algorithm either as a function of the epoch or a cyclical approach.

Using a polynomial based scheduler which outputs a learning rate at each epoch as a function of a integer parameter and epoch number, it is possible to create a variable learning rate which decreases in size with increasing epoch numbers.

[2] https://www.jeremyjordan.me/nn-learning-rate/

$$d(e, e_{max}, p) = \left(1 - \frac{e}{e_{max}}\right)^p$$
$$\alpha = i \cdot d(e, e_{max}, p)$$

FIGURE 15: Polynomial Learning Rate Scheduler

The final α calculation can be seen in Figure 15, where e is the current epoch value, e_{max} is the maximum epoch value, d is the resultant decay coefficient and i is the initial learning rate value.

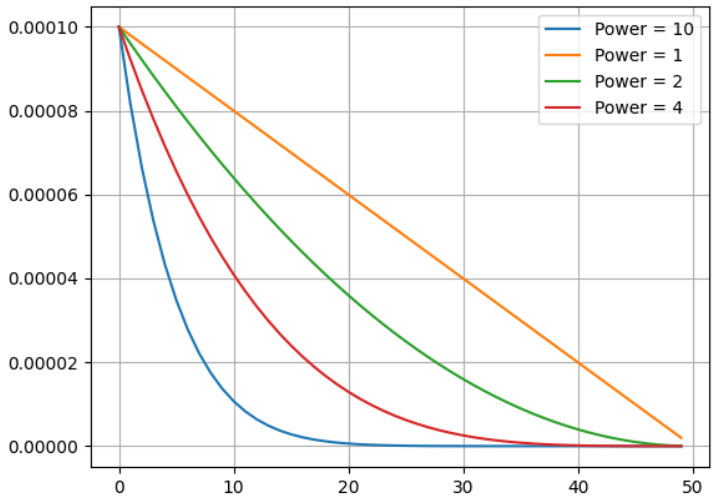

FIGURE 16: Comparison of learning rate decay rates as a function of epoch

Using a polynomial learning rate proved beneficial in this project, with the p value of the polynomial function being set to low integer values proving to be the most effective. The effects of various p values can be seen in Figure 16.

4.3.4 Dropout

As mentioned previously, including a dropout layer in the model can be an effective way to combat overfitting. In general an aggressive dropout probability of 0.7 was selected due to the small size of the dataset. This should help to stop reliance on any single example too heavily, but can be altered.

4.3.5 Optimiser

The optimiser is the part of the network responsible for calculating the gradient descent direction, or in other words minimising the loss function. There are various different types to choose from, each performing gradient descent differently. The latest and most popular optimiser at the moment is Adam[45], which uses stochastic gradient descent with adaptive estimation of first and second-order moments.

4.3.6 Data splitting

Intuitively the more data that is fed into a network, the better the expected accuracy. But how will you know if that accuracy is correct? To solve this problem the dataset can be split into three sections:

Training The data that is fed into your network and learned from

Validation Unseen data that the network predicts classes of at the end of each epoch. This gives us feedback on how to tune the hyperparameters to improve accuracy.

Testing A completely unseen set of data that the fully trained network attempts to classify.

The ratios of these three classes of data can affect the final accuracy of the network. It can be a trade-off between having more confidence in your answer, and better accuracy. Taking too much data away from the training set can have negative effects on performance, so the balance has to be carefully tuned. In general a dataset is split into training and testing first, then the training dataset is split to create a training and validation set. Common ratios to use might be 90/10, where the 90 is then split 70/30, resulting in a 63/27/10 (train/val/test) split.

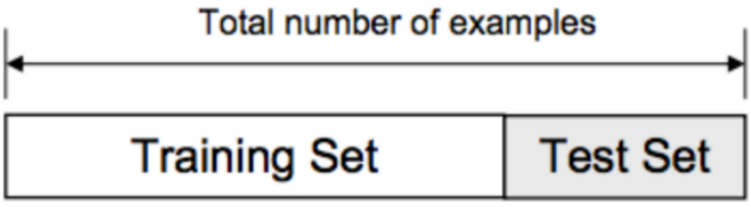

FIGURE 17: Test/Train split diagram

As mentioned in the data preparation section, Dataset 1 was already split into two categories - train and test. To increase the amount of training data these two directories were instead used as training and validation sets, with no final testing data. Dataset 3 was obtained later which could serve as a massive testing dataset.

4.3.7 Class weighting

Dataset 1 has a severe class imbalance, so providing a weighting to each class during training can make the network learn more from examples that are rarer. This should mean that features in under-represented classes are still learnt from. The weighting is automatically calculated by counting the number of examples of each class, setting one class to a weight of 1, and then setting other classes relative to that e.g if we had half as many examples of one class, it's class weight would be 2.

4.3.8 Initial parameters

Some initial experimentation was done to find hyperparameters which produced good results. These values were found via rapid trial and error experiments with a small Custom 1 network which took a very small amount of time to run for 50 epochs on Dataset 1. Good starting parameters for Dataset 1 were selected as follows:

Parameter	Value
Learning rate	1e-3
Batch size	32
Dropout	0.7

TABLE 6: Initial hyperparameters for Dataset 1

For working with neural networks the Tensorflow-based Keras library[46] was used, which is written in Python. This library allows rapid prototyping and comes with lots of built in functions for managing datasets, as well as ready-to-go networks with downloadable weights from training on the ImageNet dataset[36] (where applicable).

4.4 Data loading

Data was loaded using one of Keras' utility functions 'flow_from_directory' which allows loading images in batches instead of all at once at training time. This is

particularly necessary when you have many thousands of images that do not fit into main memory, and while it is not needed for Dataset 1, if any more data is procured it would be useful to have.

The recommended way to perform this batch loading can be seen below:

```
datagen = ImageDataGenerator(preprocess_func=preprocess)

train_gen = datagen.flow_from_directory(
    directory,
    target_size=(256,256),
    batch_size=32,
    class_mode='categorical',
    classes=self.classes,
)
val_gen = datagen.flow_from_directory(
    directory,
    target_size=(256,256),
    batch_size=32,
    class_mode='categorical',
    classes=self.classes,
)
```

Another benefit of this batch loading mechanism is that it allows augmentation to happen at the same time. Each batch has the augmentations applied before being passed on as inputs to the network, which is again more memory and storage space efficient than pre-applying the modifications (which for a large dataset would require a lot more hardware to store). It's worth noting here that this Keras function only returns the augmented versions of the images, so the network never actually receives the original images[46].

4.5 Network

Beginning with the VGG16[29] architecture, models could be instantiated with the following code:

```
model = VGG16(include_top=False, input_shape=(256,256,3))
```

A generalised version of this code can be made by wrapping it inside a class object, which allows creating an instance of it from a simple script, and supplying parameters from the command line, making running experiments much easier.

```
class VGG16(object):
    def __init__(self, model_config):
        for k, v in model_config.items():
            setattr(self, k, v)

        self.model = keras_VGG16(
            include_top=False,
            input_shape=self.input_shape + (3,)
        )
```

The same functions can be used across all types of network, so creating an inheritance-based class structure makes sense. Generalising this code further into a base class provides a way to easily add more architectures without rewriting the same code again.

```
class Model(object):
    def __init__(self, model_config):
        for k,v in model_config.items():
            setattr(self, k, v)

class VGG16(Model):
    def __init__(self, model_config):
        super(VGG16, self).__init__(model_config)
        self.name = 'VGG16'
        self.model = keras_VGG16(
            include_top=False,
            input_shape=self.input_shape + (3,)
        )
```

Once the model is created, the optimiser needs to be selected, and then the final object 'compiled' into a model that can be trained. This is done via the 'fit' method, which performs the process of fitting the data to the model.

```
optimizer = Adam(lr=1e-4)

model.compile(
    loss='categorical_crossentropy',
    metrics=['accuracy'],
    optimizer=optimizer,
)
```

```
model.fit(
    train_gen,
    epochs=50,
    validation_data=val_gen,
    class_weight=class_weights,
    steps_per_epoch=train_gen.samples // batch_size,
    validation_steps=val_gen.samples // batch_size,
)
```

4.6 Deployment

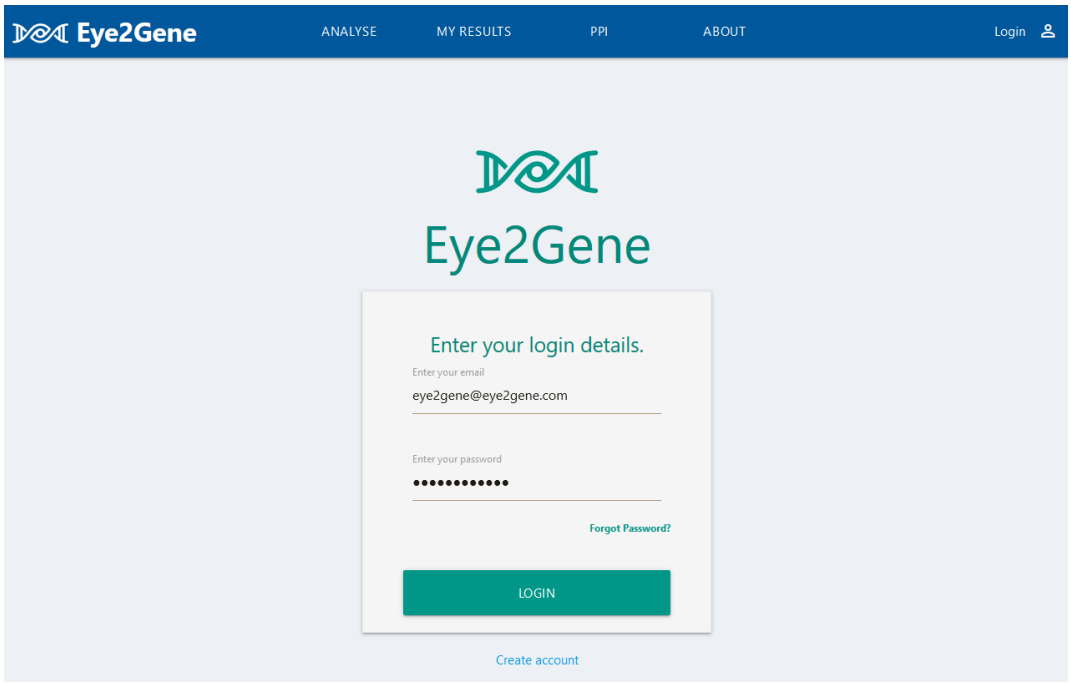

FIGURE 18: Eye2Gene homepage

The final model was deployed to the cloud using Amazon Web Services (AWS)[47]. A small website was created in Ruby which was provided as the frontend for the system, which was adapted to send uploaded user images to the model for the label to be inferred. The website contained basic login functionality as well as image upload capabilities. Modifications were made to convert uploaded images into a base64 string, which could then be passed via HTTP to an AWS 'Lambda' endpoint, which returns a JSON string containing the results from the model. 'Lambda' is a service provided by AWS which allows the user to just upload the

code for their application to Amazon and pay just for the execution of said code. Amazon handles the rest of the infrastructure below the code (hardware, operating system etc). This makes it much easier and cheaper to run an application if it is small enough to be run in AWS Lambda (there are strict size limits on the code and libraries that can be uploaded). The user doesn't interact with AWS Lambda directly, but via the Ruby website, which is also running in AWS but as a standalone webserver on a EC2 instance (Amazon's name for a Virtual Machine).

Figure 20 shows the components and logical flow of the application, which can be summarised briefly:

- User logs in to website
- User uploads images
- Images are converted by Website to base64 strings and sent to AWS Lambda
- AWS Lambda converts base64 string back into an image
- AWS Lambda loads pre-trained neural network and predicts label for image
- Website receives response from AWS Lambda containing prediction
- Website displays prediction to user

All of the above steps should be completed within a reasonable timeframe to provide a good user experience. By making use of Tensorflow Lite[48], the final model can be converted into a more compact version that can be loaded by a Tensorflow Lite interpreter inside the AWS Lambda function. When tested with an Inception-ResnetV2 model, the HTTP and Lambda phases returned a prediction within 10 seconds. This could be tested with a small python script that performs the HTTP requests manually and records the time taken, the output of which can be seen below.

```
Time taken: 8.32 seconds
{
    "label": "USH2A",
    "message": "Classified as USH2A",
    "data": [
        4.546242493574937e-08,
        1.0
    ]
}
```

FIGURE 19: Output of AWS Lambda testing script

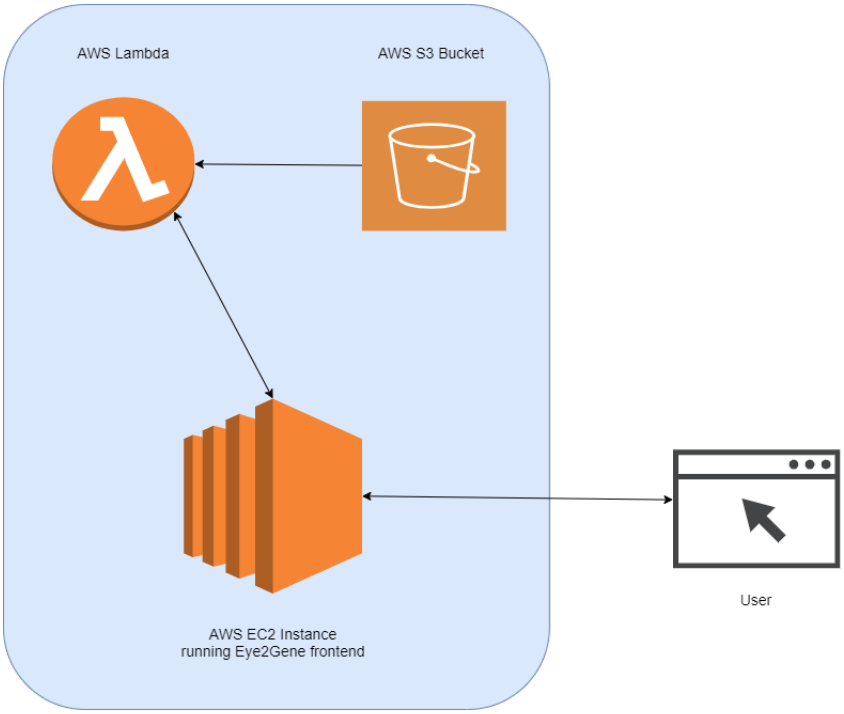

FIGURE 20: Diagram of AWS infrastructure used to host Eye2Gene website

In Figure 20 the flow of data within the system can be seen. A user interacts with the publicly available website, which then submits and images received from the user to the AWS Lambda function, which in turn reads the appropriate model files from an AWS bucket (storage location). Structuring the system like this means that it is easy to update any one component separately from the others.

Chapter 5

Results

Using the methods described above, training was performed on several different architectures of neural network, initially with the merged Dataset 1 + 2 and then with Dataset 3.

5.1 Custom 1

Without making many changes to hyperparameters and using fairly plain values, a final validation accuracy of 87.5% was achieved in Figure 21.

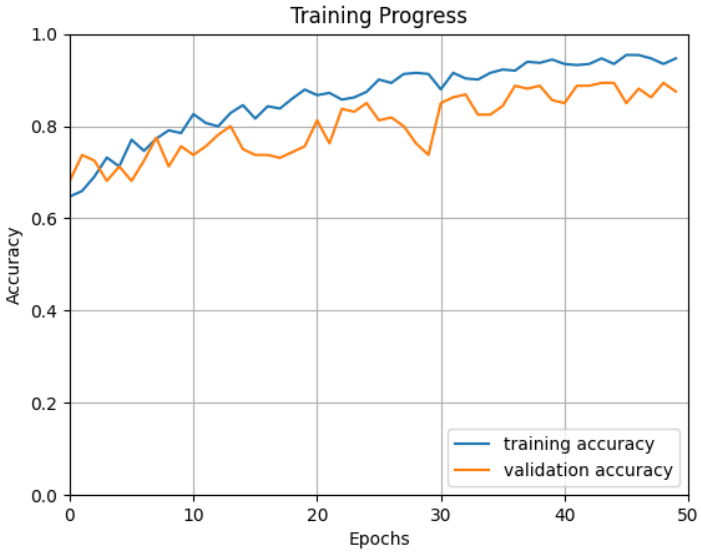

FIGURE 21: Custom 1 network training on Dataset 1. Batch size = 32, Learning rate = 1e-4, Dropout = 0.7, No augmentations

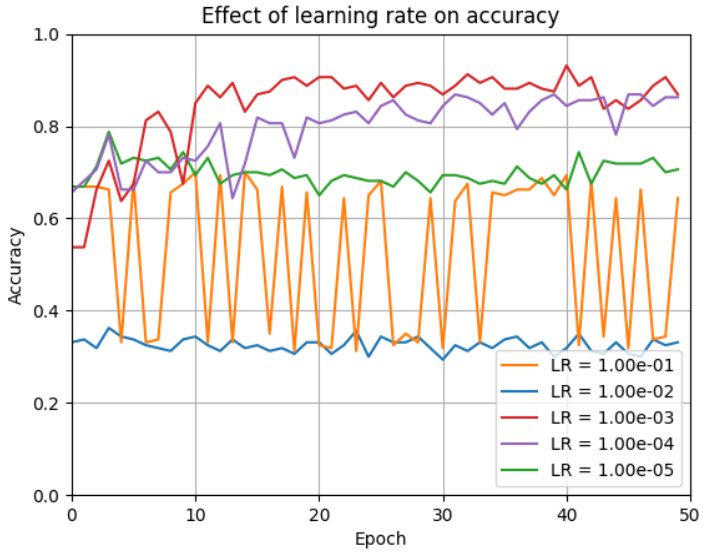

FIGURE 22: Comparison of network training accuracies with different learning rate values

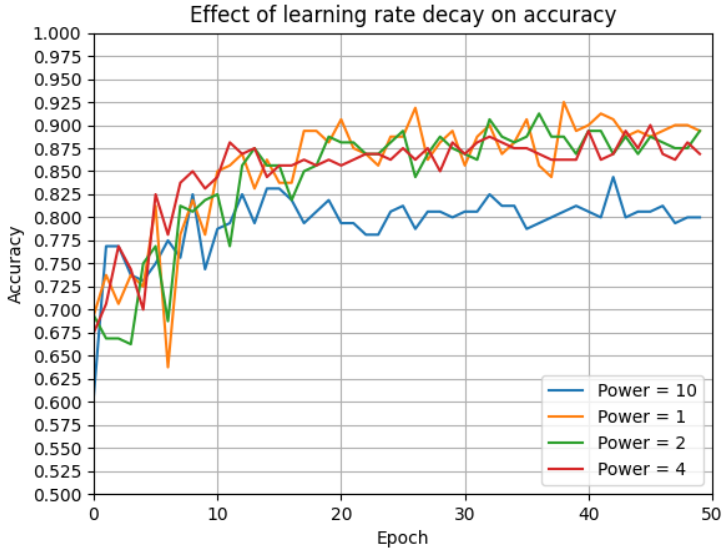

FIGURE 23: Comparison of network training accuracies with different learning rate decay rates. Initial LR = 1e-3

In Figure 22 a static learning rate of 1e-03 appears to perform best, reaching a high accuracy sooner than the others, as well as achieving an accuracy greater than

90% at one point. In Figure 23 we can see that introducing a decaying learning rate negatively affects the final training accuracy when it is higher than the 4th power. Using a linear learning decay rate appears to produce marginally better results, but accuracy is still just under 90%.

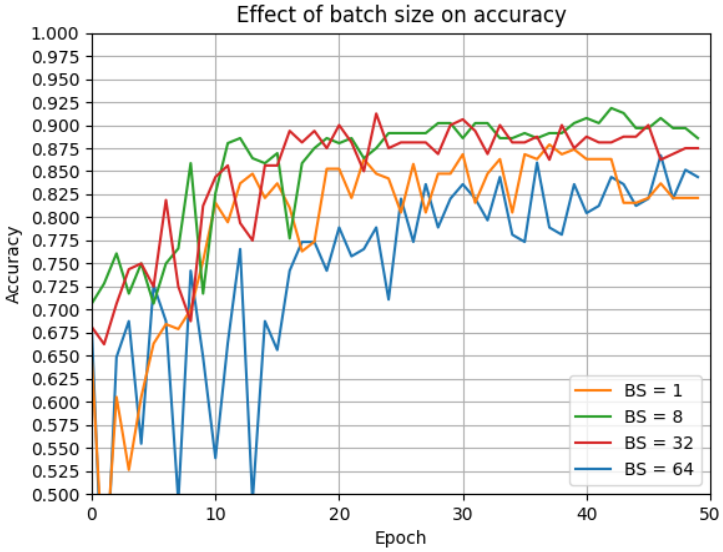

FIGURE 24: Comparison of network training accuracies with different batch size values. Initial LR = 1e-3, LR decay power = 1.

Figure 24 shows that a batch size of 8 appears to give the highest accuracy after 50 epochs.

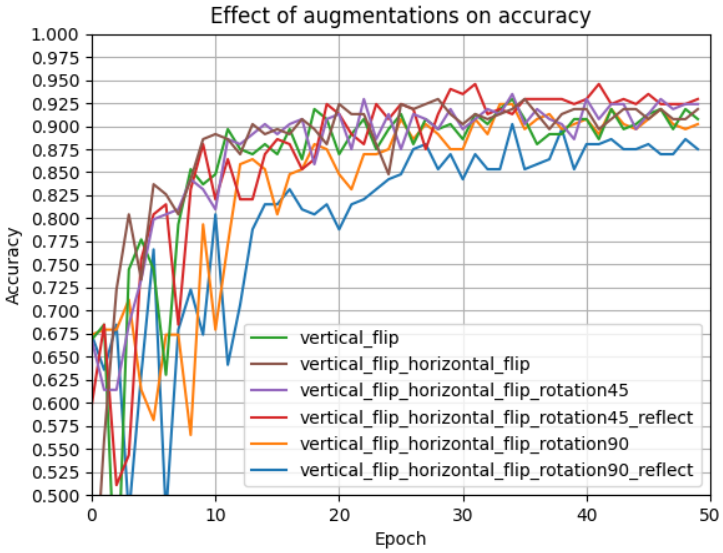

FIGURE 25: Comparison of network training accuracies with different image augmentations. Initial LR = 1e-3, LR decay power = 1, Batch size = 8.

Flipping images either horizontally or vertically gives an improved accuracy, as shown in Figure 25 but interestingly rotating an image by too large an angle actually decreases the accuracy.

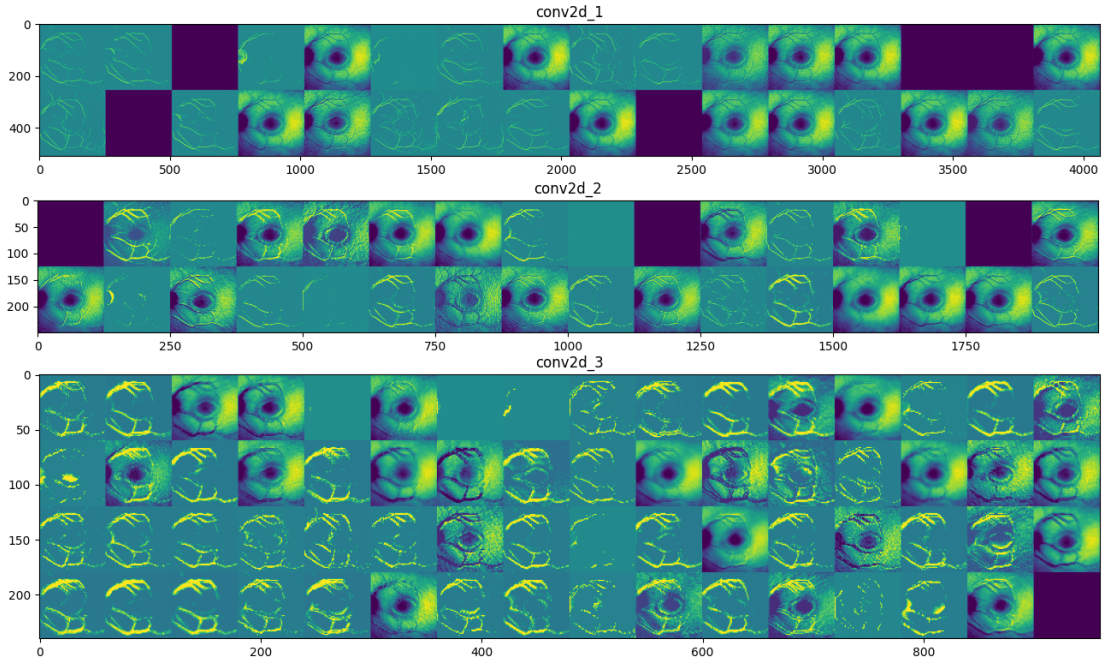

FIGURE 26: Visualisation of the neurons in the convolutional layers of a trained Custom 1 network when predicting on a new image

Figure 26 shows a visualisation of neuron activation in a trained Custom 1 network when provided with an input image. It's interesting to see how the different neurons activate based upon different features in the image. For example it appears that the blood vessels in the image are highlighted strongly in a lot of neurons, indicating that perhaps the structure of the blood vessels is important in classification. This figure isn't that useful at the moment for guiding hyperparameter decisions and is open to interpretation, but is nonetheless interesting to look at.

5.2 VGG16

An instance of the VGG16[29] network is created, loaded with weights from the ILSVRC challenge, to make use of transfer learning. The top layers are removed and our own 2-class dense layer is applied.

FIGURE 27: VGG16 network training on Dataset 1. Batch size = 32, Learning rate = 1e-4, Dropout = 0.7, No augmentations

Training an instance of the VGG16 model with sensible defaults gives a final validation accuracy of 86.25%. It appears in Figure 27 as though the accuracy was still increasing at the 50th epoch, so running the network for longer may improve the result.

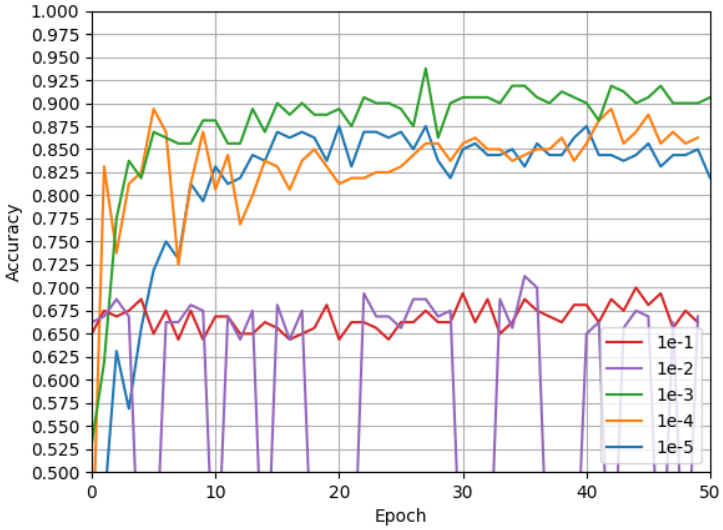

FIGURE 28: Effect of different learning rates on training of VGG16 on Dataset 1. Batch size = 32, Dropout = 0.7, No augmentations

A static learning rate of 1e-3 performs best, with a final validation accuracy of 91%. In Figure 28 it is possible to see the effects of a learning rate which is much too large. When the learning rate is 1e-2 the accuracy appears to oscillate, which is caused by the inability of the loss function to converge on a lower minimum due to constantly stepping over it.

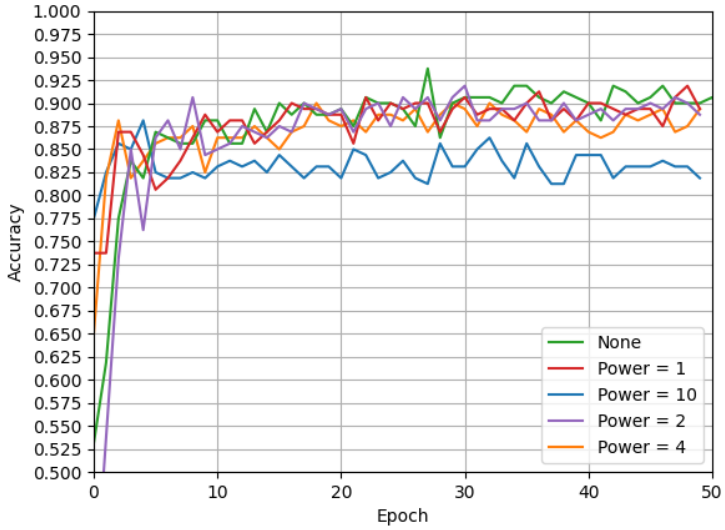

FIGURE 29: Effect of different learning rate decay values on training of VGG16 on Dataset 1. Batch size = 32, Dropout = 0.7, No augmentations, Initial LR 1e-3

Interestingly in Figure 29 introducing a decaying learning rate appears to have little effect, if anything it produces a lower final accuracy. A decay in learning rate should still be included as in principle it will be needed once the descent of the loss function requires smaller step sizes.

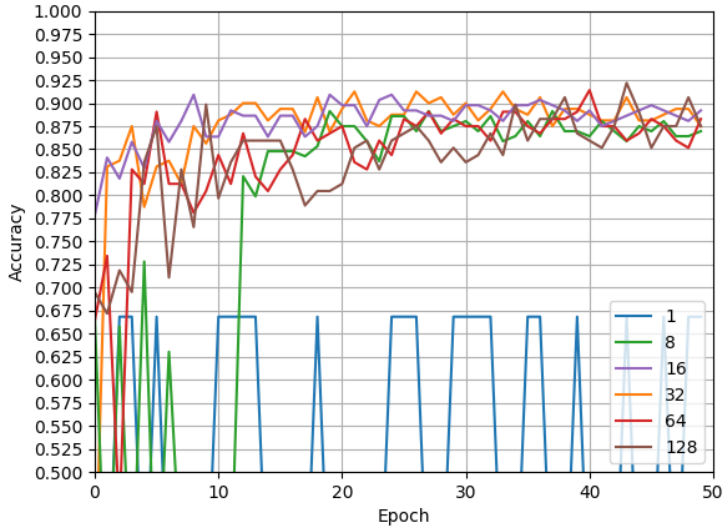

FIGURE 30: Effect of different batch size values on training of VGG16 on Dataset 1. Dropout = 0.7, No augmentations, Initial LR 1e-3, Linear LR decay

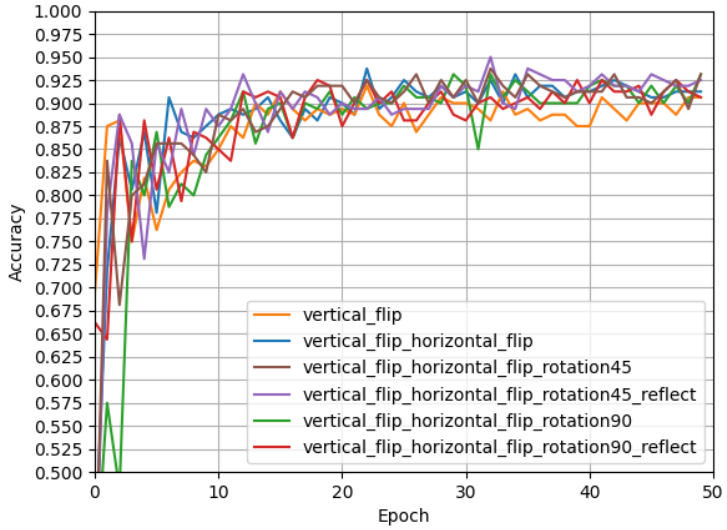

FIGURE 31: Effect of different image augmentations on training of VGG16 on Dataset 1. Dropout = 0.7, Initial LR 1e-3, Linear LR decay

In Figure 30 we can see that similarly to the Custom 1 network, a batch size of around 32 produces the best results in terms of conversion speed. Here we can also see that a batch size of 1, which is essentially stochastic, produces the expected result of not improving at all. It might be that a good accuracy can still be achieved with a batch size of 1, but it would take many many more epochs to obtain as the direction of the gradient descent is random. We can also see that in Figure 31 augmentations involving a rotation of 45 degrees seem to give the best results, although there is little difference between final accuracy values.

5.3 InceptionV3

An instance of the InceptionV3 network is created, loaded with weights from the ILSVRC[36] challenge, to make use of transfer learning. The top layers are removed and our own 2-class dense layer is applied with a softmax activation for a categorical distribution.

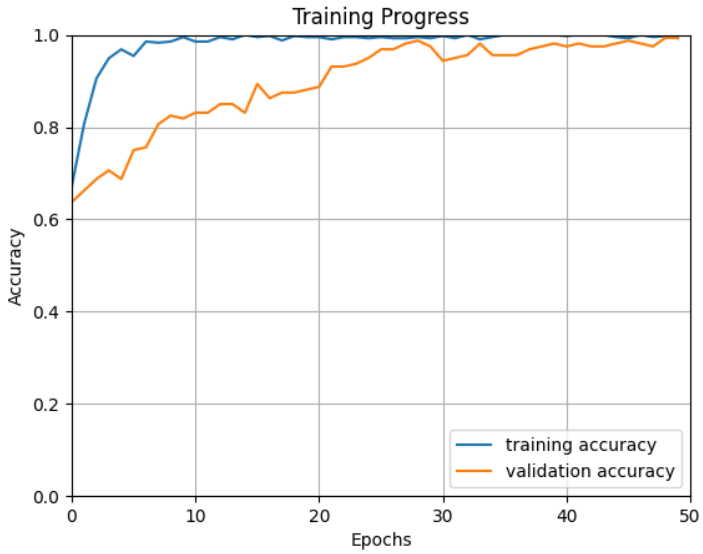

FIGURE 32: InceptionV3 network training on Dataset 1. Batch size = 32, Learning rate = 1e-4, Dropout = 0.7, No augmentations

High accuracy is achieved in Figure 32 without any changes to the chosen default parameters, resulting in a final accuracy of 99.3% after 50 epochs. There is initially some degree of overfitting, it may be possible to reduce this further with an even more aggressive dropout percentage, or by the addition of more data into the dataset.

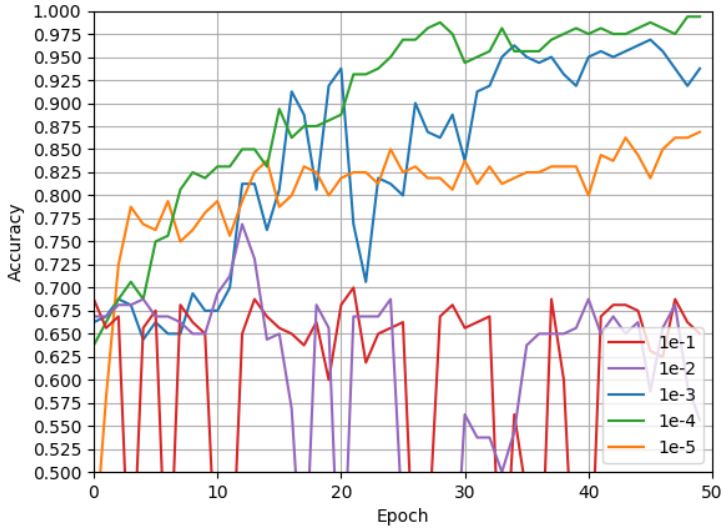

FIGURE 33: Effect of different learning rate values on training of InceptionV3 network on Dataset 1. Batch size = 32, Dropout = 0.7, No augmentations

It appears in Figure 33 that 1e-4 is the best initial learning rate for the InceptionV3 network, compared to 1e-3 previously with VGG16. Again we can see the effect of a learning rate which is too large on the accuracy, causing it to oscillate.

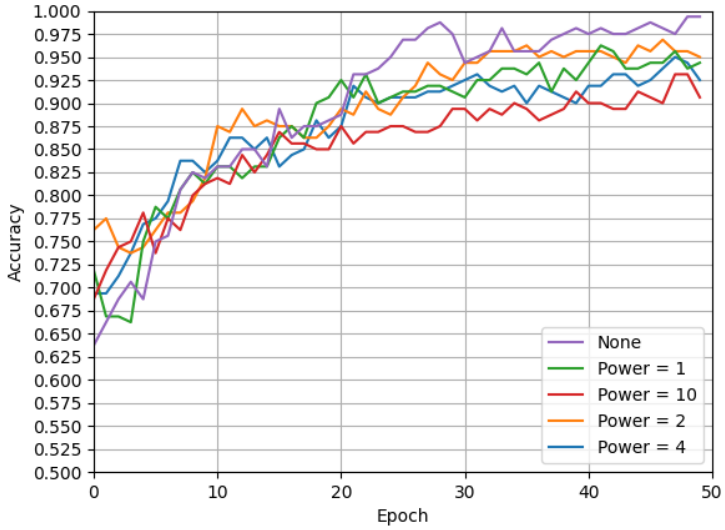

FIGURE 34: Effect of different learning rate decay values on training of InceptionV3 network on Dataset 1. Batch size = 32, Dropout = 0.7, initial LR 1e-4, No augmentations

These results show that having no variable learning rate achieves a higher final accuracy, which is counter-intuitive. As the loss function reaches a global minimum, the step size (controlled by the learning rate) will have to get smaller to allow for a more accurate result. It may be that the result shown in Figure 34 is erroneous, or that the training run with a decaying learning rate with a power of 2 will eventually reach a higher accuracy. In the interest of experimentation, further results will include an extra training run which will use the best hyperparameters with a decaying learning rate of power 2.

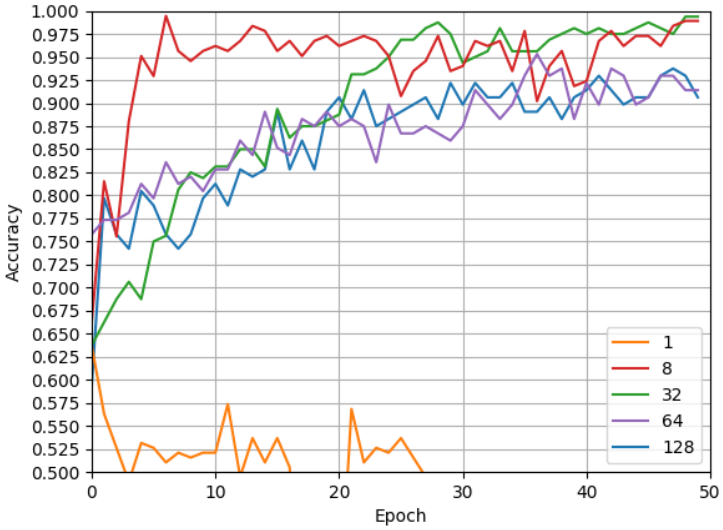

FIGURE 35: Effect of different batch sizes on training of InceptionV3 network on Dataset 1. Dropout = 0.7, Initial LR 1e-4, No augmentations

In Figure 35, a smaller batch size seems beneficial for the InceptionV3 network when trained on Dataset 1, reducing the number of epochs it takes the network to achieve high accuracy. Again the effect of having a batch size too small is shown by a very low and variable accuracy. The difference between a batch size of 1 and 8 is huge compared to previous architectures, which is noteworthy as the same difference in previous architectures was not as pronounced.

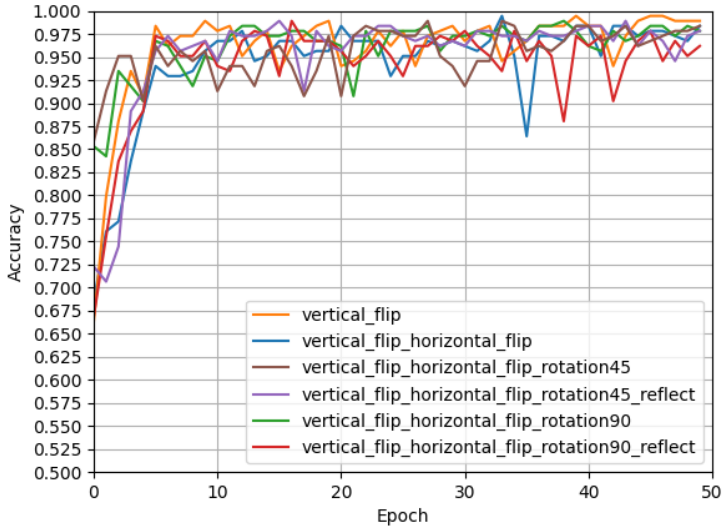

FIGURE 36: Effect of different image augmentations on training of InceptionV3 on Dataset 1. Batch size = 8, Dropout = 0.7, Initial LR 1e-4

It is quite hard to compare the different augmentations applied due to the variance in the accuracy in Figure 36. This could be due to not using a decaying learning rate, even though in previous tests it appeared to negatively affect final accuracy it might be worth testing this again now that high training accuracy is reached within fewer than 10 epochs. One theory for the negative effects of the decaying learning rate might be that the decay was too strong, preventing the network from taking big enough steps in the correct gradient descent direction.

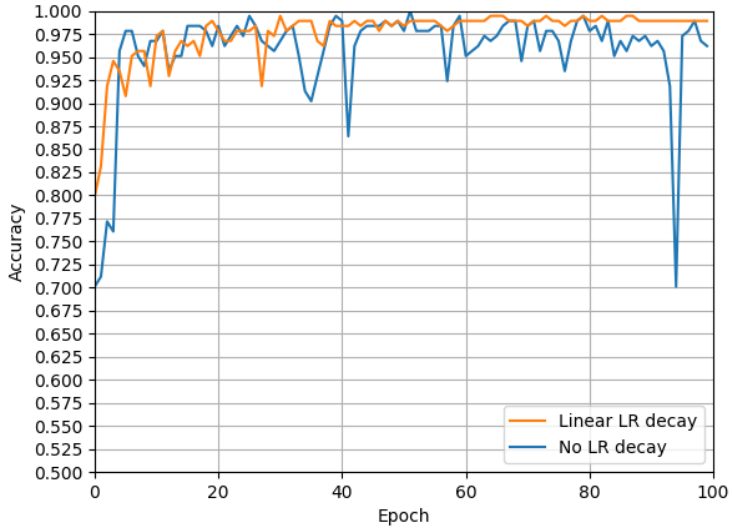

FIGURE 37: Effect of learning rate decay on training accuracy for a InceptionV3 network over 100 epochs. Augmentations: Vertical flip, Horizontal flip, Rotation range within 90 degrees

Indeed, after 100 epochs the effect of the decay on the learning rate can be seen in Figure 37 as having a stabilising effect on the accuracy. After only 50 epochs this effect is much harder to see visually.

5.4 InceptionResnetV2

An instance of the InceptionResnetV2 network is created, loaded with weights from the ILSVRC[36] challenge to make use of transfer learning. The top layers are removed and our own 2-class dense layer is applied with a softmax activation for a categorical distribution.

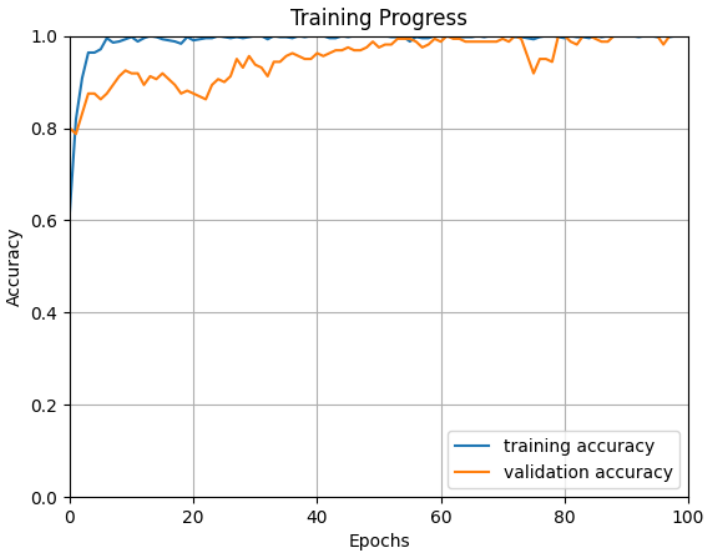

FIGURE 38: InceptionResnetV2 network training on Dataset 1. Batch size = 32, Learning rate = 1e-4, Dropout = 0.7, No augmentations

The network achieved a final accuracy of 100% on the validation data in Figure 38 with the selected default hyperparameters, the first architecture to have done so. This could be due to the improvements made to the Inception architecture, combined with the additional residual links inspired by Resnet[42] architectures.

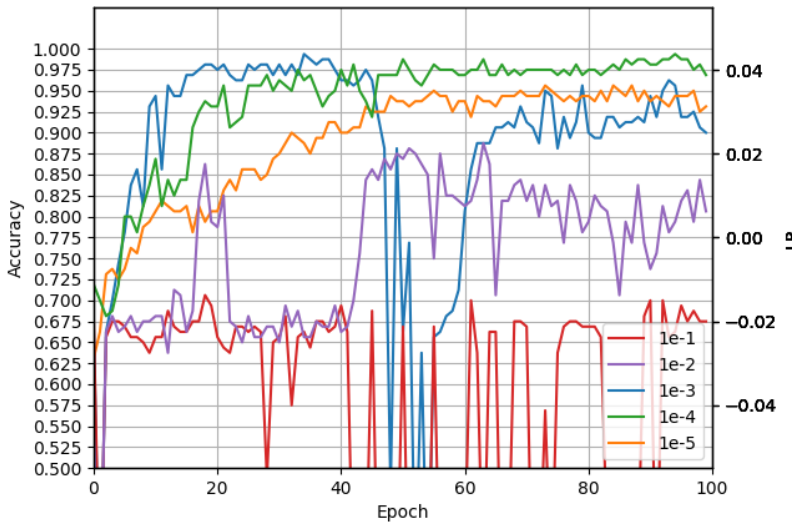

FIGURE 39: Effect of different learning rate values on training of InceptionResnetV2 network on Dataset 1. Batch size = 32, Dropout = 0.7, No augmentations

A learning rate of 1e-3 initially appears to be more desirable than 1e-4, but halfway through training suffers a dramatic loss of accuracy, possibly due to the loss function jumping around various local minima while not being able to descend any further towards the global minimum. The training run for the 1e-3 plot of Figure 39 could have been anomalous in some other way, perhaps due to particular filters chosen in the network's convolutional layers.

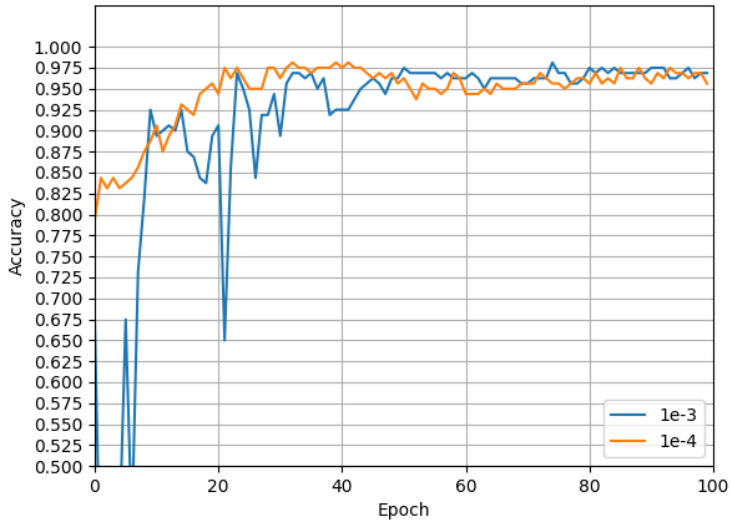

FIGURE 40: Comparison of initial learning rates of InceptionResnetV2 with LR decay of power 2

Applying a decaying learning rate does improve the situation in Figure 40, and results in a smoother curve, but still appears to favour 1e-4 as the better initial learning rate.

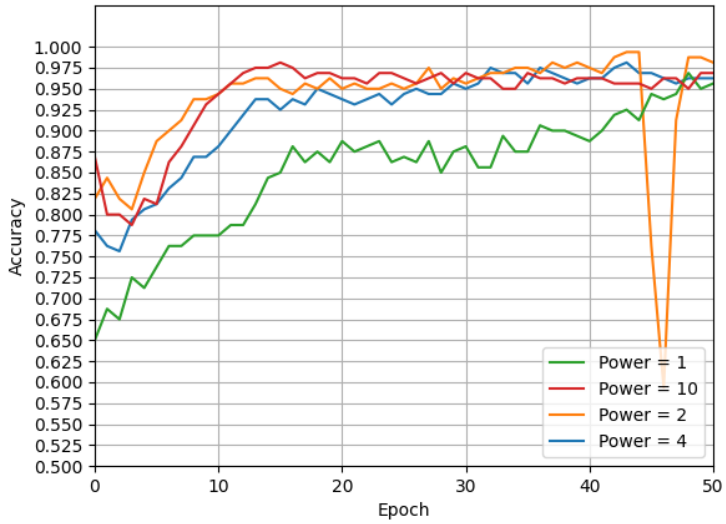

FIGURE 41: Effect of different learning rate decay values on training of InceptionResnetV2 network on Dataset 1. Dropout = 0.7, Initial LR 1e-4, No augmentations

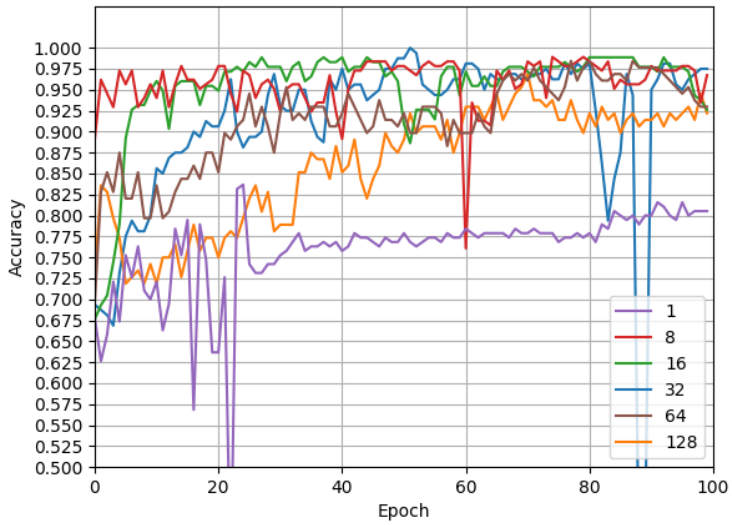

FIGURE 42: Effect of different batch sizes on training of InceptionResnetV2 network on Dataset 1. Dropout = 0.7, Initial LR 1e-4, No augmentations

Performance continues to be good with InceptionResnetV2, with some interesting effects being shown in Figure 41. Different learning rate decay values appear to perform similarly, with a decay value of 2 performing slightly better than others, although its arguable that this is just due to the randomness of dataset shuffling producing different results. Figure 42 shows an increase in peak accuracy again, up to 100% at one point using a batch size of 32, even though a batch size of 8 yields a very high accuracy within much fewer epochs. A batch size of 8 appears to be less consistent over many epochs compared to a batch size of 32, this is probably due to the nature of the batch size parameter, in that a larger batch size includes more images, producing a potentially more accurate direction for gradient descent. It's possible again that the shuffling of data might also contribute to this discrepancy, and that re-running these tests without shuffling the data might be more useful when comparing them.

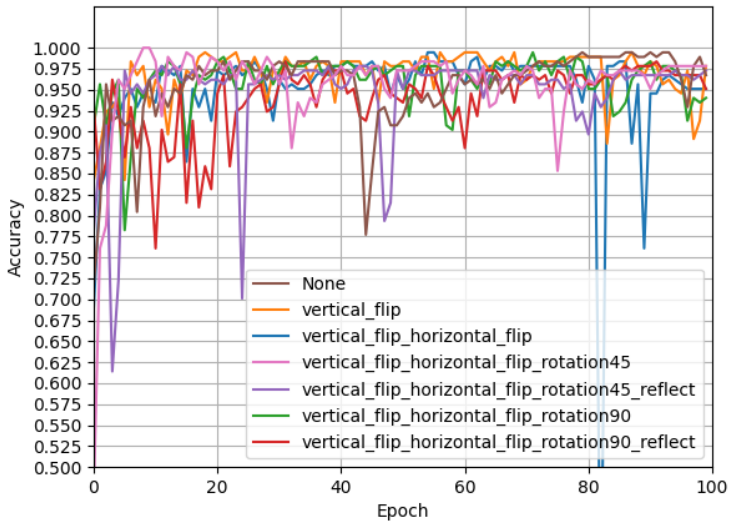

FIGURE 43: Effect of different image augmentations on training of InceptionResnetV2 on Dataset 1. Batch size = 8, Dropout = 0.7, Initial LR 1e-4

Using augmentations appears to only slightly affect the accuracy, in some cases producing a more consistent value, but it is hard to discern this from Figure 43.

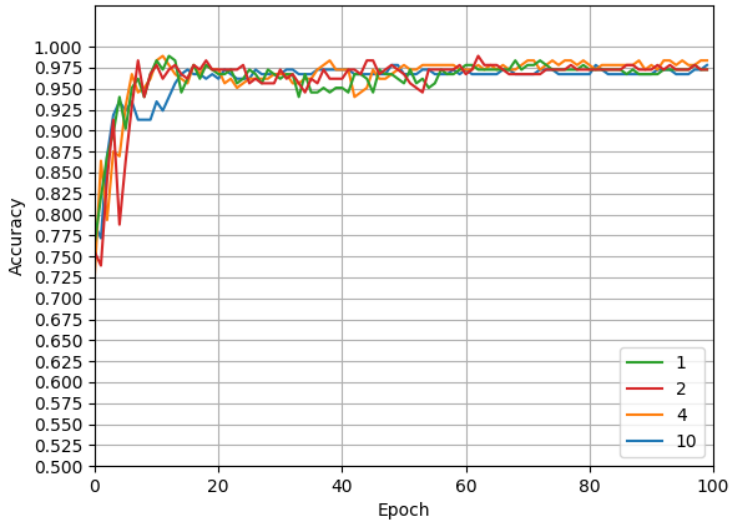

FIGURE 44: Effect of learning rate decay on training run with best hyperparameters of InceptionResnetV2 on Dataset 1. Batch size = 8, Dropout = 0.7, Initial LR 1e-4

It is hard to see which augmentations performed best, but we can see in Figure 43 that 'vertical_flip_horiztonal_flip_rotation45' achieved 100% accuracy at one point. Using those augmentations, perhaps the inconsistency in accuracy can be addressed with a decaying learning rate. In Figure 44 the best hyperparameters so far are selected and the learning rate decay value is changed. Using a decaying learning rate as opposed to no learning rate decay rate appears to produce a smoother curve. The hyperparameters used are shown in Table 7.

Model	Initial LR	LR Decay	Dropout	Batch Size	Epochs
InceptionResnetV2	1e-4	N/A	0.7	8	100

TABLE 7: Hyperparameters for best InceptionResnetV2 training run

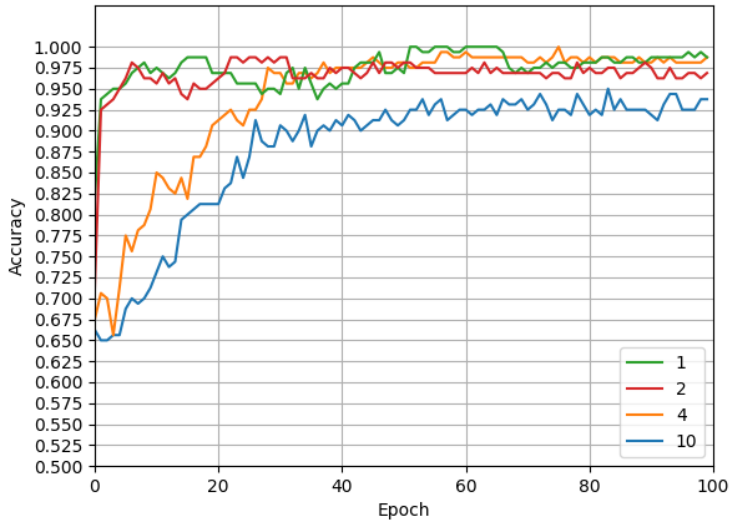

FIGURE 45: Effect of learning rate decay on training run with best hyperparameters of InceptionResnetV2 on Dataset 1. Batch size = 32, Dropout = 0.7, Initial LR 1e-4

Even with the learning rate decay, it seems the network is unable to achieve the 100% accuracy that was seen in Figure 42. In an attempt to obtain a better gradient descent direction the batch size is set to 32 and another test with varying learning rate decay values is performed. 100% accuracy is obtained with a batch size of 32 and briefly with batch size of 4, however the peak with a batch size of 32 is sustained for many more epochs so appears to be more beneficial. The best hyperparameters are now shown in Table 8.

Model	Initial LR	LR Decay	Dropout	Batch Size	Epochs
InceptionResnetV2	1e-4	1	0.7	32	100

TABLE 8: Hyperparameters for best InceptionResnetV2 training run with batch size 32

5.4.1 Timings

Model	Time Taken (s)	Number of layers
Custom 1	2	5
VGG16	3	23
InceptionV3	4	159
InceptionResnetV2	11	572

TABLE 9: Time taken to train for 1 epoch with different network architectures (batch size of 32, on Dataset 1)

Model	Peak Acc.	Final Acc.	Layers
Custom	95	93	5
VGG16	95	93	23
InceptionV3	99.3	**98.5**	159
InceptionResnetV2	**100**	98	572

TABLE 10: Number of images in original dataset

While InceptionV3 produced the best final accuracy after 100 epochs, Inception-ResnetV2 reached a high accuracy in fewer epochs than InceptionV3. However the running time of InceptionResnetV2 is larger than InceptionV3, as shown in Table 9. We can see that the time taken to train one epoch for each network increases with the number of layers the network has. The difference in time taken per epoch for the two Inception based architectures is relatively large, so it appears that if training time is important here it might be worth using InceptionV3 rather than InceptionResnetV2 to train quicker at the expense of around 1% accuracy. This might actually not be optimal at all, as InceptionResnetV2 doesn't need as many epochs as InceptionV3 to reach peak accuracy (Table 10). In general the models with more layers seem to perform better, reaching a progressively higher peak accuracy on Dataset 1.

5.5 Dataset 3

Due to time constraints it was not possible to perform the full breadth of hyperparameter testing that was performed on Dataset 1, so the best performing hyperparameters were picked from each model class and used to train each class of model. Dataset 3 is of the same data type (being retinal scans taken with AF

imaging), so should have similar properties to that of Dataset 1 and hopefully not affect the maximum accuracy that can be obtained.

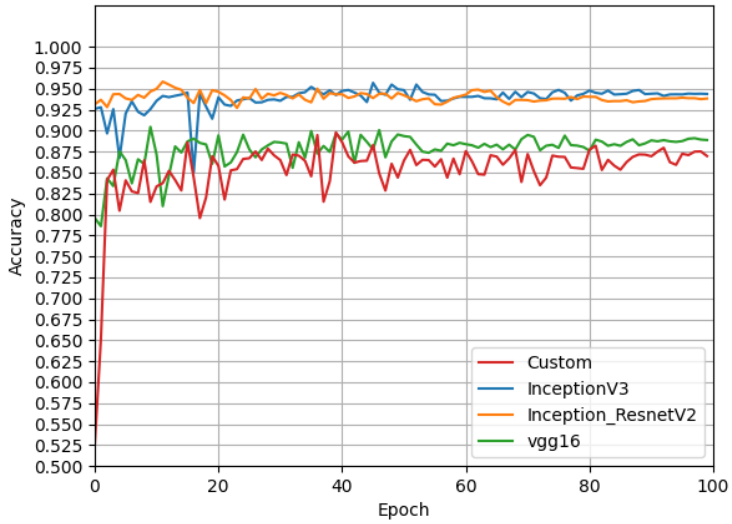

FIGURE 46: Comparison of best models from Dataset 1 trained on Dataset 3

In Figure 46 we see the results of training on Dataset 3. High accuracy was still obtained with all types of model architecture, but the Inception architecture was again responsible for the highest accuracies of 96%. There is a clear distinction seen on the graph between inception-based models and their VGG counterparts, with Inception models achieving 5-6% better peak accuracy in most cases. More research could be done to identify which augmentations are relevant in increasing the accuracy, as here it appears they have little effect. The discovery of a larger batch size having an increase in accuracy with the InceptionResnetV2 network was integral to achieving 100% accuracy with that model.

Model	Initial LR	LR Decay	Batch Size	Epochs
Custom 1	1e-3	N/A	8	100
VGG16	1e-3	1	32	100
InceptionV3	1e-4	1	8	100
InceptionResnetV2	1e-4	1	32	100

TABLE 11: Best hyperparameters for each model trained on Dataset 1, trained on Dataset 3

Model	Augmentations
Custom 1	vertical_flip,horizontal_flip,rotation_range=45
VGG16	vertical_flip,horizontal_flip,rotation_range=45,fill_mode=reflect
InceptionV3	N/A
InceptionResnetV2	vertical_flip,horizontal_flip,rotation=45

TABLE 12: Augmentations used on the best training runs of Dataset 3

Table 11 shows the various parameters that were used when training each architecture on Dataset 3. The largest anomaly here is the omission of any data augmentations from the InceptionV3 parameters. This could be incorrect, or at least distort the results compared to other architectures. If the experiment is run again some more research should be done to identify relevant augmentations.

Chapter 6

Discussion

The Custom model appears to reach an impressive maximum accuracy of 95% on Dataset 1 given that it is a relatively small model of 5 layers with relatively little learning capacity. This model could be of use in scenarios where a prediction on new data is required as quickly as possible, or where storage space and compute time comes at a premium. Using a learning rate of 1e-3 plus image augmentations with the Custom 1 model appears to increase the accuracy by 1-2% compared to the baseline hyperparameters. The best accuracy for the Custom 1 network was achieved with a batch size of 8, learning rate of 1e-3 and a learning rate decay parameter of 1. Notably here image rotations in the 0-45 degree range seemed to positively affect accuracy, but rotations in the 0-90 degree range negatively affected it.

VGG16 did not really provide any big improvement in accuracy, so its possible that the convolution + max pooling layer pair architecture had reached it's limit by this point, and adding more layers provides no improvement. Modifications to batch size and learning rate ultimately providing an increase of about 6% with some types of augmentation. This similarity is to be expected as the Custom 1 network uses essentially the same components in it's architecture that the VGG family uses.

Training with InceptionV3 marks the first large improvement in accuracy. Using the selected default hyperparameters a validation accuracy of 99.3% was achieved on the first attempt, which is obviously extremely close to the best possible result. The validation split of 30% on Dataset 1 means that there are 190 images in the validation dataset, which can provide an accuracy of $\frac{1}{190}$ (0.52%), meaning that 99.3% is about as close as we can get to perfect without achieving 100%. The success of this network could possibly be because of the 'Inception module', but could also be because of the greatly increased depth of the network, with InceptionV3 containing a depth of 159 layers compared to VGG16's 23, allowing for more capacity in the network to learn.

InceptionResnetV2 further improves upon this by achieving a 100% validation accuracy with default parameters. The number of epochs needed to achieve a very high accuracy was also decreased, showing the effect of the improvements made over InceptionV3. Using a small batch size of 8 produced some good results for Dataset 1, this could be due to the small size allowing the network to escape local

minima in the process of calculating the descent direction. With the Inception architecture a small batch size actually produced the best results and the fastest increase in accuracy.

In conclusion, I managed to train a neural network to 100% accuracy on a small cleaned data set of retinal images, and achieved 95% accuracy on a much larger dataset after some cleaning steps. This shows that it is possible to learn the features in retinal images created by different gene mutations and develop a model that can classify them with extremely high accuracy.

The accuracy on Dataset 3 could possibly be further improved in a number of ways. It's possible that irrelevant or bad images are still present in Dataset 3 that were not removed by my initial cleaning steps. The cleaning could be further improved using a medical assessment by an expert of which images are clinically relevant and removing those which are not. There are however tens of thousands of images to manually label which would be very tedious, perhaps a more useful method would be to label a few thousand manually as 'good' and 'bad', then train a CNN on these images to recognise obviously invalid images. Steps would have to be taken to ensure the subset of images you train the network on has sufficient diversity. This would save the expert having to sift through data for weeks, and also might provide a cleaning network which could be reused for further data. Failing any of these methods, an increase in data points would also obviously be beneficial, and attempts to acquire more data could be pursued.

The higher accuracy on Dataset 1 and lower on Dataset 3 could indicate that there are variants of images in Dataset 3 which the image augmentations applied do not enhance the feature extraction of. More augmentations could be applied to extract more features.

Further improvements could be made to a variety of aspects of this project. New learning rate schedulers could be used such as a cyclic learning rate scheduler[49] which has been shown to improve accuracy further by repeating certain patterns of learning rate decay, to try and prevent the loss function falling into a local minimum rather than a global one. Other methods of validating the results of the model could be used, e.g cross-fold validation could be utilised to increase the confidence in the prediction that the model gives. Saliency maps could be used to identify which areas of the data the network is learning from, and develop different augmentations to apply that provide better clarity on those areas.

Of course using different architectures might also provide further benefits, not just to peak accuracy, which is already near its maximum, but maybe in training time or resource usage of the model. The current Inception models take up quite a lot of hard disk space and require a fair amount of memory to load and run, one promising model to try might be InceptionV4, an improved version of the InceptionV3 model, which wasn't included in my testing as it wasn't readily available

in the Keras library whereas the others were.

During the deployment of the system to AWS there were no problems encountered, working with AWS Lambda reduced the amount of maintenance required for the system to run, and decouples the predictive function of the network from whatever front end is chosen to be used.

Overall I think the project was completed successfully, with the main goal of achieving high accuracy on the provided dataset being reached. There are many other genes that are responsible for IRDs, and the network could potentially be used to identify more classes than just ABCA4 and USH2A to include some of these. The main problem that is encountered with IRD classification is the availability of data, and as more classes are included there may not be enough information to achieve very high accuracy. Hopefully as time goes on this situation will change, with more data being collected making this method of diagnosis increasingly viable. The work done here will be continued by Moorfields Eye Hospital, with the initial aim of extending the classification to the top 4 genes in Dataset 3, but longer term extending it to classify many more.

Bibliography

[1] Gerald Liew, Michel Michaelides, and Catey Bunce. "A Comparison of the Causes of Blindness Certifications in England and Wales in Working Age Adults (1664 Years), 19992000 with 20092010". In: *BMJ Open* 4.2 (Feb. 1, 2014), e004015. ISSN: 2044-6055, 2044-6055. DOI: 10.1136/bmjopen-2013-004015. pmid: 24525390. URL: https://bmjopen.bmj.com/content/4/2/e004015 (visited on 04/24/2020).

[2] Yu Fujinami-Yokokawa et al. "Prediction of Causative Genes in Inherited Retinal Disorders from Spectral-Domain Optical Coherence Tomography Utilizing Deep Learning Techniques". In: *Journal of Ophthalmology* 2019 (Apr. 9, 2019), pp. 1–7. ISSN: 2090-004X, 2090-0058. DOI: 10.1155/2019/1691064. URL: https://www.hindawi.com/journals/joph/2019/1691064/ (visited on 03/25/2020).

[3] Neruban Kumaran et al. "Leber Congenital Amaurosis / Early-Onset Severe Retinal Dystrophy Overview". In: *GeneReviewső*. Ed. by Margaret P. Adam et al. Seattle (WA): University of Washington, Seattle, 1993. pmid: 30285347. URL: http://www.ncbi.nlm.nih.gov/books/NBK531510/ (visited on 04/24/2020).

[4] Nashila Hirji et al. "Achromatopsia: Clinical Features, Molecular Genetics, Animal Models and Therapeutic Options". In: *Ophthalmic Genetics* 39.2 (Mar. 4, 2018), pp. 149–157. ISSN: 1381-6810, 1744-5094. DOI: 10.1080/13816810.2017.1418389. URL: https://www.tandfonline.com/doi/full/10.1080/13816810.2017.1418389 (visited on 04/24/2020).

[5] Genetics Home Reference. *What Is a Gene Mutation and How Do Mutations Occur?* URL: https://ghr.nlm.nih.gov/primer/mutationsanddisorders/genemutation (visited on 04/24/2020).

[6] International Human Genome Sequencing Consortium. "Initial Sequencing and Analysis of the Human Genome". In: *Nature* 409.6822 (Feb. 2001), pp. 860–921. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/35057062. URL: http://www.nature.com/articles/35057062 (visited on 04/24/2020).

[7] Nikolas Pontikos et al. "Genetic Basis of Inherited Retinal Disease in a Molecularly Characterized Cohort of More Than 3000 Families from the United Kingdom". In: *Ophthalmology* (Apr. 16, 2020). ISSN: 0161-6420. DOI:

10.1016/j.ophtha.2020.04.008. URL: http://www.sciencedirect.com/science/article/pii/S0161642020303328 (visited on 08/30/2020).

[8] Alessandra Maugeri et al. "Mutations in the ABCA4 (ABCR) Gene Are the Major Cause of Autosomal Recessive Cone-Rod Dystrophy". In: *The American Journal of Human Genetics* 67.4 (2000), pp. 960–966.

[9] Bo Dreyer et al. "Identification of Novel USH2A Mutations: Implications for the Structure of USH2A Protein". In: *European Journal of Human Genetics* 8.7 (July 1, 2000), pp. 500–506. ISSN: 1476-5438. DOI: 10.1038/sj.ejhg.5200491. URL: https://doi.org/10.1038/sj.ejhg.5200491.

[10] Steffen Schmitz-Valckenberg et al. "FUNDUS AUTOFLUORESCENCE IMAGING: Review and Perspectives". In: *Retina* 28.3 (Mar. 2008), pp. 385–409. ISSN: 0275-004X. DOI: 10.1097/IAE.0b013e318164a907. URL: https://insights.ovid.com/crossref?an=00006982-200803000-00002 (visited on 04/24/2020).

[11] *The Nuts and Bolts of Fundus Autofluorescence Imaging*. 9/1/2012 12:00:00 AM. URL: https://www.aao.org/eyenet/article/nuts-bolts-of-fundus-autofluorescence-imaging (visited on 04/13/2020).

[12] Janet R. Sparrow et al. "Interpretations of Fundus Autofluorescence from Studies of the Bisretinoids of the Retina". In: *Investigative Opthalmology & Visual Science* 51.9 (Sept. 1, 2010), p. 4351. ISSN: 1552-5783. DOI: 10.1167/iovs.10-5852. URL: http://iovs.arvojournals.org/article.aspx?doi=10.1167/iovs.10-5852 (visited on 08/20/2020).

[13] Ann E. Elsner et al. "Infrared Imaging of Sub-Retinal Structures in the Human Ocular Fundus". In: *Vision Research* 36.1 (Jan. 1, 1996), pp. 191–205. ISSN: 0042-6989. DOI: 10.1016/0042-6989(95)00100-E. URL: http://www.sciencedirect.com/science/article/pii/004269899500100E (visited on 04/24/2020).

[14] D Huang et al. "Optical Coherence Tomography". In: *Science* 254.5035 (Nov. 22, 1991), p. 1178. DOI: 10.1126/science.1957169. URL: http://science.sciencemag.org/content/254/5035/1178.abstract.

[15] *Retina*. URL: http://medcell.med.yale.edu/histology/sensory_systems_lab/retina.php (visited on 08/30/2020).

[16] Geoffrey E. Hinton and Tim Shallice. "Lesioning an Attractor Network: Investigations of Acquired Dyslexia". In: *Psychological Review* 98.1 (1991/00/00), pp. 74–95. ISSN: 0033-295X.

[17] F. Rosenblatt. "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain." In: *Psychological Review* 65.6 (1958), pp. 386–408. ISSN: 1939-1471, 0033-295X. DOI: 10.1037/h0042519. URL: http://doi.apa.org/getdoi.cfm?doi=10.1037/h0042519 (visited on 04/24/2020).

[18] Bernard Widrow and Marcian E Hoff. *Adaptive Switching Circuits*. Stanford Univ Ca Stanford Electronics Labs, 1960.

[19] G. Hinton and Terrence Sejnowski. "Learning and Relearning in Boltzmann Machines". In: *Parallel Distributed Processing* 1 (Jan. 1, 1986).

[20] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning Representations by Back-Propagating Errors". In: *Nature* 323.6088 (Oct. 1986), pp. 533–536. ISSN: 1476-4687. DOI: 10.1038/323533a0. URL: https://www.nature.com/articles/323533a0 (visited on 04/24/2020).

[21] Y. Le Cun and Françoise Fogelman-Soulié. "Modèles connexionnistes de l'apprentissage". In: *Intellectica* 2.1 (1987), pp. 114–143. DOI: 10.3406/intel.1987.1804. URL: https://www.persee.fr/doc/intel_0769-4113_1987_num_2_1_1804 (visited on 04/24/2020).

[22] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (Nov. 1, 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: https://doi.org/10.1162/neco.1997.9.8.1735 (visited on 04/24/2020).

[23] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. "A Fast Learning Algorithm for Deep Belief Nets". In: *Neural Computation* 18.7 (July 2006), pp. 1527–1554. ISSN: 0899-7667, 1530-888X. DOI: 10.1162/neco.2006.18.7.1527. URL: http://www.mitpressjournals.org/doi/10.1162/neco.2006.18.7.1527 (visited on 04/24/2020).

[24] Yann LeCun et al. "Generalization and Network Design Strategies". In: *Connectionism in perspective* 19 (1989), pp. 143–155.

[25] George E. Dahl, Navdeep Jaitly, and Ruslan Salakhutdinov. "Multi-Task Neural Networks for QSAR Predictions". In: (June 4, 2014). arXiv: 1406.1231 [cs, stat]. URL: http://arxiv.org/abs/1406.1231 (visited on 04/24/2020).

[26] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. "Searching for Exotic Particles in High-Energy Physics with Deep Learning". In: *Nature Communications* 5.1 (Sept. 2014), p. 4308. ISSN: 2041-1723. DOI: 10.1038/ncomms5308. arXiv: 1402.4735. URL: http://arxiv.org/abs/1402.4735 (visited on 04/24/2020).

[27] Seymour Knowles-Barley et al. “Deep Learning for the Connectome”. In: *GPU Technology Conference*. Vol. 26. 2014.

[28] Zhou and Chellappa. “Computation of Optical Flow Using a Neural Network”. In: *IEEE 1988 International Conference on Neural Networks*. IEEE 1988 International Conference on Neural Networks. July 1988, 71–78 vol.2. DOI: 10.1109/ICNN.1988.23914.

[29] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: (Apr. 10, 2015). arXiv: 1409.1556 [cs]. URL: http://arxiv.org/abs/1409.1556 (visited on 03/25/2020).

[30] Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overtting”. In: *Journal of Machine Learning Research* 15 (June 2014), pp. 1929–1958.

[31] Vinod Nair and Geoffrey E Hinton. “Rectified Linear Units Improve Restricted Boltzmann Machines”. In: *ICML’10* 27 (June 2010), pp. 807–814.

[32] *MedicMind*. URL: https://www.medicmind.tech/ (visited on 04/23/2020).

[33] Mital Shah, Ana Roomans Ledo, and Jens Rittscher. “Automated Classification of Normal and Stargardt Disease Optical Coherence Tomography Images Using Deep Learning”. In: *Acta Ophthalmologica* (Jan. 24, 2020), aos.14353. ISSN: 1755-375X, 1755-3768. DOI: 10.1111/aos.14353. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/aos.14353 (visited on 03/25/2020).

[34] Y. LeCun et al. “Backpropagation Applied to Handwritten Zip Code Recognition”. In: *Neural Computation* 1.4 (Dec. 1, 1989), pp. 541–551. ISSN: 0899-7667. DOI: 10.1162/neco.1989.1.4.541. URL: https://doi.org/10.1162/neco.1989.1.4.541 (visited on 04/24/2020).

[35] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. Version 3. In: (Mar. 2, 2015). arXiv: 1502.03167 [cs]. URL: http://arxiv.org/abs/1502.03167 (visited on 04/26/2020).

[36] Olga Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[37] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Communications of the ACM* 60.6 (May 24, 2017), pp. 84–90. ISSN: 00010782. DOI: 10.1145/3065386. URL: http://dl.acm.org/citation.cfm?doid=3098997.3065386 (visited on 03/25/2020).

[38] Christian Szegedy et al. "Going Deeper with Convolutions". In: (Sept. 16, 2014). arXiv: 1409.4842 [cs]. URL: http://arxiv.org/abs/1409.4842 (visited on 04/19/2020).

[39] Min Lin, Qiang Chen, and Shuicheng Yan. "Network In Network". In: (Mar. 4, 2014). arXiv: 1312.4400 [cs]. URL: http://arxiv.org/abs/1312.4400 (visited on 04/24/2020).

[40] Ross Girshick et al. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation". In: (Oct. 22, 2014). arXiv: 1311.2524 [cs]. URL: http://arxiv.org/abs/1311.2524 (visited on 04/24/2020).

[41] Christian Szegedy et al. "Rethinking the Inception Architecture for Computer Vision". In: (Dec. 11, 2015). arXiv: 1512.00567 [cs]. URL: http://arxiv.org/abs/1512.00567 (visited on 04/22/2020).

[42] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: (Dec. 10, 2015). arXiv: 1512.03385 [cs]. URL: http://arxiv.org/abs/1512.03385 (visited on 04/22/2020).

[43] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[44] Christian Szegedy et al. "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning". In: (Aug. 23, 2016). arXiv: 1602.07261 [cs]. URL: http://arxiv.org/abs/1602.07261 (visited on 04/22/2020).

[45] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: (Jan. 29, 2017). arXiv: 1412.6980 [cs]. URL: http://arxiv.org/abs/1412.6980 (visited on 08/21/2020).

[46] *Keras-Team/Keras*. Keras, Apr. 24, 2020. URL: https://github.com/keras-team/keras (visited on 04/24/2020).

[47] *Amazon Web Services (AWS) - Cloud Computing Services*. URL: https://aws.amazon.com/ (visited on 04/24/2020).

[48] *TensorFlow Lite | ML for Mobile and Edge Devices*. URL: https://www.tensorflow.org/lite (visited on 08/21/2020).

[49] Leslie N. Smith. "Cyclical Learning Rates for Training Neural Networks". In: (Apr. 4, 2017). arXiv: 1506.01186 [cs]. URL: http://arxiv.org/abs/1506.01186 (visited on 08/21/2020).