Abstract

The objective of this project is to investigate the mathematics behind deep learning techniques which can be applied to images, these techniques will be programmed to evaluate their effectiveness using different optimizers.

We start this report by understanding neural networks and build up to convolutional neural networks (CNN). We then evaluate optimizers to maximize accuracies. Research into autoencoders and generative adversarial networks (GAN) is performed to look at the uses of these more advanced techniques.

We then code our CNN, autoencoder and a GAN to evaluate their results to search for further improvements, such as how does the amount of epochs effect the accuracy of the networks predictions? What is the accuracy of results with different data sets (is it adaptive)? How well do our autoencoders denoise images and how well goes the GAN generate images?

Contents

1 Introduction **4**

1.1 Objectives 4

1.2 Background 4

1.3 Review of literature 5

2 Neural Networks **8**

2.1 Biological Neuron 8

2.2 Artificial Neuron 8

2.3 Activation Functions 9

2.4 Neural Network 10

2.5 Propagation 10

2.6 Backpropagation 12

3 Convolutional Neural Networks **14**

3.1 Convolutions 14

3.2 Convolutional Neural Network Architecture 14

3.3 Convolutional Layers and forward propagation 15

3.4 Convolutional Layer Backpropagation 16

3.5 Pooling Layers 17

3.6 Max Pooling Layer Backpropagation 18

4 Optimization Techniques **19**

4.1 Optimizers 19

4.2 Gradient Descent 19

4.3 Stochastic Gradient Descent 20

4.4 Adaptive Moment Estimation 21

5 Autoencoders **22**

5.1 Introduction 22

5.2 Denoising autoencoders 22

6 Generative adversarial network **24**
6.1 Introduction . . . 24
6.2 GAN algorithm . . . 25

7 Implementation **26**
7.1 Data set . . . 26
7.2 Desired Discoveries . . . 26
7.3 Code Explanation . . . 27
7.3.1 Convolutional Neural Network . . . 27
7.3.2 Autoencoder . . . 28
7.3.3 Generative Adversarial Network . . . 29
7.3.4 Activation function comparison . . . 29

8 Results **30**
8.1 Activation function simulation comparison . . . 30
8.2 Visualising . . . 30
8.3 Predicting . . . 31
8.4 How the amount of epochs effects accuracy . . . 32
8.5 Optimization comparison . . . 32
8.6 How adaptive is our CNN on a different data set . . . 33
8.7 Autoencoder results . . . 33
8.8 Generative Adversarial Networks . . . 37

9 Conclusion **38**
9.1 Conclusion . . . 38

10 Acknowledgements **39**

11 Appendix **40**
11.1 Code . . . 40
11.1.1 CNN Code . . . 40
11.1.2 Autoencoder Code . . . 42
11.1.3 Generative Adversarial Network Code . . . 44
11.1.4 Optimizer speed test . . . 47
11.2 Research Plan . . . 48
11.2.1 Description . . . 48
11.2.2 Motivation . . . 48
11.2.3 Connection with previous studies . . . 48
11.2.4 Literature survey . . . 49
11.2.5 References . . . 50

11.2.6 Equipment, facilities and cost 51
11.2.7 Action plan 51
11.2.8 Risk assessment 51
11.2.9 Discussion arrangements 52

Chapter 1

Introduction

1.1 Objectives

The aims for this project are to achieve a deep understanding of the mathematics behind the deep learning techniques used on images and then understand how to apply and optimize these techniques whilst showing their uses in the scientific world through discussion and implementation in Python.

We would like these techniques to be able to classify images, denoise images and generate similar images to images shown to our models. The techniques discovered in this project will therefore be convolution neural networks, denoising autoencoders and generative adversarial networks.

These techniques will be firstly applied on the MNIST handwritten data set, and then other data sets to determine how versatile our model is. Accuracies of each technique will be examined and an investigation into how to maximize our accuracies will be carried out.

1.2 Background

The importance behind this project is image recognition, it's extremely valuable as it has a wide range of applications in the scientific community. It can be used for diseases, for example it's an enormous ally to have against cancer as it can help detect its early stages which professional doctors are not capable of.

The theory behind convolutional neural networks has developed over a number of years. Neurophysiologists Torsten Wiesel and David Hubel's discovery of simple and complex cells responding to images in 1959 [10] was the catalyst to Kunihiko Fukushima's research in 1980 [11] on a neocognitron model with simple cells as layer 1 and complex cells as layer 2 which are capable of recognising images in 1980. Nine years later the first fully developed CNN was created and used for recognising digits for the postal service by Yann LeCun [12]. [17]

The reasoning for the development of convolutional neural networks can be shown here, recognising images such as digits can help automate society preventing the need for humans to do repetitive and sometimes difficult tasks at a high accuracy.

Generative adversarial networks are a more recent invention by a group of researchers at the University of Montreal [13]. In 2014 a framework was proposed which allowed the generation of images, including some (slightly scary) looking human faces!

Autoencoders are a very useful tool, particularly in regards to allowing image recognition when images are noisy, this is done by reconstructing the image. It was in 1986 when they were theorised by D.E. Rumelhart, G.E. Hinton, and R.J. Williams [14].

1.3 Review of literature

The mathematics behind neural networks, convolutional neural networks, autoencoders and generative adversarial networks are already available. A book with a wide range of deep learning explanations is the MIT Deep Learning book [2] which is the main literature for theory we will be using. They find the mathematical equations behind these techniques and the algorithms used in their designs.

To demonstrate these techniques we need some practical resources. We will also being using the Tensorflow website since it contains examples and explanations of a vast number of functions used in our code. One tutorial [5] used one of the data sets we are going to use but used a neural network as opposed to a convolutional neural network. It achieved 91 percent accuracy. One could say the addition of convolutional layers would improve these results since it may identify more features.

The Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow [4] is one of the greatest books for learning machine learning, it explains a lot of theory behind the techniques, and they demonstrate this using different python libraries. This has a general scope with some specific examples. Autoencoders are also demonstrated in here showing strong results.

The conference paper by Luke Metz, Soumith Chintala and Alec Radford on deep generative neural networks use a generative network to generate faces, their results are a little terrifying but you can still identify faces the network has created:

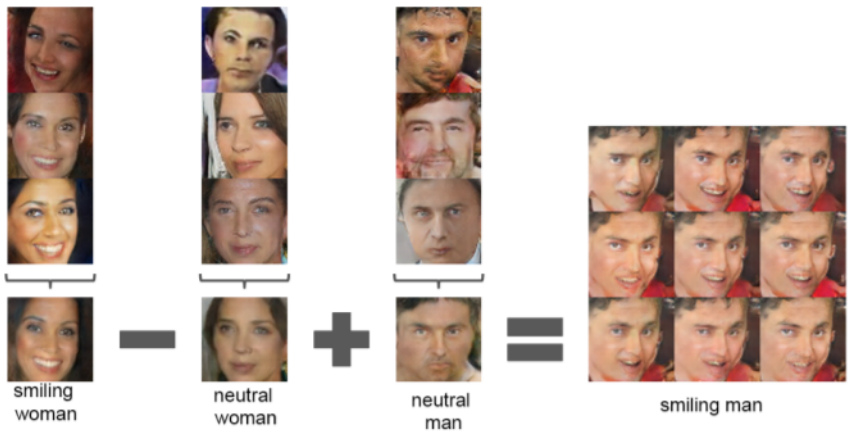


Figure 1.1: New faces generated using a generative adversarial network[16 , Fig.7]

For two simple networks this is impressive.

A real world example which would be interesting to look at is the ”Medical Image Classification with Convolutional Neural Network” paper from the International Conference on Control, Automation, Robotics and Vision [3]:

Qing Li and her colleagues at the University of Sydney made use of a CNN they developed by classifying a lung into categories, these categories being if a lung is diseased and what the disease is. Their method interestingly only has one convolutional layer and fully connected network of 100-50-5 neurons in each layer at the end. The reasoning behind only having one convolutional layer is they don’t need to extract a higher number of features.

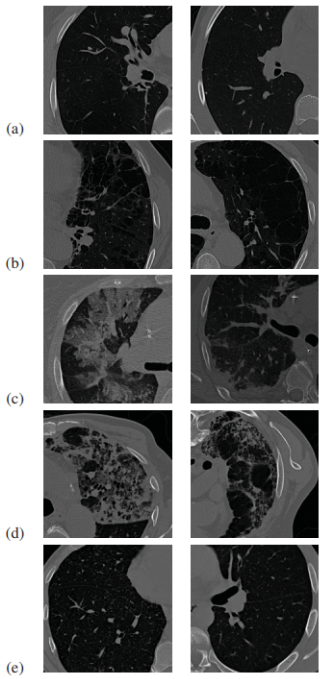


Figure 1.2: Images with disease diagnosis: a: none, b: emphysema, c: ground glass, d:fibrosis and e: micronodules [3,Fig.1]

They compared their results to other methods, theirs outperformed others showing a convolutional neural network is a very good technique for classification. They

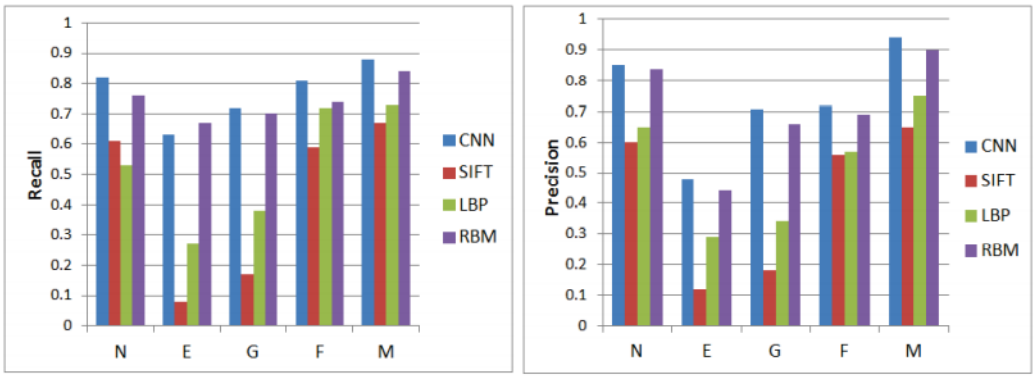


Figure 1.3: CNN compared to other algorithms [3, Fig. 4]

used recall and precision to evaluate their network. Recall is the true positives by the predicted results and precision is true positives by the actual results. This is a better way to evaluate medical images. These are their equations [20].

$$Recall = \frac{TP}{TP + FN} \tag{1.1}$$

$$Precision = \frac{TP}{TP + FP} \tag{1.2}$$

Chapter 2

Neural Networks

2.1 Biological Neuron

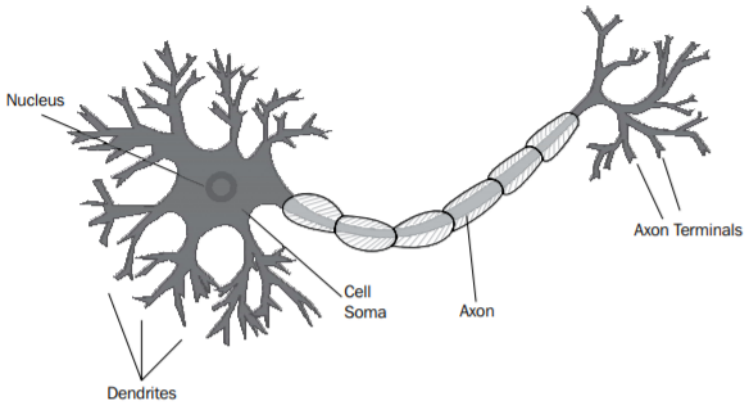


Figure 2.1: Biological Neuron [15,pp.126]

According to [15] a biological neuron contains dendrites which receive inputs and apply weights from chemical signals passed from other nearby neurons. The cell soma totals these signals, and are then sent through the axon and has an activation function applied to it which determines if a signal is passed on to other nearby signals.

2.2 Artificial Neuron

A simple artificial neuron similarly contains an input layer, summation function, activation function and an output layer.

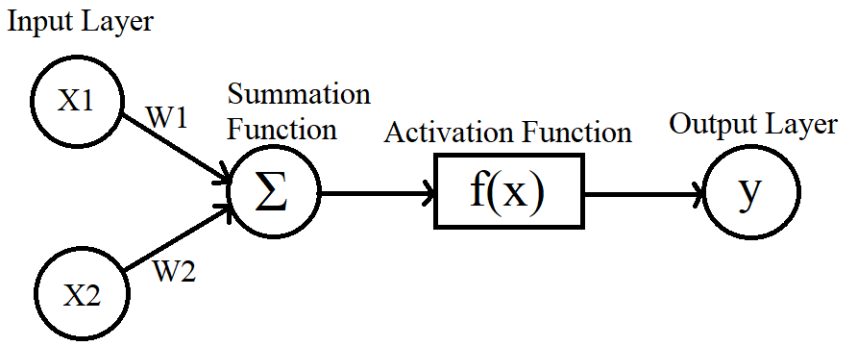


Figure 2.2: Simple Artificial Neuron

x1 and x2 are inputs.
w1 and w2 are weights.

We learn from [1] that the inputs are multiplied by the weights, the product is fed into the summation function which collects all weighted inputs. This number is passed to the activation function which outputs the final value for our neuron. In reality usually more than 2 input nodes and weights are used for each neuron.

2.3 Activation Functions

Activation functions convert the weighted sum of input signals of a neural network to an output signal.

Here are some popular activation functions which appeared in [1]:

Linear	Sigmoid	Tanh	ReLU
$f(x) = cx$	$f(x) = \frac{1}{1+e^{-x}}$	$f(x) = \frac{2}{1+e^{-2x}} - 1$	$f(x)$ = 0 if x<0, x if x≥0
Range: $-\infty \to \infty$	Range: $0 \to 1$	Range: -1 $\to$ 1	Range: 0 $\to$ Max(x)
Output layer	Output layer	Hidden layers	Hidden layers

I would argue when choosing our activation functions in our convolutional neural network, ReLU for the middle layers and linear for the output layers are the best options due to it's function being easier to calculate, therefore reducing the time it takes

to computate. I decided to investigate this by simulating the time it takes to complete 1,000,000 calculations from the function. The precise results can be seen in section 8.1 however I did discover that ReLU and linear functions are much quicker than sigmoid and tanh functions.

Another reason for ReLU is it solves the issue of unstable gradients which would significantly hinder our results [4].

2.4 Neural Network

A neural network is more complicated than this since it may have multiple hidden layers of neurons: The algorithm of an artificial neural network is explained further

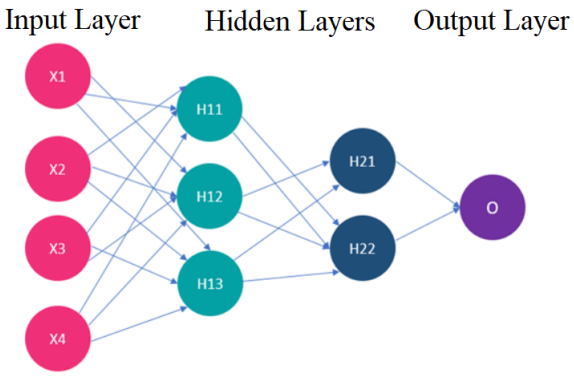


Figure 2.3: Simple Artificial Network [1]

in section 2.5 and 2.6 since it has two parts to it's learning process. The brain is a very complex neural network and is the inspiration for the algorithm's design.

2.5 Propagation

From [21] we learn about a neural network's main two parts in it's training process (propagation and backpropagation) and can show the mathematical equations used in them. Our inputs x_1 and x_2 are weighted, summed and inputted into the activation function. The resulting values go through the same process with the weights and activation functions for the next layer. This can be represented using the equations below:

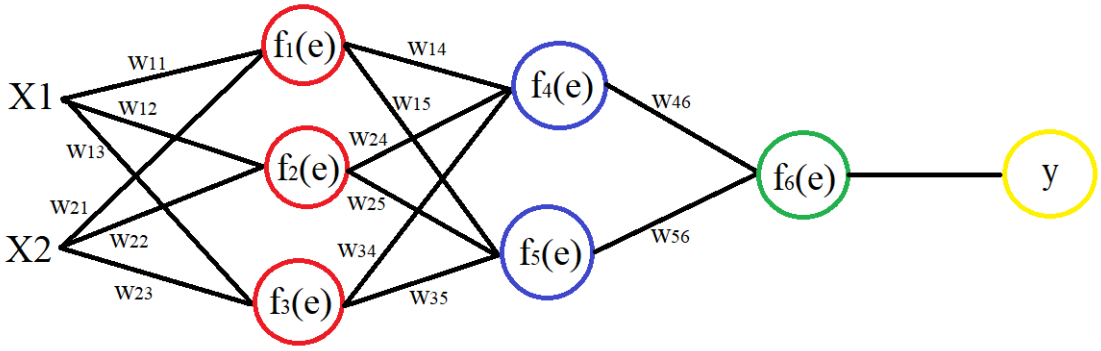


Figure 2.4: Propagation in Neural Network

$$y_1 = f_1(w_{11}x_1 + w_{21}x_2) \tag{2.1}$$

$$y_2 = f_2(w_{12}x_1 + w_{22}x_2) \tag{2.2}$$

$$y_3 = f_3(w_{13}x_1 + w_{23}x_2) \tag{2.3}$$

$$y_4 = f_4(w_{14}y_1 + w_{24}y_2 + w_{34}y_3) \tag{2.4}$$

$$y_5 = f_5(w_{15}y_1 + w_{25}y_2 + w_{35}y_3) \tag{2.5}$$

$$y_6 = f_6(w_{46}y_4 + w_{56}y_5) \tag{2.6}$$

Each y value is the output after each activation function. For example y_2 is the output value from $f_2(e)$.

2.6 Backpropagation

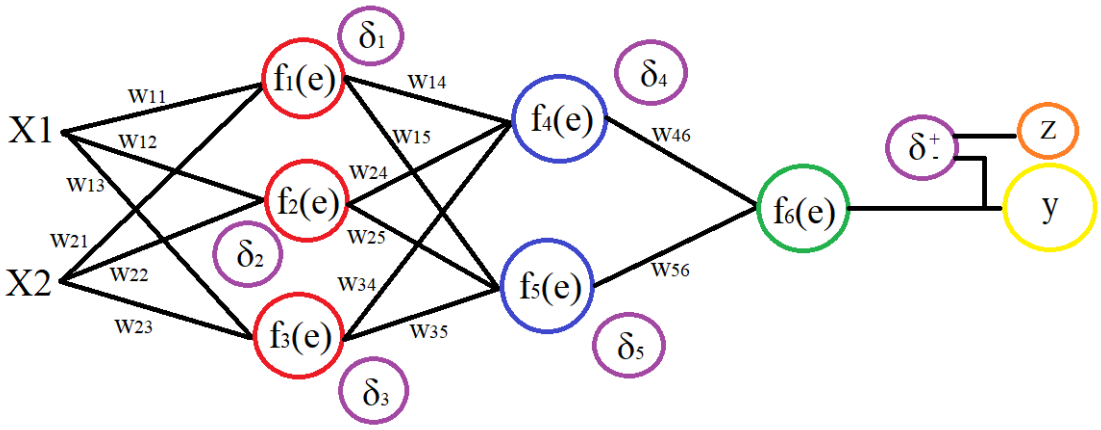


Figure 2.5: Backpropagation in Neural Network

In our algorithms next step, the output signal of the network y is compared with the desired output value. The difference is the error signal δ.

$$\delta_1 = w_{14}\delta_4 + w_{15}\delta_5 \tag{2.7}$$

$$\delta_2 = w_{24}\delta_4 + w_{25}\delta_5 \tag{2.8}$$

$$\delta_3 = w_{34}\delta_4 + w_{35}\delta_5 \tag{2.9}$$

$$\delta_4 = w_{46}\delta \tag{2.10}$$

$$\delta_5 = w_{56}\delta \tag{2.11}$$

As you can see you multiply δ by the weights and add any δ values from the layers after. Now the weights can be updated. In the formulas below $\frac{df(e)}{de}$ represents the derivative of the neuron activation functions (which weights are updated), and n represents the ”learning rate”.

$$w'_{11} = w_{11} + n\delta_1 \frac{df_1(e)}{de} x_1 \tag{2.12}$$

$$w'_{12} = w_{12} + n\delta_2 \frac{df_2(e)}{de} x_1 \tag{2.13}$$

$$w'_{13} = w_{13} + n\delta_3 \frac{df_3(e)}{de} x_1 \tag{2.14}$$

$$w'_{21} = w_{21} + n\delta_1 \frac{df_1(e)}{de} x_2 \tag{2.15}$$

$$w'_{22} = w_{22} + n\delta_2 \frac{df_2(e)}{de} x_2 \tag{2.16}$$

$$w'_{23} = w_{23} + n\delta_3 \frac{df_3(e)}{de} x_2 \tag{2.17}$$

$$w'_{14} = w_{14} + n\delta_4 \frac{df_4(e)}{de} y_1 \tag{2.18}$$

$$w'_{15} = w_{15} + n\delta_5 \frac{df_5(e)}{de} y_1 \tag{2.19}$$

$$w'_{24} = w_{24} + n\delta_4 \frac{df_4(e)}{de} y_2 \tag{2.20}$$

$$w'_{25} = w_{25} + n\delta_5 \frac{df_5(e)}{de} y_2 \tag{2.21}$$

$$w'_{34} = w_{34} + n\delta_4 \frac{df_4(e)}{de} y_3 \tag{2.22}$$

$$w'_{35} = w_{35} + n\delta_5 \frac{df_5(e)}{de} y_3 \tag{2.23}$$

$$w'_{46} = w_{46} + n\delta_1 \frac{df_1(e)}{de} y_4 \tag{2.24}$$

$$w'_{56} = w_{56} + n\delta_1 \frac{df_1(e)}{de} y_5 \tag{2.25}$$

As mentioned this research was from [21]. These weights are updated for a given value of epochs. The more times this process is iterated the more accurate results you will get for the data set provided.

Chapter 3

Convolutional Neural Networks

3.1 Convolutions

From the MIT Deep Learning book [2] we can understand what a convolutional neural network is and the equations within one. A convolution is a specialized kind of linear operation. These are used in conventional neural networks in place of a general matrix multiplication in one or more layers in the network.

Convolutions for two-dimensional CNNs are: [2,eq. (9.4)]

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{3.1}$$

A convolution is commutative, meaning equation 3.1 can be rewritten as: [2,eq. (9.5)]

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) \tag{3.2}$$

where K represents the kernel which is updated the the training process, I represents the 2D image inputted into our network with pixel positions(m,n), S is the feature map/output with pixel positions (i,j). The feature map is the result of one filter which can give our network key features of an image to learn from for future recognition.

3.2 Convolutional Neural Network Architecture

In a convolutional neural network, an image is used as the input which goes through numerous operations. In this example we have a convolutional layer followed by max pooling and then a fully connected neural network.

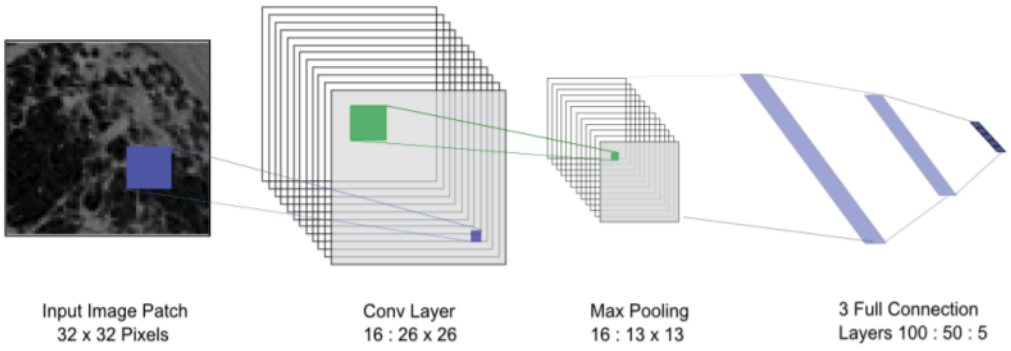


Figure 3.1: CNN Architecture Example [3,Fig.2]

We need to question the dimensions of our layers in our network. A higher quantity of filters in our convolutional layers will extract more information. Handwritten digits don't require huge numbers of filters however more complex images may, particularly if you're classifying images which look similar regardless of it's class.

Filter/kernal size is also an important aspect to consider, since the resolutions of our handwriting set is only 28x28 we can argue our kernal size should also be small to allow us to extract features.

Finally we have a dense layer which is the fully connected neural network, the final dense layer needs to be equal to the amount of possible categories. For example, since our handwriting data set has numbers 0 to 9, we have 10 possible outputs.

3.3 Convolutional Layers and forward propagation

A filter is applied across the input image for the convolutional layer, we can produced an example of the filter being applied at a certain point using equation 3.1:

8	4	2	5	6	1
5	8	1	4	7	2
2	3	9	1	6	1
4	6	4	5	6	1
7	5	8	4	7	2
5	2	5	1	9	1
3	1	6	3	1	3

5	6	1
4	7	2
1	9	1

$*$

1.1	0.9	1
1.05	1	0.95
1.1	0.9	1

$=$

21	25	17
24	35	18
20	24	20

Figure 3.2: Image convolution example

This process is repeated for many filters, depending on the image size. We can argue small simple images won't contain as many features as large complex images so it is case dependent.

Once a convolution has been calculated an activation function can be applied on the output to emphasise it's features.

3.4 Convolutional Layer Backpropagation

From [18] we learn how a backpropagation works in convolutional layers using the chain rule:

Output from a filter:

$$Y = X_{i,j} F_{i,j} \tag{3.3}$$

We can then find the derivatives with respect to filter F:

$$\frac{\partial Y_{i,j}}{\partial F_{i,j}} = X_{i,j} \tag{3.4}$$

Then using the chain rule we can calculate the gradient to update the filter:

$$\frac{\partial L}{\partial F} = \frac{\partial L}{\partial Y} * \frac{\partial Y}{\partial F} \tag{3.5}$$

We repeat this for every element in the filter:

$$\frac{\partial L}{\partial F_i} = \sum_{k=1}^{m} \frac{\partial L}{\partial Y} * \frac{\partial Y}{\partial F_i} \tag{3.6}$$

We can then take a similar method to our neural networks in section 2 and update the filters by multiplying this gradient by the current filter and learning rate and adding it to our old kernel to update it.

Weights are shared in convolutional neural networks since a filter's parameters are

repeated over an image. We should also choose a relevant loss function when back-propagating, cross-entropy loss is commonly used [22]:

$$CE = -\sum_{i}^{C} y_i' log(y_i) \tag{3.7}$$

Where y_i is the predicted probability of an image belonging to a class and y_i' is the actual probability for that class (which is generally 0 or 1) for C classes.

3.5 Pooling Layers

Max pooling are used in CNNs after a convolutional layer as seen in the architecture in figure 3.1, this is done to reduce the memory usage, and the number of parameters (thereby limiting the risk of overfitting). [4] It would therefore be a good idea to include this in our model to increase accuracy and decrease the run time.

The max pooling operation takes the highest value for a specified section e.g each 2x2 pixels in a feature map:

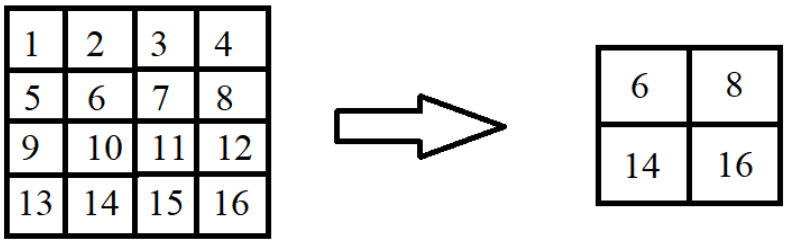


Figure 3.3: Max pooling example

3.6 Max Pooling Layer Backpropagation

Since max pooling layers doesn't have any weights to be updated, we simply use a mask to backpropagate. Where the location of the highest number where the filter was applied is stored and used to pass back values in backpropagation.

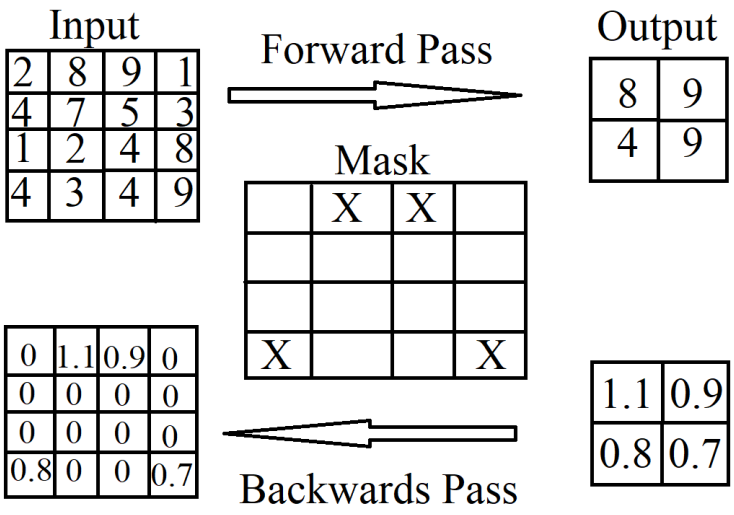


Figure 3.4: Backpropagating max pooling layer

Chapter 4

Optimization Techniques

4.1 Optimizers

Optimizers are the algorithms used to reduce our error. This is done by adjusting the learning rate and the weights. There are many optimizers which can be utilized so we will be understanding and evaluating the ones most commonly applied in machine learning. Research on optimizers will be with the MIT Book [2].

4.2 Gradient Descent

Gradient descent is the basic algorithm for calculating the parameters in neural networks and other machine learning techniques. Gradient-based optimization maximises or minimises an objective function. Minimizing the function is known as the cost/loss/error function.

We often denote the value that optimizers a function with a *: [2]

$$x* = argmin f(x). \tag{4.1}$$

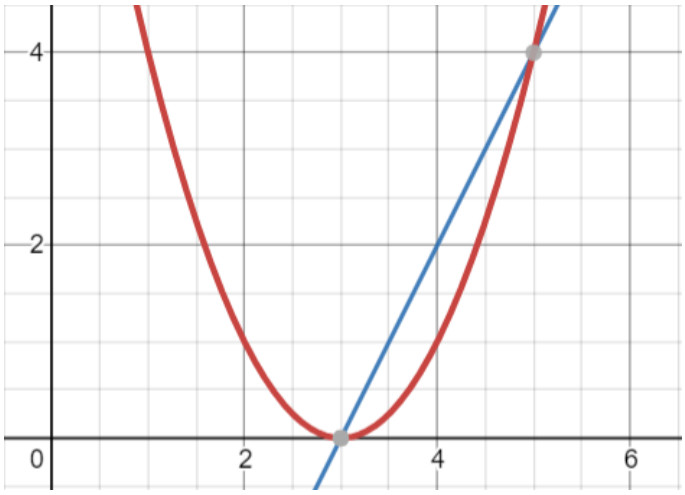


Figure 4.1: Gradient descent algorithm virtualized created using Desmos [9]

The red line is:

$$f(x) = (x-3)^2. \tag{4.2}$$

The blue line is:

$$f'(x) = 2(x-3). \tag{4.3}$$

With this example we see when x $<$ 3, f'(x) $<$ 0. When x $>$ 3, f'(x) $>$ 0. And when x = 0 f'(x) = 0 meaning we have a stationary point where the gradient descent algorithm stops since the minimum.

f'(x) gives the derivative of f(x). As x changes we can evaluate the slope of f(x) at any point. Meaning it can identify how to scale a small change in the input to obtain the corresponding change in the output: [2]

$$f(x+\epsilon) \approx f(x) + \epsilon f'(x). \tag{4.4}$$

Where the epsilon represents the learning rate. This equation indicates how to change x in order to make a small improvement in y, this is known as minimizing a function which is used improve our machine learning predictions: [2]

$$f(x - \epsilon sign(f'(x)) < f(x) \tag{4.5}$$

when changing x in tiny magnitudes iteratively using the opposite sign of the derivative. Repeating these steps until our minimum is located is called gradient descent.

4.3 Stochastic Gradient Descent

Many deep learning models now use stochastic gradient descent due to speed benefits. This is because stochastic gradient descent utilizes a small set of samples, also known as mini-batches.

The gradient is estimated like so: [2]

$$\hat{g} = \frac{1}{m}\nabla_\theta \sum_i \text{L(f(x}^i; \theta) y^i) \tag{4.6}$$

where m is the mini-batch size and θ is the weight.

The algorithm: [2]

1. Firstly we get our initials values. The iteration count will be set to 1, the weight will be received and our learning rates list is initialized. In practical use we have multiple learning rates as they slowly decrease with each iteration to allow them to hone in on a specific value.
2. Sample a minibatch of examples from the training set x_m with corresponding targets y_i.
3. Calculate the gradient using equation 4.6.
4. Update weight: $\theta \leftarrow \theta - \epsilon_k \hat{g}$
5. Update iteration counter: k $\leftarrow$ k + 1
Repeat steps 2 to 5 until convergence (until our improvements are minuscule).

4.4 Adaptive Moment Estimation

Adaptive Moment Estimation(ADAM) is the quickest of our optimizers, it's only disadvantage is convergence issues are more likely.

The algorithm: [2]

1. Firstly we require to set our moments, s = 0 and v = 0. Our timestep is set to 0. We receive our objective function f(θ) and our initial parameters θ.We have exponential decay rates for moment estimates β_1 and β_2 and the learning rate ϵ.
2. Update iteration counter: t $\leftarrow$ t + 1
3. Calculate gradient: $g_t \leftarrow \nabla_\theta \, f_t(\theta_{t-1})$
4. Update first moment: $s_t \leftarrow \beta_1 s_{t-1} + (1 + \beta_1) g_t$
5. Update second moment: $v_t \leftarrow \beta_1 v_{t-1} + (1 + \beta_2) g_t \odot g_t$
6. Create unbiased estimate: $\hat{s}_t \leftarrow \frac{\hat{s}_t}{1-\beta_1^t}$
7. Create unbiased estimate: $\hat{v}_t \leftarrow \frac{\hat{v}_t}{1-\beta_2^t}$
8. Update Objective parameters: $\theta_t \leftarrow \theta_{t-1} - \frac{\epsilon \hat{s}_t}{\sqrt{\hat{v}_t} + \lambda}$

Repeat steps 2 to 8 until convergence (until our improvements are minuscule).

Chapter 5

Autoencoders

We will again be using the MIT Book [2] for reasearch into the mathematics behind autoencoders.

5.1 Introduction

The assignment of autoencoders are to almost duplicate inputted data to it's output.

An autoencoder has two segments [2]:

1. The encoder h = f(x).
2. The decoder r = g(h).

h is the hidden layer and the code which represents the input. g is the reconstruction function whilst f is the encoder function.

An autoencoder shouldn't be 100 percent accurate, otherwise they're not useful. It should be approximate.

5.2 Denoising autoencoders

Denoising autoencoders has a noisy version of the original data inputted as opposed to original data, our network learns to predict the original data from it's noisy input. This could be used for training noisy images to be recognised.

Figure 5.1 shows the denoising autoencoders learning process. C($\tilde{x} \mid x$) is the corruption process, which represents a conditional distribution over corrupted/noisy images, $\tilde{x}$, given an image x. The next stage is to develop a reconstruction distribution $P_{reconstruction}(\tilde{x} \mid x)$ estimated from training pairs (x, $\tilde{x}$) as follows: [2]

Firstly we get a training image x from the training data.

Then we get the noisy image $\tilde{x}$ from C($\tilde{x} \mid x = x$)

Finally we use (x, $\tilde{x}$) for calculating the reconstruction distribution $P_{reconstruct}(\tilde{x} \mid x)$ = $P_{encoder}(x \mid h)$ where h=f($\tilde{x}$) from the encoder and $P_{decoder}$ g(h).

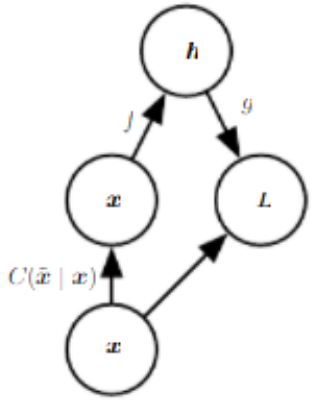


Figure 5.1: Denoising autoencoder training procedure [2, Fig.14.3]

$$Loss(L) = -logp_{decoder}(x \mid h = f(\tilde{x})) \tag{5.1}$$

$p_{decoder}$ is usually a factorial distribution with mean parameters from the feedforward network g. [2]

We can now perform the optimization technique SDG on the following expectation ($\tilde{p}_{data(x)}$ being the training distribution): [2]

$$- E_{x \sim \tilde{p}_{data(x)}} E_{x \sim C(\tilde{x} \mid x = x)} logp_{decoder}(x \mid h = f(\tilde{x})) \tag{5.2}$$

Parameters are trained to minimize the loss, to have the reconstruction as close as possible to x. [19]

Chapter 6

Generative adversarial network

6.1 Introduction

A generative adversarial network is a machine learning technique which contains two neural networks competing to see if one neural network (the generator) can trick the other neural network (the discriminator). The aim is for both networks to become developed but the generator should end the stronger of the two to create realistic looking images which the discriminator believes to be real.

We will be using the paper [13] to look at the mathematics of GAN. Below is an equation for generative adversarial networks: [13, eq.(1)]

$$\min_{G}\max_{D} V(D,G) = E_{x\sim p_{data}(x)}\, logD(x) + E_{z\sim p_z(z)}\, log(1 - D(G(x))) \qquad (6.1)$$

G generates x images for D to classify if an image is real or not. D(x) is the probability the discriminator correctly determines if an image is fake or not. The generator wants 1-D(G(z)) to equal 0 whilst the discriminator wants it to equal 1. p(z) is the prior probability distribution of the noise variables.

As opposed to letting D maximize first, we should alternate between optimizing the D and G in our routine to allow the other network to gain intelligence and keep up with the other. D should be permitted k steps and then G has one.

The paper produced the following results:

Equation 6.1 may have an unfair advantage towards the discriminator. This is because the generator has very deficient results it is easy for discriminator the deduce the true images from the fake. 1-D(G(z)) inflates. To improve learning of G in early training we can train for G to maximize D(G(z)) since it provides stronger gradients at the early stages of development.

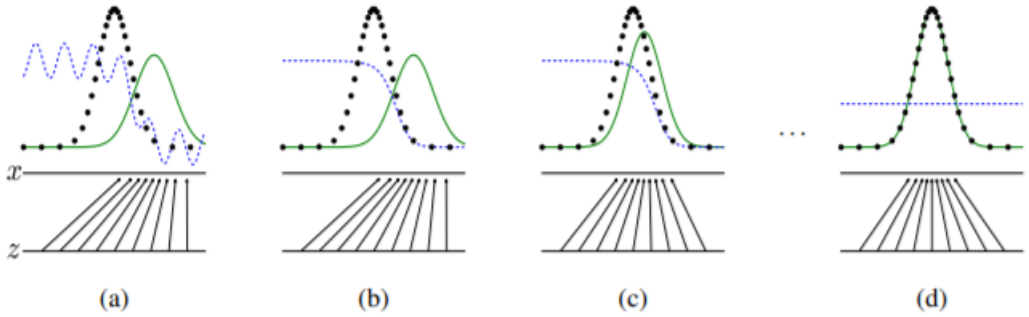


Figure 6.1: GAN results [13, Fig.1]

6.2 GAN algorithm

[13] 1. Sample minibatch of m noise images z(1), . . . , z(m) from noise prior pg(z).
2. Sample minibatch of m images x(1), . . . , x(m) from data generating distribution pdata(x).
3. Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} (logD(x^i) + log(1 - D(G(z^i)))) \tag{6.2}$$

(repeat steps 1-3 for k steps)
4. Sample minibatch of m noise samples z(1), . . . , z(m) from noise prior pg(z)
5. Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} log(1 - D(G(z^i))) \tag{6.3}$$

(repeat all steps for number of training repetitions/epochs).

Chapter 7

Implementation

7.1 Data set

The first data set we will be using is the MNIST Handwriting data set since there is plenty of train and test data and it is a good classification problem.

There are 60,000 train images and 10,000 test images. 60,000 train images will prevent overfitting, and 10,000 images should accurately measure our accuracy since it will include digits written in a vast amount of ways just like real life.

The second data set we can use is the MNIST Fashion data set since it has the same dimensions as our handwriting data set so we can directly compare results of the same network on two different data sets.

7.2 Desired Discoveries

We should therefore have eight desired discoveries:

Firstly we should have our calculated times for each activation function to test which is quickest.

Secondly we should also be able to visualise our data by displaying an image and displaying the quantity of each class.

Thirdly we should be able to classify new images of digits which our trained model has not yet seen.

Forthly we should have high accuracies which we can analyse against the amount of epochs, we should see the accuracy plateau to a high result.

Fifthly we should compare our accuracies for our two best optimizers identified from our research (SGD and ADAM).

Sixthly we should test our CNN on a new data set to see how adaptive it is.

Sevently our autoencoders should successfully denoise images which may have been taken on a poor camera.

Finally we should use our networks with other data sets to analyse how effective and adaptive they are on different images.

7.3 Code Explanation

7.3.1 Convolutional Neural Network

With help from the TensorFlow website to compile the cross-entropy loss function and record accuracy [6], I managed to implement the neural networks in Python and plot graphs for analysis.

After importing our numpy and TensorFlow libraries, we receive our data set which can be done using:

(train_images, train_labels) , (test_images, test_labels)= tf.keras.datasets.mnist.load_data().

We then use numpy to reshape our data into a format which can be entered into our CNN module:

train_images = train_images.reshape(train_images.shape[0], 28, 28, 1).astype('float32')
test_images = test_images.reshape(test_images.shape[0], 28, 28, 1).astype('float32')

We then normalise our data to speed up the convergence of the model. Since RGB colours are from 0 to 255, we divide our data by 255 to get our data from 0 to 1.

Now we can build our model, we firstly declare our model us to the add layers into our model. model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28,28,1))) adds a convolutional layer with a 3x3 kernal size of 32 filters with a ReLU activation function and allows an input of a 28x28 image and outputs a tensor of outputs which is equal to the number of filters.

We follow a Conv2D with a max pooling layer as noted in our CNN architecture research. model.add(layers.MaxPooling2D((2, 2))) is only 2x2 since our image is

quite small in resolution (28x28).

We add another convolutional layer, then another max pooling layer and then one final convolutional layer. This means we have three convolutional layers in total which is ideal since it allows one hidden layer which is recommended for image classification, more than one hidden layer offers little benefit.

We then flatten our data to allow it to be inputted into a densely connected neural network using layers.Dense(). The output from the dense layer is calculated:

$$output = f(input * kernel + bias) \tag{7.1}$$

The kernel in this case is the weight matrix. And f the activation function.

We then compile our model and state the optimizer we are using, for example Adaptive Moment Estimation (ADAM) and finally train our model with our train data for a stated number of epochs and evaluate it with the test data. Finally we can plot this accuracy against epochs.

7.3.2 Autoencoder

With assistance from online tutorials on the TensorFlow website [7] I implemented an autoencoder of the handwriting data set.

After importing relevant libraries and loading and preprocessing the data like in the convolutional neural network, we add noise to our train and test images to train our autoencoder and evaluate it: (0.2*tf.random.normal(shape=train_imgs.shape)) + train_imgs. We then add our encoder and decoder functions as discoursed in section 5.1. This involves convolution layers in the encoder function and the transpose of convolutional layers in the decode function.

We then compile and train our network with our train set and test our network using the test set using the principles from section 5: autoencoder.decoding(autoencoder.encoding(test_imgs).numpy()).numpy().

We can plot our noisy image against it reconstructed as well as use our CNN to see how accurate the reconstructed images are.

7.3.3 Generative Adversarial Network

To acquire results for generative neural networks is quite difficult, there is a network from the TensorFlow website I based mine on more heavily than I did the previous two websites. [8]

We again start by importing our libraries and loading and preprocessing the data as done previously. Next we define our first network, the generator to create believable images, and the discriminator to identify if an image is fake or real. We then define the discriminator loss and generator loss. Cross-entropy is how we calculate the loss and the Adam optimizer is the optimization choice. We train our networks and images are created at each epoch.

7.3.4 Activation function comparison

We define a begin variable t1 and end variables t2 after an activation function has been calculated 1,000,000 times. t2-t1 is calculated for each function showing the time it took to calculate linear, sigmoid, ReLU and tanh functions.

Chapter 8

Results

8.1 Activation function simulation comparison

Below we have the results of our simulation for calculating the exact time it takes to compute 1,000,000 iterations of each function:

Linear time: 0.091757 seconds

Sigmoid time: 1.608700 seconds

Tanh time: 1.834138 seconds

ReLU: 0.069825 seconds

This demonstrates presuming sigmoid and tanh functions don't have an advantage in terms of accuracy linear and ReLU are better functions for our network. In fact ReLU often provides better accuracies for image recognition.

8.2 Visualising

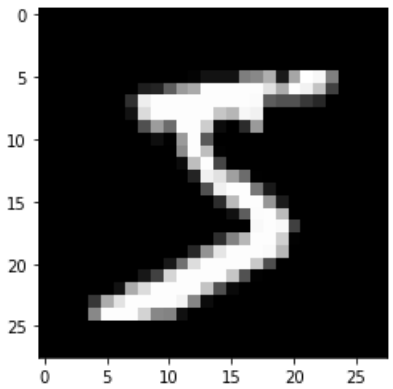


Figure 8.1: Image example (5)

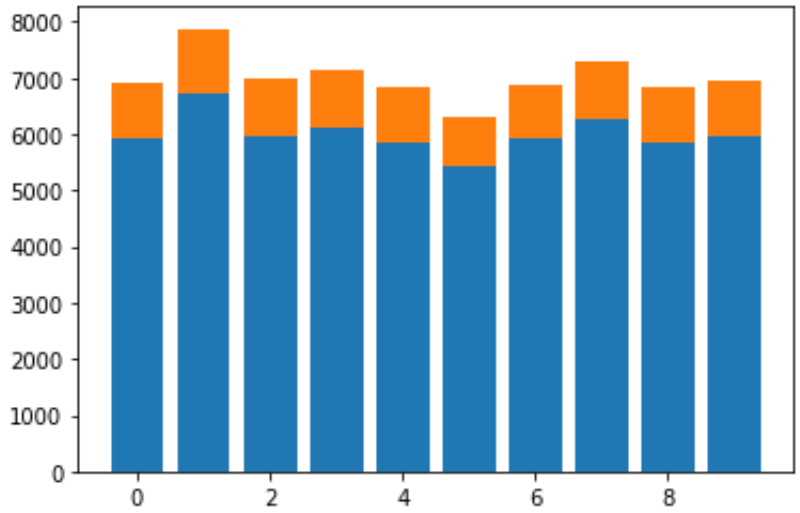


Figure 8.2: Quantity of number in data set vs number. Blue: train set, orange: test set

This section is used to visualise our data. Figure 8.1 shows that we can load our images. Figure 8.2 shows the quantity of each number and if they belong to the train set or the test set. 1 has the most images meaning it will likely be the easiest to predict particularly when you factor the simplicity of the number 1. We may find the number 5 is the weakest link lowering our accuracy of our model.

8.3 Predicting

Here we out put our 1st and 500th test image as it's true value and it's predicted value, as you can see both predictions are correct!

```
True: 7
Predicted: 7


True: 6
Predicted: 6
```

Figure 8.3: Predictions

In regards to how accurate the entire data set is we can analyse this in our next section which shows the accuracy over each epoch.

8.4 How the amount of epochs effects accuracy

Our validation set's accuracy plateaus when the epoch count = 6. Whilst our training data set accuracy is continuously increasing until it converges to almost 1. The validation set is more important since the network will see new data in the real world and is unlikely to see a number written in the exact same way as in the test set.

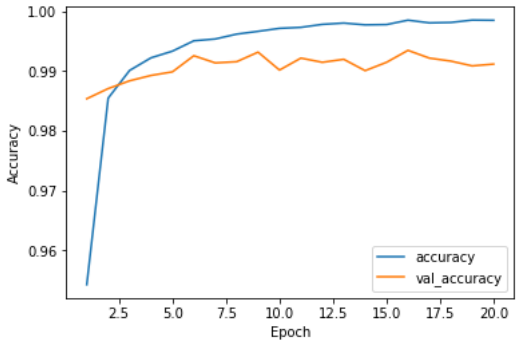


Figure 8.4: Test set accuracy and validation set accuracy vs epoch number (ADAM)

8.5 Optimization comparison

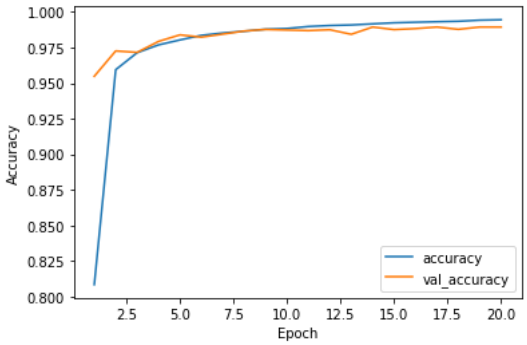


Figure 8.5: Test set accuracy and validation set accuracy vs epoch number (SDG)

As was theorised in our research, stochastic gradient descent was slower at optimizing our model as well was not optimizing as fully as adaptive moment estimation. ADAM plateaued at around 99.3 percent compared to SDG at 98.9 percent.

8.6 How adaptive is our CNN on a different data set

Next we calculate our validation accuracy of the fashion data set. It's a little lower than our previous data set at 91.07 percent, 8.23 percent less than our handwriting data set.

Comparing this method (CNN) to normal neural networks [5], we see our results are very similar, from this I decided to remove a convolutional layer since our network may be too complex for a simple data set. This slightly improves our results by 0.8 percent as seen below:

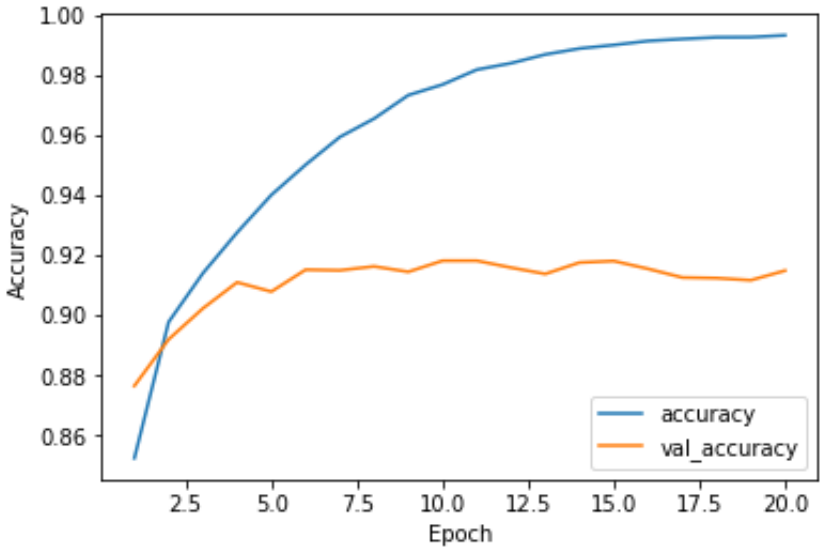


Figure 8.6: Accuracy and validation accuracy of the fashion data set with two convolutional layers

8.7 Autoencoder results

Our autoencoder results show our image is denoised well, figure 8.7 has 20 percent noise added to it which caused accuracies to peak at 99.11 percent. Figure 8.9 also performs well having 97.99 percent accuracy despite having 60 percent noise added to it. Figure 8.11 having almost 100 percent noise added understandably performs worse but still manages to achieve 94.82 percent accuracy as seen below:

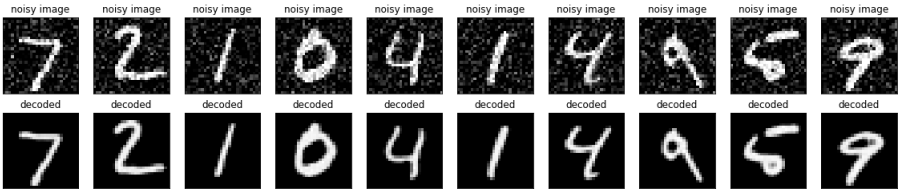


Figure 8.7: Autoencoder reconstruction with noise = 0.2

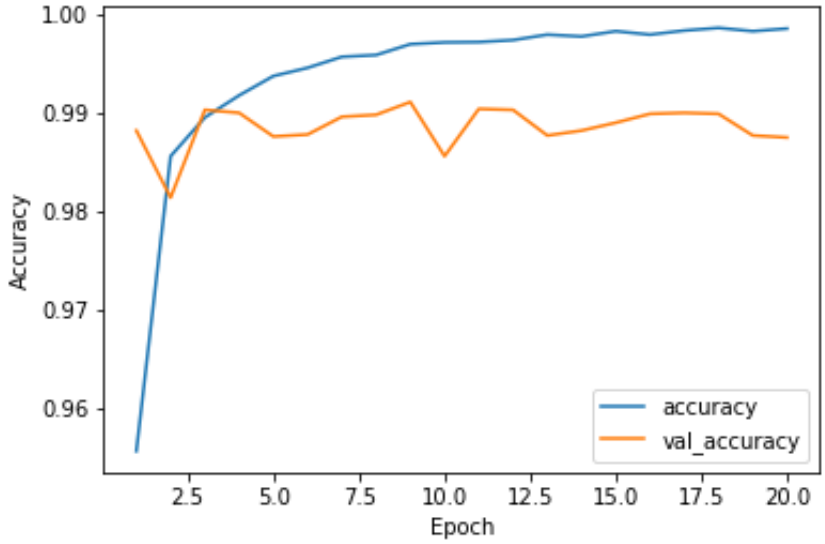


Figure 8.8: Accuracy and value accuracy of reconstructed data on our CNN

Validation accuracy mean: 0.9886 = 98.86 percent
Validation accuracy Max at epoch 9: 0.9911 = 99.11 percent

Prediction 1:

True: 7
Predicted: 7

Prediction 2:

True: 6
Predicted: 6

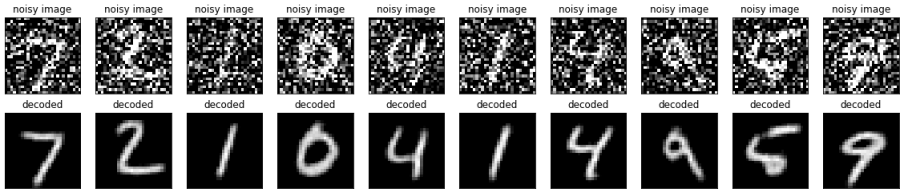


Figure 8.9: Autoencoder reconstruction with noise = 0.6

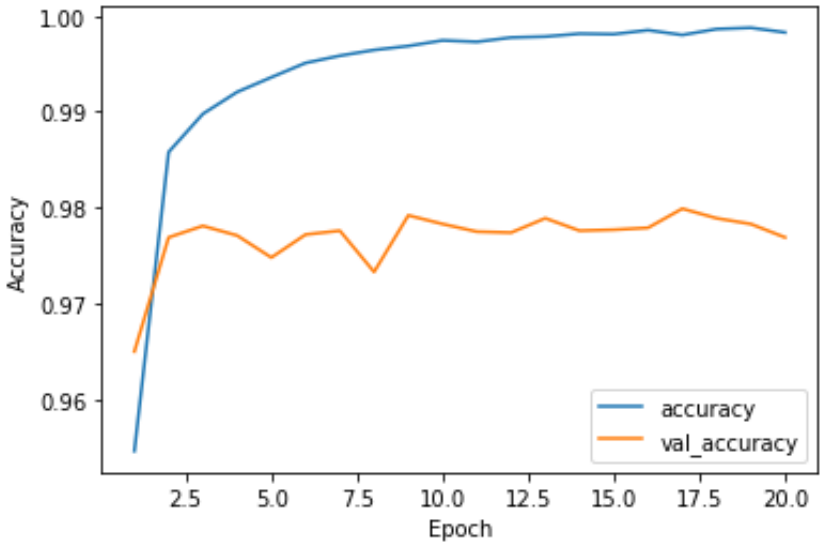


Figure 8.10: Accuracy and value accuracy of reconstructed data on our CNN

Validation accuracy mean: 0.9769333 = 97.69333 percent
Validation accuracy Max at epoch 18: 0.9799 = 97.99 percent
Prediction 1:

True: 7
Predicted: 7

Prediction 2:

True: 6
Predicted: 6

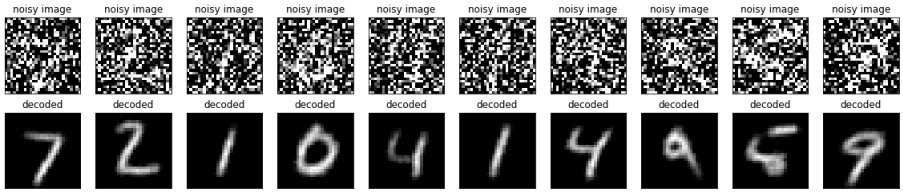


Figure 8.11: Autoencoder reconstruction with noise = 1

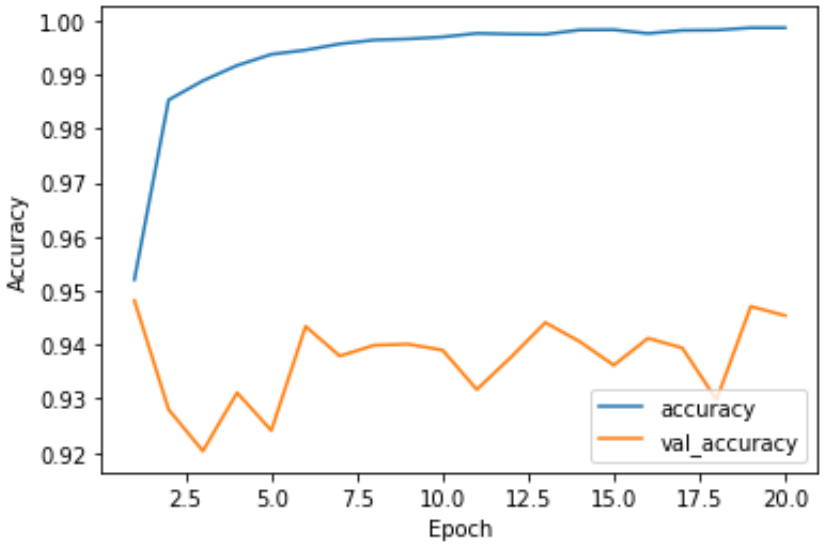


Figure 8.12: Accuracy and value accuracy of reconstructed data on our CNN

Validation accuracy mean: 0.93726 = 93.726 percent
Validation accuracy Max at epoch 1: 0.9482 = 94.82 percent
Prediction 1:

True: 7
Predicted: 7

Prediction 2:

True: 6
Predicted: 8

8.8 Generative Adversarial Networks

Figure 8.13 shows our final results after 50 epochs of generation. As you can see some numbers seems identifiable to us, the best example being the 8 2nd from left at the top. However a few don't such as the 2nd from top to the far left. Our CNN struggles with these numbers only identifying 20 percent showing a user requires a certain handwriting quality to be identified. A bigger data set from people with poorer handwriting may be able to improve this. That being said some numbers are impressive considering a computer has created them.

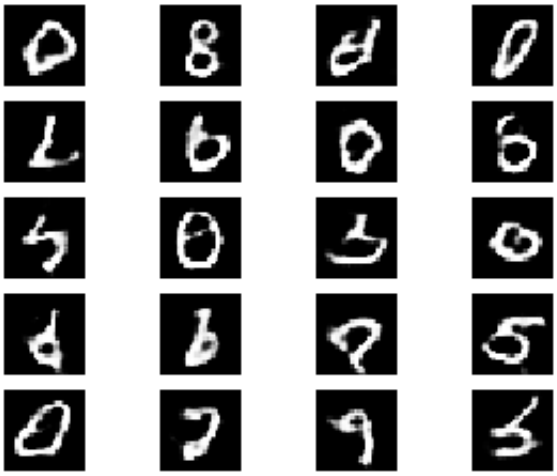


Figure 8.13: Numbers generated from GAN

Chapter 9

Conclusion

9.1 Conclusion

Throughout this project we have understood theoretically, mathematically and practically how deep learning techniques work, specifically on images. Results from our three techniques were largely positive. Convolutional neural networks seem very effective in recognising images through feature extraction. Achieving over 99.3 percent accuracy on the handwriting data set and 91.07 percent for the fashion data set. Our autencoders successfully denoise new images presented to it with very high accuracies even with lots of noise. Our generative adversarial network although not accurate is still impressive in concept and still produces some good images.

Although we have a high accuracy of 99 percent, our predictions may need to be higher in wide scale uses, this could be looked into in the future to see how we can make our model almost flawless from looking at adding more layers, more data or extra features such as the use of autoencoders for anomaly detection.

Further future work could entail developing a wide of programs with image classification. The medical industry to diagnose diseases, such as if a patient has a lump it could be diagnosed as cancer or simply a cyst. Denoising autoencoders may be used in crime to increase the accuracy when identifying suspects on security cameras. Generative adversarial networks would be an interesting prospect for creating new media such as art or music.

The original title for my project was ”Atomic Force Microscopy Image Analysis” where machine learning is used to classify atomic force microscopy images, however after some research I discovered there was nowhere enough data available to produce a functioning model to achieve this to a satisfactory degree. Once this data is available, our model’s dimensions could be could be adapted to these images to assist scientists in quickly classifying these images to a high accuracy.

Chapter 10

Acknowledgements

Firstly I would like to thank my supervisor Matt for his support over the last 9 months through the pandemic in our weekly meetings.

Additionally I'd like to thank my coursemates Nathan, Sam and Dami for being great friends and being there to talk about the course.

I'd also like to thank my friend Myron for being a great friend over the past 15 years and whom I hope to see more of now lockdown is easing.

A huge to thank my housemates Caitlin and Sophie is in order for helping me enjoy this year through the pandemic and keeping me happy and sane!

Next I'd like to thank my Grandad, Grandma and Nan for being lovely grandparents who I'm also looking forward to seeing over the summer.

Thank you to my sister Beth who I wish the best of times to, she will also be studying at the University of Lincoln.

Finally I'd like to thank my Mum and Dad for their constant love and support, I love you both so much.

I shall not forget any of these lovely people and hopefully we will all keep in touch.

Chapter 11

Appendix

11.1 Code

11.1.1 CNN Code

As stated I used [6] to help with the loss and accuracy measurement.

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras import datasets, layers, models (train_imgs, train_labels) , (test_imgs,
test_labels)= tf.keras.datasets.mnist.load_data()
print(len(train_labels))
print(len(test_labels))
print(train_imgs.shape)
print(test_imgs.shape)
plt.imshow(train_imgs[0])
plt.figure()
print(train_labels[0])
train_count = [0,0,0,0,0,0,0,0,0,0]
test_count = [0,0,0,0,0,0,0,0,0,0]
for i in range(len(train_labels)):
    if train_labels[i] == 0:
        train_count[0] += 1
    if train_labels[i] == 1:
        train_count[1] += 1
    if train_labels[i] == 2:
        train_count[2] += 1
    if train_labels[i] == 3:
        train_count[3] += 1
    if train_labels[i] == 4:
        train_count[4] += 1
```

```
    if train_labels[i] == 5:
        train_count[5] += 1
    if train_labels[i] == 6:
        train_count[6] += 1
    if train_labels[i] == 7:
        train_count[7] += 1
    if train_labels[i] == 8:
        train_count[8] += 1
    if train_labels[i] == 9:
        train_count[9] += 1
for i in range(len(test_labels)):
    if test_labels[i] == 0:
        test_count[0] += 1
    if test_labels[i] == 1:
        test_count[1] += 1
    if test_labels[i] == 2:
        test_count[2] += 1
    if test_labels[i] == 3:
        test_count[3] += 1
    if test_labels[i] == 4:
        test_count[4] += 1
    if test_labels[i] == 5:
        test_count[5] += 1
    if test_labels[i] == 6:
        test_count[6] += 1
    if test_labels[i] == 7:
        test_count[7] += 1
    if test_labels[i] == 8:
        test_count[8] += 1
    if test_labels[i] == 9:
        test_count[9] += 1
plt.bar(range(len(train_count)), train_count)
plt.bar(range(len(test_count)), test_count, bottom=train_count)
plt.show()
train_imgs = (train_imgs.reshape(train_imgs.shape[0], 28, 28, 1).astype('float32'))/255
test_imgs = (test_imgs.reshape(test_imgs.shape[0], 28, 28, 1).astype('float32'))/255
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), input_shape=(28,28,1) ,activation='relu'))
```

```
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.summary()
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(10))
model.summary()
model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
optimizer='adam', metrics=['accuracy'])
history = model.fit(train_imgs, train_labels, epochs=20, validation_data=(test_imgs,
test_labels))
epochs = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
plt.figure()
plt.plot(epochs,history.history['accuracy'],label = 'accuracy')
plt.plot(epochs,history.history['val_accuracy'],label = 'val_accuracy')
plt.legend(loc='lower right')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
predictions = model.predict(test_imgs)
print("True: " + str(test_labels[0]))
print("Predicted: " + str(np.argmax((predictions[0]))))
print(" ")
print("True: " + str(test_labels[499]))
print("Predicted: " + str(np.argmax((predictions[499]))))
```

11.1.2 Autoencoder Code

As mentioned some of this code was from the [7] tutorial.

```
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras import datasets, losses, layers, models
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
(train_imgs, train_labels), (test_imgs, test_labels) = tf.keras.datasets.mnist.load_data()
```

```
train_imgs = train_imgs.astype('float32')/255.
test_imgs = test_imgs.astype('float32')/255.
train_imgs = train_imgs[..., tf.newaxis]
test_imgs = test_imgs[..., tf.newaxis]
train_imgs_noisy = (0.2*tf.random.normal(shape=train_imgs.shape)) + train_imgs
test_imgs_noisy = (0.2*tf.random.normal(shape=test_imgs.shape)) + test_imgs
train_imgs_noisy = tf.clip_by_value(train_imgs_noisy,clip_value_min=0.,clip_value_max=1.)
test_imgs_noisy = tf.clip_by_value(test_imgs_noisy,clip_value_min=0.,clip_value_max=1.)
class Restore(Model):
    def __init__(self):
        super(Restore, self).__init__()
        self.encoding = tf.keras.Sequential([
            layers.Input(shape=(28, 28, 1)),
            layers.Conv2D(16, (3, 3),strides=2,padding='same',activation='relu'),
            layers.Conv2D(8, (3, 3), strides=2,padding='same',activation='relu')])
        self.decoding = tf.keras.Sequential([
            layers.Conv2DTranspose(8,(3, 3), strides=2,padding='same',activation='relu'),
            layers.Conv2DTranspose(16,(3, 3), strides=2,padding='same',activation='relu'),
            layers.Conv2D(1,(3, 3), padding='same', activation='relu')])
    def call(self, img):
    restored_img = self.decoding(self.encoding(img))
    return restored_img
autoencoder = Restore()
autoencoder.compile(loss=losses.MeanSquaredError(),optimizer='adam')
autoencoder.fit(train_imgs_noisy, train_imgs,epochs=10,validation_data=(test_imgs_noisy,
test_imgs),shuffle=True)
restored_imgs = autoencoder.decoding(autoencoder.encoding(test_imgs).numpy()).numpy()
plt.figure(figsize=(25, 5))
for i in range(10):
    ax = plt.subplot(2, 10, i + 1)
    ax.axes.xaxis.set_visible(False)
    ax.axes.yaxis.set_visible(False)
    plt.title("noisy image")
    plt.gray()
    plt.imshow(tf.squeeze(test_imgs_noisy[i]))
    bx = plt.subplot(2, 10, i + 10 + 1)
    bx.axes.xaxis.set_visible(False)
    bx.axes.yaxis.set_visible(False)
```

```
    plt.title("decoded")
    plt.gray()
    plt.imshow(tf.squeeze(restored_imgs[i]))
plt.show()
```

11.1.3 Generative Adversarial Network Code

As mentioned some of this code was from the [8] tutorial.

```
import tensorflow as tf
import tensorflow_docs.vis.embed as embed
import glob
import imageio
import matplotlib.pyplot as plt
import numpy as np
import os
import PIL
from tensorflow.keras import layers
import time
from tensorflow.keras import datasets, layers, models
from IPython import display
(train_images, train_labels), (test_images, test_labels) = tf.keras.datasets.mnist.load_data()
train_images = train_images.reshape(train_images.shape[0], 28, 28, 1).astype('float32')
train_images = (train_images - 127.5) / 127.5
test_images = test_images.reshape(test_images.shape[0], 28, 28, 1).astype('float32')
test_images = (test_images - 127.5) / 127.5
train_dataset = tf.data.Dataset.from_tensor_slices(train_images).shuffle(60000).batch(256)
def make_generator_model():
    model = tf.keras.Sequential()
    model.add(layers.Dense(7*7*256, use_bias=False, input_shape=(100,)))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())
    model.add(layers.Reshape((7, 7, 256)))
    assert model.output_shape == (None, 7, 7, 256)
    model.add(layers.Conv2DTranspose(128, (5, 5), strides=(1, 1), padding='same',
use_bias=False))
    assert model.output_shape == (None, 7, 7, 128)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())
```

```
    model.add(layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same',
use_bias=False))
    assert model.output_shape == (None, 14, 14, 64)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())
    model.add(layers.Conv2DTranspose(1, (5, 5), strides=(2, 2), padding='same',
use_bias=False, activation='tanh'))
    assert model.output_shape == (None, 28, 28, 1)
    return model
generator = make_generator_model()
generated_image = generator(tf.random.normal([1, 100]), training=False)
plt.imshow(generated_image[0, :, :, 0], cmap='gray')
def make_discriminator_model():
    model = tf.keras.Sequential()
    model.add(layers.Conv2D(64, (5, 5), strides=(2, 2), padding='same', input_shape=[28,
28, 1]))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))
    model.add(layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))
    model.add(layers.Flatten())
    model.add(layers.Dense(1))
    return model
discriminator = make_discriminator_model()
decision = discriminator(generated_image)
cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)
def discriminator_loss(real_output, fake_output):
    real_loss = cross_entropy(tf.ones_like(real_output), real_output)
    fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
    total_loss = real_loss + fake_loss
    return total_loss
def generator_loss(fake_output):
    return cross_entropy(tf.ones_like(fake_output), fake_output)
generator_optimizer = tf.keras.optimizers.Adam(1e-4)
discriminator_optimizer = tf.keras.optimizers.Adam(1e-4)
checkpoint_dir = './training_checkpoints'
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt")
```

```
checkpoint = tf.train.Checkpoint(generator_optimizer=generator_optimizer, discrim-
inator_optimizer=discriminator_optimizer, generator=generator, discriminator=discriminator)
seed = tf.random.normal([20, 100])
def train_step(images):
    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:            generated_images
= generator(tf.random.normal([256, 100]), training=True)
        real_output = discriminator(images, training=True)
        fake_output = discriminator(generated_images, training=True)
gen_loss = generator_loss(fake_output)
        disc_loss = discriminator_loss(real_output, fake_output)
    gradients_of_generator = gen_tape.gradient(gen_loss, generator.trainable_variables)
    gradients_of_discriminator = disc_tape.gradient(disc_loss, discriminator.trainable_variables)
    generator_optimizer.apply_gradients(zip(gradients_of_generator, generator.trainable_variables))
    discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator, discrim-
inator.trainable_variables))
def train(dataset, epochs):
    for epoch in range(epochs):
        start = time.time()
        for image_batch in dataset:
            train_step(image_batch)
        display.clear_output(wait=True)
        generate_and_save_images(generator, epoch + 1, seed)
        if (epoch + 1) % 15 == 0:
            checkpoint.save(file_prefix = checkpoint_prefix)
        print ('Time for epoch  is  sec'.format(epoch + 1, time.time()-start))
    display.clear_output(wait=True)
    predictions = generate_and_save_images(generator,epochs,seed)
    return predictions
def generate_and_save_images(model, epoch, test_input):
    predictions = model(test_input, training=False)
    fig = plt.figure(figsize=(5, 4))
    for i in range(predictions.shape[0]):
        plt.subplot(5, 4, i+1)
        plt.imshow(predictions[i, :, :, 0] * 127.5 + 127.5, cmap='gray')
        plt.axis('off')
        plt.savefig('image_at_epoch_:04d.png'.format(epoch))
        plt.show()
        return predictions
```

```
predictions = train(train_dataset, 50)
checkpoint.restore(tf.train.latest_checkpoint(checkpoint_dir))
def display_image(epoch_no):
    return PIL.Image.open('image_at_epoch_:04d.png'.format(epoch_no))
anim_file = 'dcgan.gif'
with imageio.get_writer(anim_file, mode='I') as writer:
    filenames = glob.glob('image*.png')
    filenames = sorted(filenames)
    for filename in filenames:
        image = imageio.imread(filename)
        writer.append_data(image)
    image = imageio.imread(filename)
    writer.append_data(image)
embed.embed_file(anim_file)
```

11.1.4 Optimizer speed test

```
import datetime
import numpy
c = 3
t1 = datetime.datetime.now()
for x in range(1000000):
    linear = c*x
t2 = datetime.datetime.now()
print("Linear time: " + str(t2-t1))
t1 = datetime.datetime.now()
for x in range(1000000):
    1/(1+numpy.exp(-x))
t2 = datetime.datetime.now()
print("Sigmoid time: " + str(t2-t1))
t1 = datetime.datetime.now()
for x in range(1000000):
    2/(1+numpy.exp(-x)) - 1
t2 = datetime.datetime.now()
print("tanh time: " + str(t2-t1))
t1 = datetime.datetime.now()
for x in range(1000000):
    if x≥ 0 :
        x
```

$$
\begin{aligned}
&else: \\
&\quad 0 \\
&t2 = datetime.datetime.now() \\
&print("ReLU : " + str(t2 - t1))
\end{aligned}
$$

11.2 Research Plan

11.2.1 Description

Deep learning is a powerful tool in the current age, a tool which can be utilized in the field of image recognition. This project aims to understand mathematically and use computationally, deep learning techniques such as convolutional neural networks, which analyse and classify images.

I will attempt to recognise images in datasets, such as digits in a handwritten digits dataset, and explain how these methods could be applied onto different types of images. For example, atomic force microscopy images, which are images of material surfaces at nanometer resolutions, may be classified into different materials. Further techniques may be investigated and used like autoencoders to denoise images and generative adversarial networks to learn to create realistic images using preexisting images. An evaluation of the accuracy of the programs will be produced to see how reliable these techniques are.

11.2.2 Motivation

Artificial intelligence is of huge interest to me due it's possibilities, once we achieve "general AI", it's said to be humanities final discovery. Image recognition is extremely valuable, it's used for diseases, for example it's an enormous weapon to have against cancer as it can help detect early stages which professional doctors are not capable of. The medical industry is just one of many areas it can influence, the scientific community who attempt to classify AFM images may be helped.

11.2.3 Connection with previous studies

Linear Algebra has strong links in deep learning, for example matrix multiplication is used in neural networks, this understanding will most likely help me understand

convolutional neural networks and more. Calculus provides concepts such as inputs and functions. My artificial intelligence module contains vast links, like neural networks.

Currently I'm studying machine learning and image processing, machine learning has a wider range of techniques to be applied, although it may not be using the same programming libraries as this project, it should be beneficial. In image processing, we handle images using python.

However, none of these modules use machine learning libraries, so this is a good opportunity to expand my knowledge of these modules studied through application.

11.2.4 Literature survey

A range of literature should be researched to understand deep learning. The mathematics, the theory and the application are all different aspects which would be beneficial to research. A general view should be researched and then a more specific one.

For the mathematics aspect, a general view is suited. Linear Algebra is key in neural networks, therefore I have chosen David Poole's: "Linear algebra: a modern introduction" [1] to understand the matrices in deep learning. Calculus [2] by Briggs, Cochran and Gillett will be useful for understanding the functions in AI.

A book with a wide range of deep learning explanations is the MIT Deep Learning book [3].

Hands-on Machine Learning with Scikit-Learn, Keras TensorFlow [4] is one of the greatest books for learning machine learning, it explains a lot of theory behind the techniques, and they demonstrate this using different python libraries. This has a general scope with some specific examples.

For the application, specific examples would be most useful, the TensorFlow website [5] has a vast number of tutorials you can learn from. The matplotlib website [6] may be useful for displaying the accuracy of our results. Along the way I require assistance with learning Python, Python Essentials [7] is a book I can use for the programming.

Another specific application would be a conference on image classification using convolutional neural networks. I've chosen an article on "Medical Image Classification with Convolutional Neural Network" [8], it provides a great explanation of the theory along with its application with real world issues.

Further research into variations of autoencoders using the Journel of Machine Learning Research under "Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion" [9], provides an understanding on how autoencoders are used to help denoise images.

A conference paper on generative adversarial networks [10] will be used to introduce the topic to myself.

11.2.5 References

1. D.Poole, Linear algebra: a modern introduction, 4th ed., Stamford CT: Cengage Learning, 2015
2. W.Briggs, L.Cochran, B.Gillett, Calculus, 2nd ed., Boston MA: Pearson, 2015
3. I.Goodfellow, Y.Bengio and A.Courville, Deep Learning, 1st ed., Cambridge, MA: MIT Press, 2016.
4. A.Géron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd ed., Sebastopol CA: O'Reilly Media, 2019
5. "Tensorflow Tutorials," Tutorials — Tensorflow Core, Aug. 18, 2020. [Online] Available: tensorflow.org/tutorials
6. "Tutorials", Matplotlib, Sep 15. 2020.[Online] Available: https://matplotlib.org/3.3.2/tutorials/index.html
7. S.Lott, Python Essentials, Birmingham, UK : Packt Publishing, 2015
8. Q. Li, W. Cai, X. Wang, Y. Zhou, D. D. Feng and M. Chen, "Medical image classification with convolutional neural network," in 13th International Conference on Control, Automation, Robotics Vision Marina Bay Sands, Singapore, 2014. [Online] Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7064414casa_token=qJI7EVoFFnIAAAAA:QKiGaY8Xz2YorWO0thAuy6qlnKny-v1GtfUYm-eijKt-uQH_WAJRTZ0gL7uti6Qbz9bogtqDSeEtag=1
9. P.Vincent, H.Larochelle, I.Lajoie, Y.Bengio and P.Manzagol, "Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion": Journal of Machine Learning Research, vol.11, no.12, pp.3371-3408, Oct. 2010. [Online] Available: https://www.jmlr.org/papers/volume11/vincent

10a/vincent10a.pdf?source=post_page————————————
10. A.Radford, L.Metz and S.Chintala, “Unsupervised representation learning with deep convolutional generative adversarial networks” in International Conference on Learning Representations (ICLR), San Juan, United States, 2016. pp. 1-16 [Online] Available: https://arxiv.org/pdf/1511.06434.pdf

11.2.6 Equipment, facilities and cost

I will use my own laptop for running models, TensorFlow and Python will be used, they’re free and widely used. Spyder will be my working environment, I will have a backup on Jupiter notebook incase my laptop breaks, as the code can be ran using servers instead of locally if my backup device isn’t powerful enough.

11.2.7 Action plan

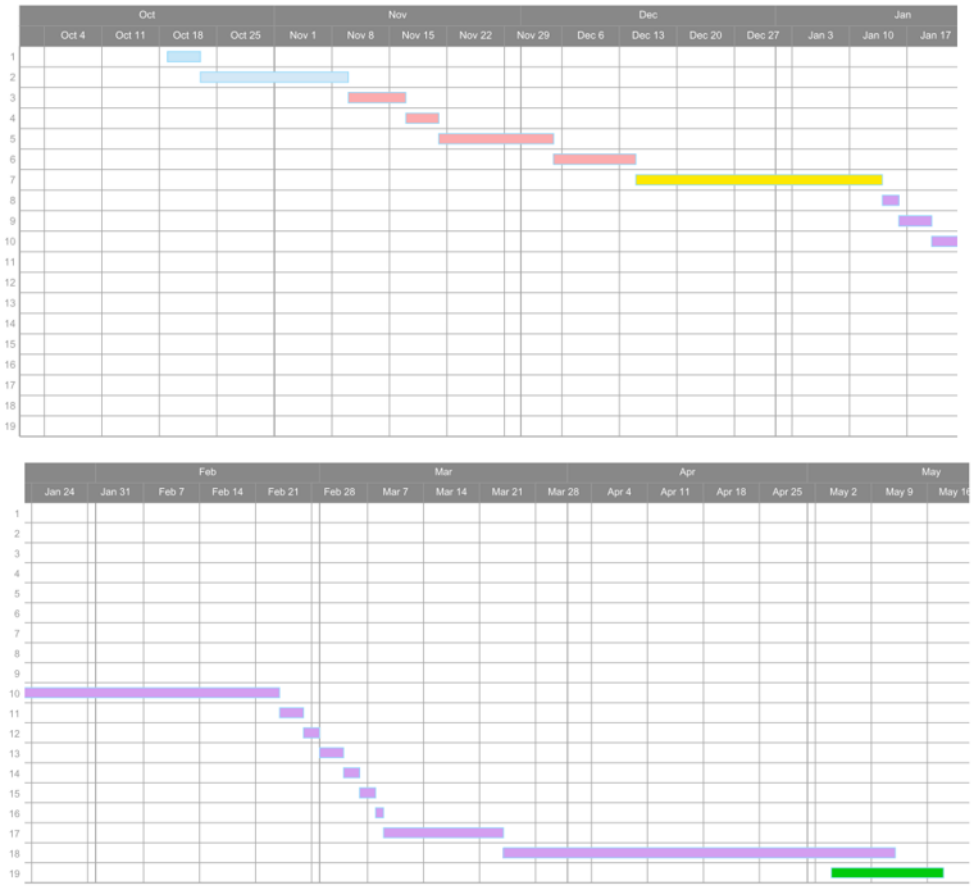
Here is an schedule I should aim to approximately follow, I shall do my references and logbook throughout.

No	Task	Duration	Start	Finish
1	Understand project	4d	19/10/20	22/10/20
2	Research Plan	18d	23/10/20	09/11/20
3	Linear Algebra Research	7d	10/11/20	16/11/20
4	Calculus Research	4d	17/11/20	20/11/20
5	Deep learning theory research	14d	21/11/20	04/12/20
6	Deep learning practical research	10d	05/12/20	14/12/20
7	Programming	30d	15/12/20	13/01/21
8	Report - Methods	2d	14/01/21	15/01/21
9	Report - Results	4d	16/01/21	19/01/21
10	Report – Mathematics and Deep learning research	35d	20/01/21	23/02/21
11	Report - Introduction	3d	24/02/21	26/02/21
12	Report - Abstract	2d	27/02/21	28/02/21
13	Report - Conclusion	3d	01/03/21	03/03/21
14	Report – Table of contents	2d	04/03/21	05/03/21
15	Report - Appendix	2d	06/03/21	07/03/21
16	Report – Title Page	1d	08/03/21	08/03/21
17	Report – Improve Draft	15d	09/03/21	23/03/21
18	Report – Improve Scientific Report	49d	24/03/21	11/05/21
19	Viva Preparation	14d	04/05/21	17/05/21

Hand in dates:

11.2.8 Risk assessment

As I’m using a laptop, I must make sure water is kept away from it. I will be down at a desk so the chair and desk should be stable, and I should be sitting with good



Work	Date
Research Plan	09/11/2020
Logbook submission 1	25/11/2020
Logbook submission 2	16/12/2020
Logbook submission 3	17/02/2021
Draft	23/03/2021
Scientific Report	11/05/2021
Viva	17-28/05/2021

posture.

11.2.9 Discussion arrangements

Matt and I will call via Microsoft Teams on Thursdays at 14:30, however there will be times when this is to be rearranged due to Matt or I being occupied, this will be discussed and rearranged in the meeting prior.

Bibliography

[1] *S.Gajawada, "The Math behind Artificial Neural Networks," Towardsdatascience Blog, Nov.17, 2019. [Online] Available: https://towardsdatascience.com/the-heart-of-artificial-neural-networks-26627e8c03ba*

[2] *I.Goodfellow, Y.Bengio and A.Courville, Deep Learning, 1st ed., Cambridge, MA: MIT Press 2016.*

[3] *Q.Li, W.Cai, X.Wang,Y.Zho, D.D.Feng and M.Chen, "Medical Image Classification with Convolutional Neural Network," in 13th International Conference on Control, Automation, Robotics Vision Marina Bay Sands, Singapore, 2014. pp. 844-848 [Online] Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7064414casa_token=CDk0xo1MXbQAAAAA:C5y28-24mKceBJ1Tcydshr75k0v0AeudzWnsWFC1hQvwXMQ_MFTarz-OeaUwUtNG1YAEtXTYI4Y*

[4] *A.Géron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd ed., Sebastopol CA: O'Reilly Media, 2019*

[5] *"Basic classification: Classify images of clothing ," TensorFlow, Mar. 19, 2021. [Online] Available: https://www.tensorflow.org/tutorials/keras/classification*

[6] *"Convolutional Neural Network (CNN) ," TensorFlow, Mar. 19, 2021. [Online] Available: https://www.tensorflow.org/tutorials/images/cnn*

[7] *"Intro to Autoencoders ," TensorFlow, May. 01, 2021. [Online] Available: https://www.tensorflow.org/tutorials/generative/autoencoder*

[8] *"Deep Convolutional Generative Adversarial Network ," TensorFlow, Mar. 19, 2021. [Online] Available: https://www.tensorflow.org/tutorials/generative/dcgan*

[9] *Desmos Graphing Calculator, Version 6.4.5.0 for Android, New York , NY: Desmos Inc, 2014. [App] Available from: https://play.google.com/store/apps/details?id=com.desmos.calculatorhl=en*

[10] *T. Wiesel and D. Hubel, "Receptive fields of single neurones in the cat's striate cortex.," The Journal of physiology., vol. 148, no. 3, pp. 574-591, Feb. 2012. [Online] Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1363130/*

[11] *K. Fukushima, "Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position," Biological Cybernetics, vol. 36, no. 4, pp. 193–202, Apr. 1980. [Online] Available: https://www.cs.princeton.edu/courses/archive/spr08/cos598B/Readings/Fukushima1980.pdf*

[12] *Y. LeCun et all, " Backpropagation Applied to Handwritten Zip Code Recognition", Neural Computation, vol. 1, no. 4, pp. 541-551 Dec. 1989*

[13] *I. Goodfellow et all, "Generative Adversial Nets," in International Conference on Neural Information Processing Systems, Montreal, Canada, 2014, pp. 1-9. [Online] Available: https://papers.nips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf*

[14] *D. Rumelhart, G. Hinton and R. Williams, "Learning Internal Representations by Error Propagation", Parallel distributed processing: explorations in the microstructure of cognition., vol. 1, pp. 318–362, Jan. 1986*

[15] R. Forte, Mastering Predictive Analytics with R, 1st. ed, Birmingham, UK: Packt Publishing Ltd, 2015

[16] A. Radford, L.Metz and S.Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks" in International Conference on Learning Representations (ICLR), San Juan, United States, 2016. pp. 1-16 [Online] Available: https://arxiv.org/pdf/1511.06434.pdf

[17] R. Draelos, "The History of Convolutional Neural Networks," Glass Box,[Online], Apr. 13, 2019. Available: https://glassboxmedicine.com/2019/04/13/a-short-history-of-convolutional-neural-networks/

[18] P. Solia, "How does Backpropagation work in a CNN," Medium, Mar. 19, 2019. [Online] Available: https://medium.com/@pavisj/convolutions-and-backpropagations-46026a8f5d2c

[19] P.Vincent, H.Larochelle, I.Lajoie, Y.Bengio and P.Manzagol, "Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion": Journal of Machine Learning Research, vol.11, no.12, pp.3371-3408, Oct. 2010. [Online] Available:

https://www.jmlr.org/papers/volume11/vincent
10a/vincent10a.pdf?source=post_page————————————

[20] D. Radečić. ”Top 3 Classification Machine Learning Metrics,” Dec. 17 2020. [Online] Available: https://www.r-bloggers.com/2020/12/top-3-classification-machine-learning-metrics-ditch-accuracy-once-and-for-all/

[21] M.Bernacki, P.Wlodarczyk, A.Golda, ”Principles of training multi-layer neural network usng backpropagation” Department of Electroncics AGH, [Online], Sep.06,2004. Available: http://home.agh.edu.pl/ vlsi/AI/backp_t_en/backprop.html

[22] R. Gomez, ”Understanding Categorical Cross-Entropy Loss, Binary Cross-Entropy Loss, Softmax Loss, Logistic Loss, Focal Loss and all those confusing names” May.23, 2018 [Online] Available: https://gombru.github.io/2018/05/23/cross_entropy_loss/