UNIVERSITY OF
LINCOLN

SCHOOL OF MATHEMATICS
AND PHYSICS

Mathematics of Image Processing and Recognition

Adelaide Baron

BAR16630927

05/2022

Supervised by Dr Matthew Watkins

Scientific Report in the Mathematics Module Project

MTH3001M

Abstract

This project aims to provide the reader with an understanding of the mathematics behind image processing and recognition, focusing on the particular case of handwritten digits zero to nine. We begin with an in-depth study of neural networks, their history, architecture, and the learning process of categorising images. We briefly visit the topic of coding such networks and the variety of available networks, and how they may impact the process. Afterwards, we visit edge detection filters and how these may be used in image processing to enhance images. Finally, we discuss interpolation methods such as Chebyshev Polynomials.

Contents

1 Introduction

Photographs have been used for almost two hundred years to record memories. Initially taking around 20 minutes for a picture, considering time for exposure, photography was a luxury reserved for life events. Today, in the age of computers, cameras are easily accessible and developed, and so we use images to record information. From number plate recognition to CCTV, data is extracted from the images captured, where it is then stored and manipulated for development. The human eye can identify and classify information from the images - but how do we train computers to do so? This project aims to understand how we may use mathematics to process and classify images. We will strip back the recognition and classification process to the core of handwritten digit recognition. Once a method is developed to classify digits correctly, we may extend it to characters and symbols. It is then not a far stretch to develop these processes to recognise digits, characters, and symbols within images, with uses such as number plate recognition.

Throughout my undergraduate studies, I have used an electronic tablet and stylus to write my notes as I can search a word in my files to bring back all matches. When writing my early notes for this project, I wondered how the tablet recognised what I had written? Upon research, I found that neural networks are a popular method to classify digits, with success rates of over 97% [1]. We begin this project with the history of neural networks (NN) and the earliest forms to understand the process. Once this is established, we move on to the architecture of artificial neural networks (ANNs), developing an understanding of each component and its impact on the classification process. The project then shows the reader an example of digit recognition using ANNs.

Once the reader understands an ANN and its structure, we explore activation functions. The ANN functions are used to adjust the data passed through to the output. In the next section, we explore the learning process of ANNs and how they take previously classified data to adjust their network to correctly classify unseen images. We culminate our focus on ANNs with a section on coding the ANNs, using languages such as C++ and Python to automate the process using the mathematical basis developed. Before moving away from neural networks, we explore the different types of NN. We focus primarily on ANNs, but we acknowledge Recurrent neural networks and convolutional neural networks to understand the differences and how we may take this project further with future studies. This section also discusses ways to generate digits with the NNs, primarily autoencoders and generative adversarial networks. The topic of neural networks deviates from mathematics to the topic of computer science. So we

compare NNs with processes such as Edge Detection Filters (EDF) and interpolation.

Starting with EDFs, within this section, we acknowledge the Sobel Filter, which gave the initial motivation for me to take on this project [2], as discussed in my research plan. In the next section, we explore interpolation, starting with Lagrange interpolation. Once the reader has a strong foundation of interpolation, we move on to the method of Chebyshev polynomials and a brief overview of their use in image classification.

We finish the report with a section on creating datasets before our conclusion discussing future opportunities from this project and assessing the contents explored.

1.1 Literature Review

The main book I used to develop my understanding of neural networks and handwriting recognition was "**Neural Networks and Deep Learning**" by Michael Nielsen [1]. This book explores the development of NNs, the mathematics of the learning process, and derivation of. This book is referenced greatly in the online community for creating programmes for handwriting recognition. As a result, however, this book does too focus heavily on the programming of such NNs.

As the overall topic of neural networks centres around programming languages, it was difficult to find physical text to describe the processes, rather than links to online resources.

Another main book that I used for my understanding of neural networks and their structure was "**Machine Learning: An Algorithmic Perspective**" by Stephen Marsland [3]. Focussing more on the structure NNs, and the mathematics and statistics supporting it. Marsland's work was supported in other texts, such as "How many hidden layers and nodes?" by D. Stathakis [4]. Additionally, Marsland touches upon the topic of interpolation and basis functions, once again giving a good overview of the topic as a whole.

The main paper inspiring our focus on Chebyshev polynomials and interpolation overall was "**Representing and Characterizing Handwritten Mathematical Symbols through Succient Functional Approximation**" by Burce Char and Stephen Watt [5]. This paper also explores the development of Chebyshev polynomials for character recognition, modelling handwriting as strokes and capturing the shape of handwriting. It also references the InkML programme, which allowed me to explore documentation produced by W3C to explain the syntax and process [6].

1.2 Data Sets

Throughout this project we will use data from the MNIST database [7]. This is a set of handwritten digits by National Institute of Standards and Technology including digits from employees of United States Census Bureau, and students in highschool. We split datasets into training and test data. The training set is used to improve the network, and the test data is used to test if the ANN learned as expected from the training data. We will discuss later, in section 10, how we may go about collecting our own data set.

2 ANN: An Introduction

Our brains work using biological networks between neurons. Neurons fire, triggering others to fire, akin to a chain reaction. These chain reactions are how the brain works, from controlling our senses to moving our muscles. Artificial neural networks (ANN) are modelled loosely on this biological process.

Like the brain, ANNs consist of neurons and connections. We will explore the structure of an ANN in section 3. For now, consider figure 1; showing the neurons (colourful circles), and the connections (lines).

In this report, when I refer to neural networks unless specified otherwise, I mean artificial neural networks. Additionally, some texts refer to neurons as nodes, but we will retain the term neuron.

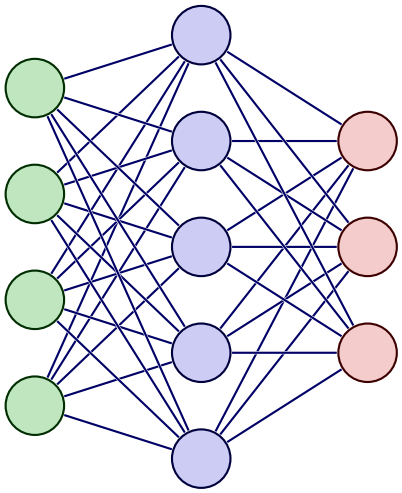


Figure 1: Artificial Neural Network

2.1 History

Biological neural networks were first observed and discussed in the late 1800s by Alexander Bain [8] and William James [9]. Until the 1940s, neural networks remained a purely biological concept. However, work by McCulloch and Pitts [10] on neural network algorithms translated the theory into mathematics; leading to networks such as perceptrons (section 2.1.1) and the theory of ANN as we know it today.

2.1.1 Perceptrons

Perceptrons, depicted in figure 2, were the first mathematical neural networks to be proposed; named and studied by Rosebnblatt in 1962 [11]. The perceptron is used of-

ten as a binary classifier. However, we will see in section 4.1 that this is not the most practical for image recognition. To understand why, let's consider an example [11] [12].

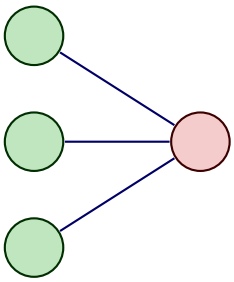


Figure 2: Perceptron

Example 1 *Consider a network with two boolean inputs and outputs, 0 and 1. The output depends on the inputs, with the following rule:*

Neuron A	*Neuron B*	*Output*
1	*1*	*1*
1	*0*	*1*
0	*1*	*1*
0	*0*	*0*

This may be modelled using a NN, similar to figure 2.

Later in this report, section 6, we will explore coding such a network. You can see the results in the appendices,13.4.1.

3 ANN: Structure

Neural networks consist of two main components; layers, and weights. Within layers there are neurons, each with an associated bias. Information is passed forward from one layer to the next, and the weights affect this information transfer. As mentioned earlier, a generic NN looks like figure 1.

3.1 Layers

Layers are the columns of neurons in an ANN, and each one has a particular purpose. The first layer (left) is called the input layer, the last (right) is called the output layer, and those in between are hidden layers, as shown in figure 3. The values of neurons in a layer, (L-1), determines the values of neurons in the next layer, L.

3.1.1 Number of Layers in a Network

When designing an ANN, we need to consider the optimal amount of hidden layers. The more layers, the longer the potential processing time may be. However, more layers may also allow for more sophisticated learning. This varies depending upon needs, and can come down to trial and error. However, a general rule for the amount of layers [13]:

i. 0 Hidden layers - used for simple decisions, such as perceptrons.

ii. 1 Hidden layer - used for simple mappings $X \mapsto Y$, both X and Y finite. Can be used within handwriting recognition, but restricts learning and may take longer to train.

iii. 2 Hidden layers - used to represent functions of any shape.

You may see ANNs with more than two, but this is not practical within this project.

3.1.2 Referencing layers

We adopt the following approach to referencing layers, indexing from the output layer backwards:

- Output = L
- Hidden layers = L - (number of layers between this layer and the output, L) - 1
- Input = L - (amount of hidden layers) - 1,

When labelling components of an ANN, such as a weight or a neuron, we use a superscript to denote the layer, for example: w^L, n^L.

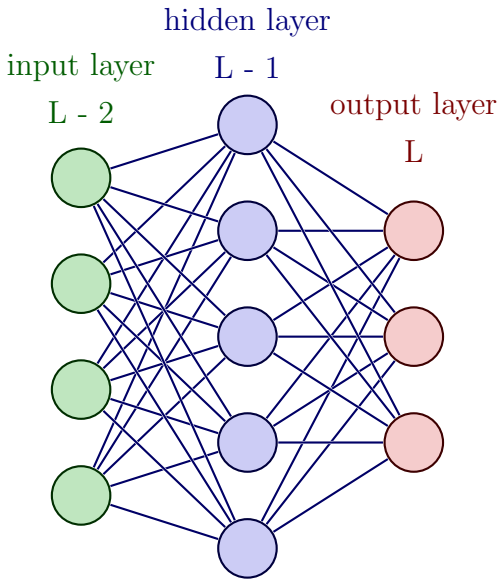


Figure 3: ANN layers

3.2 Neurons

Neurons sit within layers and have individual values. In the input layer, there is generally one neuron for each input value. In diagrams, such as figure 3, the circles are the neurons.

3.2.1 Number of Neurons in a Network

Like layers, the amount of neurons in a network is down to the creator, and depends on the requirements of the network. For the output, each neuron may represent a certain outcome, and thus we would have n neurons for n outputs.

For the case of handwriting recognition, we generally consider a 28 x 28 pixel image. So, with 784 inputs, we have 784 input neurons. We will consider 10 outputs, classifying the digits $[0, 9]$.

The amount of neurons within hidden layers, we denote n_{hidden} is more varied. Some people may make a decision based on aesthetics or at random [1], but there are some popular rules to follow [4] [13]:

i. n_{hidden} between n_{input} and n_{output}

ii. $n_{hidden} = \frac{2}{3} \, (\, n_{input} + n_{output})$

iii. $n_{neuron} < 2|n_{input}$

3.2.2 Referencing neurons

When referring to parts of an ANN, we use a subscript to denote the neuron index. We index from the top, n = 1, and work downwards. The indexing resets for each layer.

Example 2 *The 5th neuron in layer L-2 is* a_5^{L-2}
The 5th neuron in layer L-1 is a_5^{L-1}

3.2.3 Activation

The numerical value of a neuron in an ANN is called the activation. In some cases, the value determines whether or not the neuron activates (for example, section 4).

We typically refer to an activation as a, and reference them the same way as neurons. We may also refer to the activations of the neurons in the output layer as t^L, and the desired output as y^L.

Within this project, the activation of neuron n in layer L, a_n^L depends upon the the product of activations and weights (3.3), added to the bias (3.4), in layer $L-1$. I.e.: $a_1^{L-1}w_{n1} + a_2^{L-1}w_{n2} + ... + a_m^{L-1}w_{nm} + b_n^L$, which we denote z_n^L. I.e.:

$$z_n^L = a_1^{L-1}w_{n1} + a_2^{L-1}w_{n2} + ... + a_m^{L-1}w_{nm} + b_n^L \tag{1}$$

where m = amount of neurons in layer (L-1), w and b are the weights and biases, discussed in sections 3.3 and 3.4.

To aid the network in feeding forward values, it is typical to use an activation function, a function of z_n^L. We will explore this more in section 4.3.1, but in this report we restrict activation values to the range $[0, 1]$ using the sigmoid function, defined as follows:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

where $x \in \mathbb{R}, \sigma \in [0, 1]$. I.e. $a_n^L = \sigma(z_n^L)$.

In a single layer, the neuron with the highest activation is referred to as the **brightest**. The brightest neuron in the output layer is the 'choice' of the network.

3.3 Weights

The connections between neurons in different layers are given weights. Increasing the weight between connections gives that specific connection more impact.

Example 3 *Consider a section of an ANN where we have two neurons, a_n^{L-1} and a_m^{L-1}, connected to a_j^L, with respective weights w_{jn}^L and w_{jm}^L. For simplicity, assume that these are the only three neurons, and we won't use an activation function. So, $a_j^L = w_{jn}^L a_n^{L-1} + w_{jm}^L a_m^{L-1}$. Increasing the value of w_{jn}^L will increase the impact of a_{n-1}^L on the output, a_j^L.*

Thus, the higher a weight, the more influence that particular connection has on the end result. When we introduce the activation function, the relationship is no longer linear, but it remains positive.

3.3.1 Number of Weights in a Network

A neuron in layer $L-1$ has $n+m$ associated weights, where $n = |L-2|$, $m = |L|$.

3.3.2 Referencing Weights

We adopt a similar process as we do for neurons, but with two indices rather than one. A weight connecting the kth neuron in $L-1$ to to jth neuron in L is w_{jk}^L

3.4 Bias

A bias adds more impact to a particular neuron. It is added on to the value of the weights combined with activations, and each neuron has only one bias. Revisiting equation 1: $z_n^L = a_1^{L-1} w_{n1} + a_2^{L-1} w_{n2} + ... + a_m^{L-1} w_{nm} + \mathbf{b}_n^L$.

3.4.1 Number of Biases in a Network

There is one bias per neuron. E.g. for a four-layer network with ten neurons in each, there are forty biases.

3.4.2 Referencing Biases

Biases are referenced the same as neurons, but b rather than n. I.e. b_j^L

3.5 Example

I am going to demonstrate the structure of an ANN with a handwriting recognition example, referring to figure 4. We will refer to this network throughout the report.

Example 4 *We consider pictures that are 28 x 28 pixels so that our ANN is compatible with the MNIST database [7].*

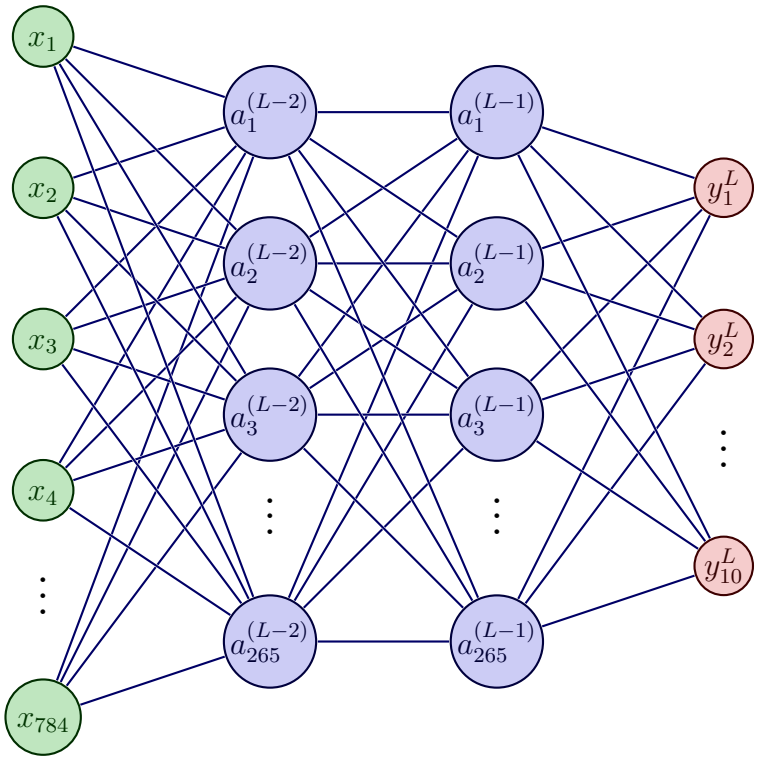


Figure 4: ANN for handwriting recognition of 28 x 28 pixel image

Our input values are the grayscale values of each individual pixel, so there are 784 inputs: $x_1, ...x_{784}$. *Our network will classify an image as any digit* $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$, *giving 10 outputs. Each output neuron,* y^L_{n-1} *indicates a digit n.*

Considering section 3.1.1 and 3.2.1, I have chosen two hidden layers, each with 265 neurons ($n_{hidden} = \frac{2}{3}$ *(* $n_{input} + n_{output}$*) , divided by two for the two hidden layers).*

Random weights ($w^{L-2}_{1,1}, ...w^L_{10,265}$*) and biases (*$b^{L-2}_1, ..., b^L_{10}$*) are assigned at first.*

We begin by feeding inputs, from the MNIST [7] database, into the network. The 'brightest' neuron in the output layer is the selected classification from the network. At first, it is unlikely that the ANN will correctly classify the digit. For example, an image that clearly looks like the digit 5 to the human eye may give the output values

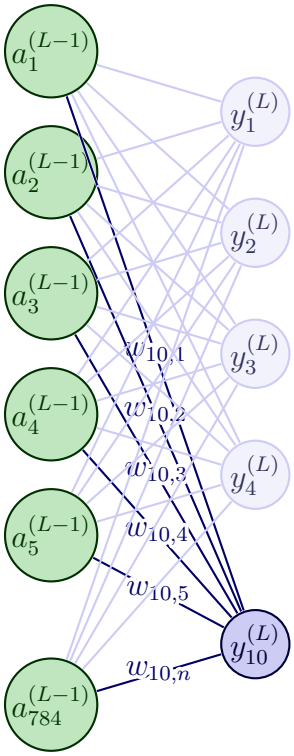


Figure 5: Random output of the example

in table 1, depicted in figure 5.

i	t_i^L
1	*0.03*
2	*0.02*
3	*0.08*
4	*0.34*
5	*0.56*
6	*0.58*
7	*0.12*
8	*0.39*
9	*0.60*
10	*0.97*

Table 1:
Random Output of Example 4

i	t_i^L
1	*0.22*
2	*0.56*
3	*0.38*
4	*0.14*
5	*0.09*
6	*0.10*
7	*0.12*
8	*0.92*
9	*0.60*
10	*0.24*

Table 2:
Potential Output of Example 4 after training

Using methods to be discussed in Chapter 5, we may adjust the weights and biases using different training data so that the ANN can classify with a higher accuracy. Eventually, the ANN may take a clear digit 7, and output results similar to table 1. As $i = 8$ is the brightest neuron, the network has selected 7 as the output.

4 Activation Functions

As mentioned in section 3.2.3, we use an activation function on z to pass values forward through a network, rather than directly passing the value of z through. It is natural to question why can't we directly feed through the value of z? Well, without an activation function, an ANN would behave like a linear regression model, performing linear transformations on the input, weights, and biases; there would be little need for multiple layers. This would reduce the accuracy in some cases, as the ANN could not be well trained for differing inputs.

Throughout this chapter, I will refer to z as the input for the activation functions, as that is the case for this project. I advise the reader to be mindful that the calculation of z initially may vary depending on the network.

As mentioned, in this project we use the sigmoid function, which is useful for classification problems. When adapting to different scenarios, however, different activation functions may be more appropriate. We shall explore some of the common types, including a deeper dive into the sigmoid function, next.

4.1 Binary Step Function

The Binary Step Function (BSF) works using a threshold [14]. If the input, z, meets a certain threshold, the neuron is activated. It is defined as follows:

$$f(x) = \begin{cases} 1 & x < 0 \\ 0 & x \geq 0 \end{cases} \tag{3}$$

Due to the limited outputs (0 or 1), this is not useful for image classification. Suppose we had a network like example 4. If we used the BSF, we may have multiple neurons in the output layer as 1. There would be no way for us to determine which neuron was the network's choice, as they would all have the same value of 1.

We will see later (section 5.2.2) that activation functions with a gradient of zero, like the BST, are not useful for the learning process.

4.2 Linear Activation Functions

Linear Activation Functions (LAF) may be given by [14]:

$$f(x) = x \tag{4}$$

Similarly to the BSF, LAFs are not useful for the learning process due to their gradient. The gradient is constant, does not vary with input, so cannot be used to adjust the weights and biases that we will see in section 5.2.2.

Using the LAF causes all preceding layers to become linear functions of the one before, making the use of layers somewhat redundant. As the use of layers is recommended in image recognition [13], we do not use the LAF.

4.3 Non-Linear Activation Functions

Unlike BSFs and LAFs, non-linear activation functions (NLAF) allow for system learning due to their non-constant gradient (section 5.2.2).

There are many different NLAFs. I, however, shall highlight three of the most common NLAFs relevant to this project [15].

4.3.1 Sigmoid Function

We begin with the sigmoid function, equation 2, which we use as follows :

$$a^L = \sigma(z^L) \tag{5}$$

Typically, the sigmoid function is used for probabilistic scenarios due to the output range $[0, 1]$. In example 4, we may think of the activation in the final layer as the probability that the number was the one that was fed into the network.

Figure 6 shows $\sigma(x)$ and the gradient $\sigma'(x)$. You can see that the gradient is only significant for roughly $x \in [-3.5, 3.5]$. For values outside of this range, the gradient tends to 0, and the learning process may not be successful. This issue is referred to as the vanishing gradient problem.

4.3.2 Hyperbolic Tangent Function

Another common activation function, $\tanh(x)$, gives a similar output to the sigmoid, and has a similarly behaving gradient. As a result, it too suffers from the vanishing gradient problem.

$\tanh(x)$, figure 7, has outputs in the range $(-1, 1)$, and is defined as follows:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{6}$$

The hyperbolic tangent is more commonly used to indicate correlation. We want to see which neuron the network has classified, and thus only need positive values like the sigmoid function provides. Having positive and negative values may increase the difficulty of comparison for the brightest neuron.

4.3.3 Rectified Linear Unit

The Rectified Linear Unit (ReLU) is a non-linear activation function that behaves linearly [16], defined as follows:

$$f(x) = \max(0, x) \tag{7}$$

The ReLU function only activates neurons that end up with a positive activation value. If the input is less than 0, the function returns 0. It is commonly used for hidden layers to avoid the vanishing gradient problem. The derviative of the ReLU function is 0 for negative numbers, but 1 for positive inputs [16]. When using a large batch of training data, the average derivative is usually not zero, allowing for gradient descent.

Throughout this project we will use the sigmoid function as the restriction of outputs between [0,1] assists with visualising the learning process. However, it must be noted that due to the vanishing gradient problem, this is not always ideal.

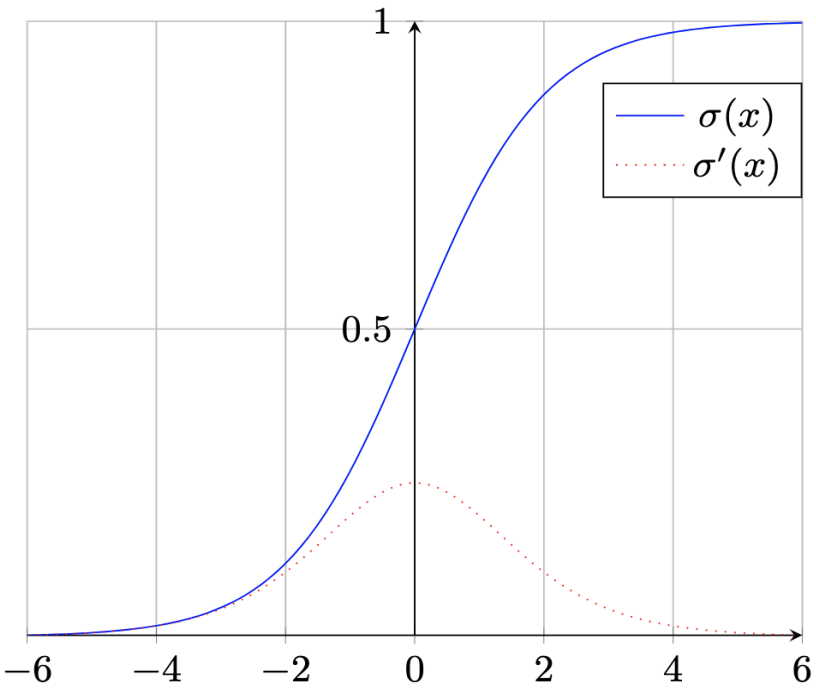


Figure 6: Sigmoid Function and it's gradient

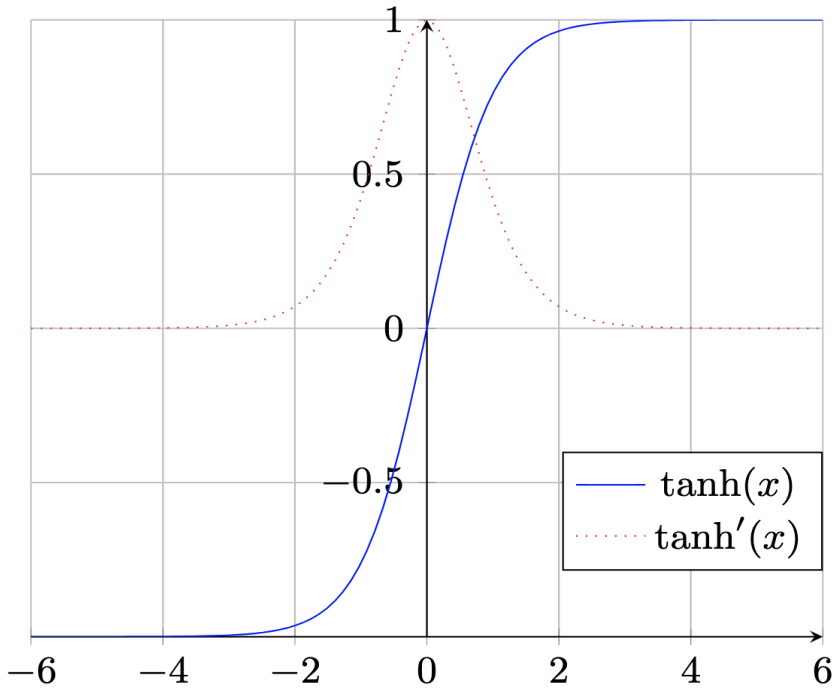


Figure 7: Hyperbolic Tangent Function and it's gradient

5 Learning in an ANN

The first step of the learning process is to assess how incorrect, or correct, the network is by assessing the outputs, $a_n^L = \sigma(z^L)$. We do this using the cost function.

5.1 Cost Function

To determine the cost of the network, first we look at the loss of the network. We define the loss as the square of the difference between the actual output, t, and the expected output, y.

$$(y - t)^2 \tag{8}$$

We then form the mean-squared error cost function:

$$C = \frac{1}{n}\sum_{i=1}^{n}(y - t)_i^2 \tag{9}$$

Where i = index of neuron, y denotes the desired output, and t denotes the actual output (i.e. $t_i = a_i^L$).

The cost of a network, C, is a k-dimensional column vector, where k is the number of output neurons.

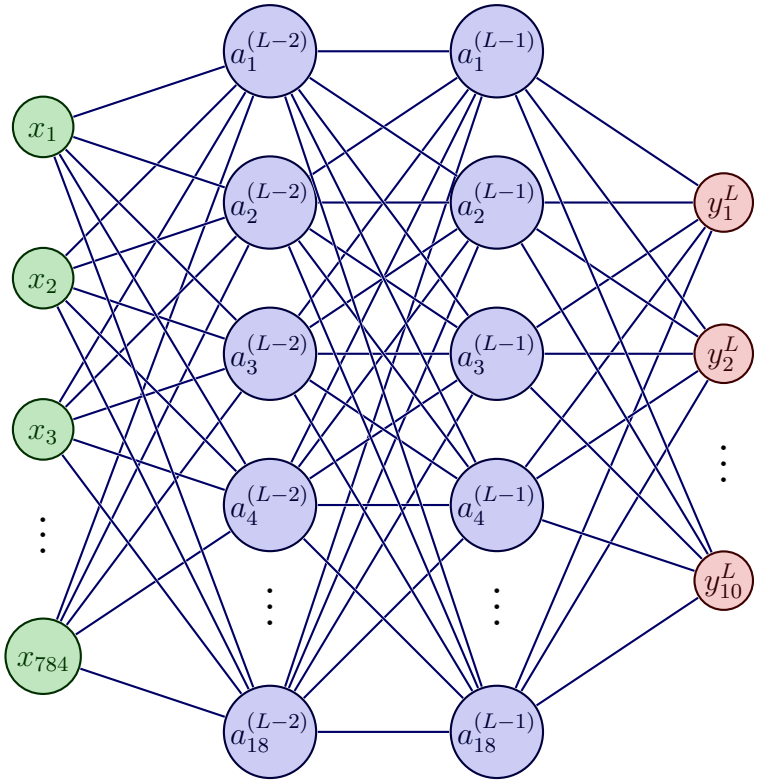


Figure 8: ANN for handwriting recognition of 28 x 28 pixel image

Example 5 *Building on example 4, table 1, we assess the cost function using* t_i^L *and* y_i^L *(table 3):*

i	t_i^L	y_i^L
1	*0.03*	*0.00*
2	*0.02*	*0.00*
3	*0.08*	*0.00*
4	*0.34*	*0.00*
5	*0.56*	*0.00*
6	*0.58*	*1.00*
7	*0.12*	*0.00*
8	*0.39*	*0.00*
9	*0.60*	*0.00*
10	*0.97*	*0.00*

Table 3:
Table 1 with the desired outputs, y

Using equation 9:

$$C = (0.03 - 0.00)^2 + (0.02 - 0.00)^2 + (0.08 - 0.00)^2 + (0.34 - 0.00)^2 + (0.56 - 0.00)^2 + (0.58 - 1.00)^2 + (0.12 - 0.00)^2 + (0.39 - 0.00)^2 + (0.60 - 1.00)^2 + (0.97 - 1.00)^2$$

$$C = 2.0807$$

5.1.1 What does the cost function tell us?

The smaller the cost function, the better the performance of the ANN. A small value indicates that the difference between the actual and desired output is small, and thus the network is correctly classifying. The exact value depends upon the network and goals. The last example, example 5, depicts an unsuccessful network.

5.1.2 Using a Cost Function to develop an ANN

As we want our C to have the smallest possible value, we aim to minimise it. To do this, first we first allocate random weights and biases; this will likely cause random and very incorrect outputs (as in example 4). Then, we run the network with a random piece of data from the training set, and evaluate the cost. We then use gradient descent, discussed in section 5.2, to determine how to minimise the cost, make those changes, and start again with a new piece of training data. This is called the learning phase.

5.2 Gradient Descent

In section 3 we introduced the structure of an ANN. Considering this along with the equation for C, equation 9, we see that to minimise the cost, we need any of the following to change [3]:

1. The inputs $x_1, x_2, ..., x_n$
2. The activations
3. The weights and biases

We cannot change the inputs, nor can we directly change the activations (as they are a function of the input and weights). We can, however, change the weights and biases, which then adjusts the activations; this process is gradient descent.

The positive gradient of the cost function, $\bigtriangledown C$, tells us which direction to move in to increase the function the quickest. Therefore, if we take the negative gradient, we can determine how to minimise the function.

$$\bigtriangledown C = \begin{bmatrix} \frac{\partial C}{\partial w^1} \\ \frac{\partial C}{\partial b^1} \\ ... \\ \frac{\partial C}{\partial w^L} \\ \frac{\partial C}{\partial b^L} \end{bmatrix} \tag{10}$$

Then we may adjust the weights as follows: $w'^{L}_{ji} = w^{L}_{ji} - \frac{\partial C_k}{\partial w^{L}_{ji}}$, and similarly for the biases. We will explore how to calculate these in section 5.2.2.

An issue with gradient descent is that we may locate only a local minimum, rather than a global. To overcome this, we adapt our method of gradient descent. The process we use in this project is Stochastic Gradient Descent (SGD), updating the weights and biases with each iteraction of the ANN. SGD suffers from noise issues, but helps minimise the local minima issue [17].

Example 6 *Take this random list of all weights and biases:*

$$\overrightarrow{w} = \begin{bmatrix} 2.25 \\ -1.57 \\ 1.98 \\ ... \\ -1.16 \\ 3.82 \\ 1.21 \end{bmatrix}$$

Using methods to be discussed, we may determine the negative gradient, $-\bigtriangledown C$, to see how to adjust each of the above to minimise the cost. Propose we have the following result for $-\bigtriangledown C$:

$$-\bigtriangledown C(\overrightarrow{w}) = \begin{bmatrix} -0.18 \\ 0.45 \\ -.051 \\ ... \\ 0.40 \\ -0.32 \\ 0.82 \end{bmatrix}$$

This tells us that we may make the following changes to the ANN:
$w_1 + 0.18, w_2 - 0.45, w_3 + 0.051,$ *...,* $w_{n-2} - 0.40, w_{n-1} + 0.32, w_n - 0.82$

Making the direct changes $w'^{L}_{ji} = w^{L}_{ji} - \frac{\partial C_k}{\partial w^{L}_{ji}}$ can alter the network too drastically, damaging the learning process. To overcome this issue, we introduce the learning rate, η.

5.2.1 Learning Rate

The learning rate is a positive parameter η used to stabilise the ANN throughout the learning process by determining the rate at which the ANN learns [3]. Without it, the network would change significantly with each incorrectly classified image. This could cause instability and fluctuation and lead to the cost function not reducing. It is used as follows:

$$\Delta w = -\eta \bigtriangledown C \tag{11}$$

Meaning each weight is adjusted according to:

$$w'^{L}_{ji} = w^{L}_{ji} - \eta \frac{\partial C_k}{\partial w^{L}_{ji}} \tag{12}$$

Values of η

The value of η varies depending on the network. Typical values of η are within the range $[0.1, 0.4]$, [3]. Setting $\eta = 1$ is the equivalent of not using it (see equation 11). The smaller the value, the longer the ANN may take to learn. In practice, we should adjust the value of η throughout the learning process, using a larger η initially to increase the speed of the learning process. When the cost function noticeably reduces and the weights and biases settle, we reduce the learning rate to prevent drastic changes.

5.2.2 Stochastic Gradient Descent with Backpropagation

Backpropagation is a form of gradient descent where we work backwards through the ANN to evaluate errors and make adjustments [3]. Back-propagation enables us to find $\bigtriangledown C$, recall equation 10, for the entire network, i.e. $\frac{\partial C}{\partial w}$. In this subsection we will look at the calculation of these derivatives. We will set the foundations with the weights, and we can then translate equations easily for biases.

To begin the calculations, lets consider a simple ANN with four layers, and one neuron in each, as shown in figure 9.

Figure 9: Single Layer Network

C is a function of a^L, w^L, a^{L-1}, and b^L , i.e.: $C(a^L, w^L, a^{L-1}, b^L)$. By the chain rule:

$$\frac{\partial C_k}{\partial w^L} = \frac{\partial z^L}{\partial w^L} \frac{\partial a^L}{\partial z^L} \frac{\partial C_0}{\partial a^L} \tag{13}$$

Where k refers to the piece of training data. Using the definitions of a^L, z^L, C_k (see section 13.3.1 in the appendices), we see that

$$\frac{\partial C_k}{\partial w^L} = 2a^{L-1}(a^L - y)\sigma'(z^L) \tag{14}$$

Equation 14 tells us how a change to a particular weight, w_L, will affect the cost for the kth training example. To calculate the cost for all of the training data, C, we calculate the average of all costs from the training data using equation 9 [1]. That is, from k = 1 to k = n.

$$\frac{\partial C}{\partial w^L} = \frac{1}{n}\sum_{k=1}^{n}\frac{\partial C_k}{\partial w_L} \tag{15}$$

Calculating $\frac{\partial C_k}{\partial b^L}$ uses the same process and has a similar result. However, as $\frac{\partial z^L}{\partial b^L} = 1$:

$$\bigtriangledown C = \frac{\partial a^L}{\partial b^L}\frac{\partial C_0}{\partial a^L} \tag{16}$$

So far, we have only focussed on the output layer. Consider figure 10, showing the relationship between C and different layers of the ANN in 9. We see that C is influenced by all elements of the network, but less so as we go further back through the network (that is, from the output, L, to earlier layers, say $L-2$). We deduce that w^L has more influence on C than w^{L-1} .

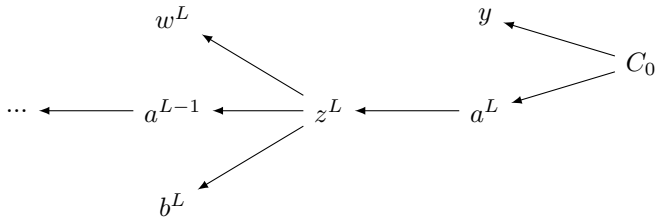


Figure 10

We cannot use the same derivative, equation 14, for elements in the hidden, or input, layers. However, similar to equation 13, we can use the chain rule to obtain:

$$\frac{\partial C_k}{\partial w^{L-1}} = \frac{\partial z^{L-1}}{\partial w^{L-1}}\frac{\partial a^{L-1}}{\partial z^{L-1}}\frac{\partial z^L}{\partial a^{L-1}}\frac{\partial a^L}{\partial z^L}\frac{\partial C_0}{\partial a^L} \tag{17}$$

Now that we can calculate $\bigtriangledown C$ for a network with one neuron in each layer, it is easy to translate this to a multilayer network with one or more neurons in each. Using equation 13, we determine:

$$\frac{\partial C_k}{\partial w^L_{jk}} = \frac{\partial z^L_j}{\partial w^L_{jk}}\frac{\partial a^L_j}{\partial z^L_j}\frac{\partial C_0}{\partial a^L_j} \tag{18}$$

In section 6, I will demonstrate briefly my coding of an ANN using backpropagation.

6 ANN: Coding a Neural Network

It is common that the processes discussed in this report will be performed by a machine as, unless the network consists of a handful of components, it is not feasible to compute by hand. Common coding languages to use include Python, C++, and Java [13]. I have chosen C++ due to it's higher performance power.

6.1 Perceptron Network

Using the simple network modelled in section 2.1.1, I created a piece of code in C++ to model this and perform the learning process. You may find the code in 13.4.1, and figure 11 shows a diagram of the processes.

The learning process of this code involves backpropagation using equation 14, where only layers L and $L-1$ are considered. This was a simple network, but effective to build upon for multilayer networks. The code was successful in the learning process, as confirmed by my supervisor.

6.2 Multilayer Network

This refers to the code in the appendices: section 13.4.2.

Next, I created a code that accounts for multiple layers. This code has four inputs, A, B, C, D and two outputs W, B. The four inputs represent a 4 x 4 pixel image, with grayscale values. Using an average rule, the network decides whether the image was mostly black pixels, or mostly white.

Once this network is established, the reader may develop the code to structure the network to suit their needs. Future studies for myself would include a code whose core learning component is not restricted to a certain amount of neurons - in this case 4 outputs.

A k-dimensional vector is created to store all weights, and m-dimensional to store all biases, where k and m are the amount of weights and biases in the network respectively. Random weights in the range 0 to 9 are then assigned. At first, this range was larger - but the network was not reducing the cost. Upon research [18], I found that the amount of weights is typically in the range $[-1, 1]$, but can vary. I had little variation in the outputs when adjusting the initial random weight range between [-1, 1] and [0,9]. I have left the weight range as [0,9] for now, however, this may be changed

by the reader as necessary.

Weights and biases are kept in separate vectors for the code as, although the learning process is the same, the equations for backpropagation vary - see equation 16.

The code works using a loop which restricts the amount of iterations, again set by the user manually.

Within the loop, the random grayscale values of the inputs are assigned. I created the following rule for the outputs:

$$\frac{A+B+C+D}{4} > 0.5 \implies \text{output black} \tag{19}$$

$$\frac{A+B+C+D}{4} \leq 0.5 \implies \text{output white} \tag{20}$$

The network then runs the randomly assigned values using the sigmoid function, equation 2 and random weights and biases.

Afterwards, the contents of the loop move on to gradient descent, following the steps laid out to the reader in section 5.2.2. New vectors of k and m dimensions respectively are created to hold the values of $\bigtriangledown C$. The expected outputs, y are assessed using the rule above. Rather than perform differentiation directly in the code, we develop relations specifically for this network, as you will see in the appendices, section 13.4.2

The final step of the loop is to implement the changes to the weights and biases using the respective $\bigtriangledown C$ vectors. These weights and biases are used for the next iteration with again random grayscale values.

In future studies, I would require my code to pull data directly from data sets, and learn from these.

The results of my code show improvements in the network, however the cost function does not significantly reduce, even with millions of iterations. Given more time, we would explore this. However, for now it is apparent that my network's process of gradient descent is not successful for the required purpose.

The topic of neural networks and developing them becomes a topic of computer science, rather than mathematics. So, we shall begin looking at other methods of image recognition shortly. Prior to this, we shall visit some types of neural network and how they may impact the process.

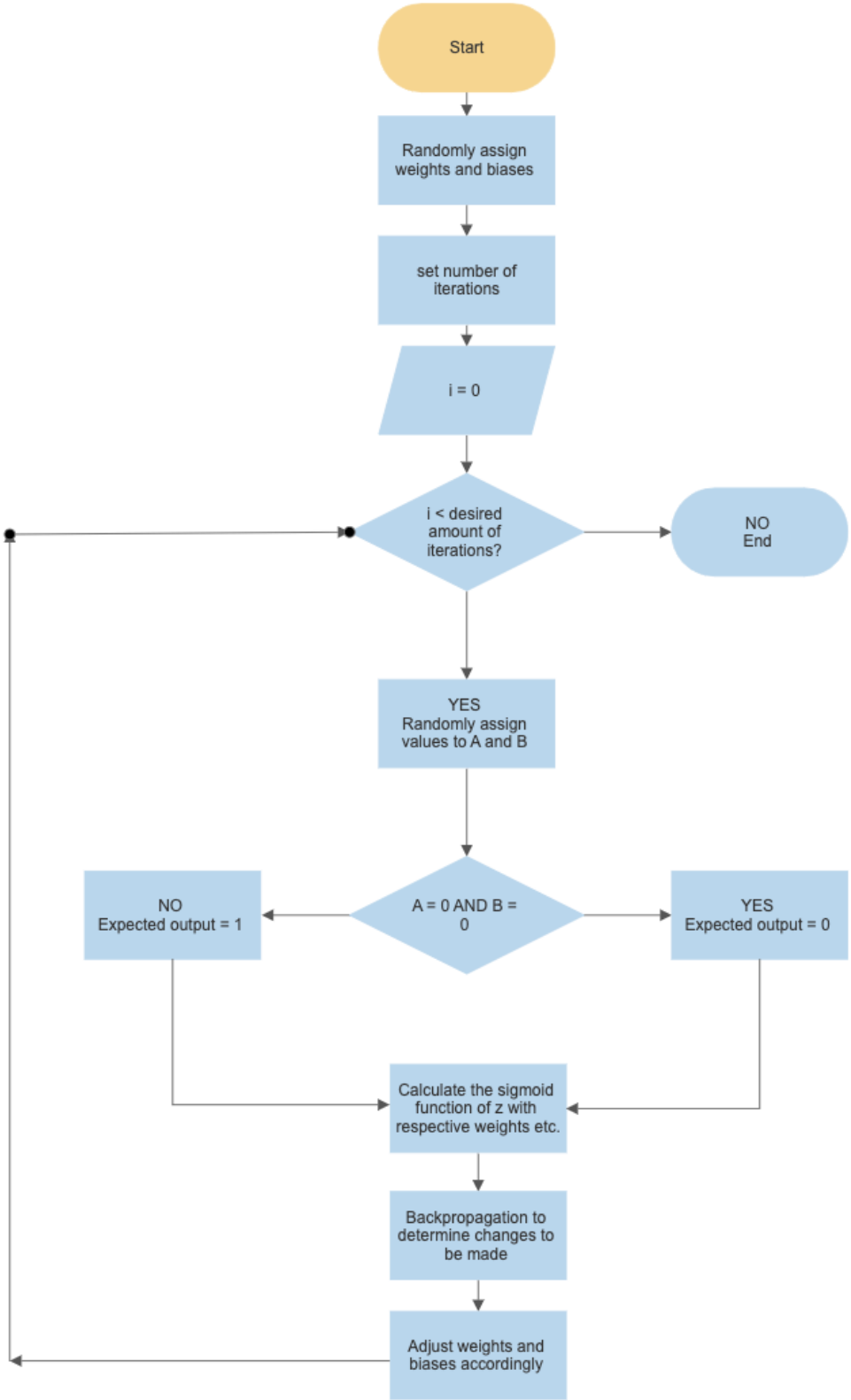


Figure 11: Process of the Perceptron Code

7 Types of Neural Network

Before moving on to handwriting recognition methods beyond neural networks, I shall introduce and compare some types of neural networks and how they may play a role in image recognition.

7.0.1 ANNs

So far we have considered ANNs; these feed data forward through the network. They have common uses in basic image recognition such as handwritten digits [19]. They are universally used, primarily for their use of activation functions, and ability to map non-linear functions with the use of weights and biases.

ANNs have restrictions though, such as the input being 1-dimensional. For example, the images we have looked at throughout have been split into a 1-dimensional vector with 784 inputs. This restricts the learning process, as the ANN can only train on the one parameter. Additionally, spacial features (such as the placement of pixels), are lost within an ANN. For image classification, this is not too vital. However, if we required our network to tell us the positioning of particular pixels, or wanted it to classify depending upon the positioning of pixels, it could not.

7.0.2 RNNs

Another type of neural networks is a recurrent neural networks (RNN), commonly used in audio and text. Within the hidden layers, data loops back to neurons, capturing sequential data [20]; something ANNs cannot do. For example, the dependency on certain words when using predictive text is sequential data. The issue of digit recognition from a set of independent data would not require this. However, it may be used if this project was taken further to recognise words from handwriting samples.

7.0.3 CNNs

The next type to mention are convolutional neural networks (CNN). CNNs use kernels (filters) to extract features, and are more commonly used for sophisticated image processing, considering images of objects with RGB colour values rather than grayscale. For example, you may input an image of an animal, and the CNN could classify whether it was a cat or not. CNNs can be used for handwriting recognition [21], but for the purpose of a mathematical report I do not explore this.

7.1 Types of Learning

It is also important to identify the types of learning. Throughout, we have used supervised learning. [3]. **Supervised** learning is where the network is provided with examples of correct outputs (our training set), so it can learn where it goes wrong and generalise responding.

On the other hand, we have **unsupervised** learning, where networks are not provided with correct examples. Rather, the networks find patterns so that certain inputs can be categorised. For example, an unsupervised digit recognition network may categorise images with certain average heights, or widths. This approach is sometimes called density estimation.

There is a mix between supervised and unsupervised, that is **reinforced** learning. The network is told when it is wrong, but not how to improve. The network then tries different possibilities until it reaches a correct answer.

7.2 Generating Images using Neural Networks

In subsection 7.0.1, we discussed a restriction of ANNs - they cannot capture spacial features. There are other types of neural networks that can, and we may use these to assess what makes up the average digit, and reconstruct images. In the world of computer science, there are two main networks to perform this process: autoencoders, and generative adversarial networks (GANs). GANs are a type of CNN [22]. Autoencoders focus on decoding and encoding a network. Or, in our case, classifying digits, and recreating them. GANs on the other hand are focussed more on regenerating data, and classifying images as real, or re-generated. As we are focussing on recognition and processing, we shall explore autoencoders.

7.2.1 Autoencoders

Autoencoders are a form of unsupervised neural network that works to reconstruct data to represent the original data [23]. Autoencoders are commonly used for denoising, but have been used to generate handwritten digits after training on a data set such as the MNIST [7], such as figure 13 reconstructed from figure 12. You will see that the reconstructed images are themselves noisy, and may not be clear to use, for example when making a font. In section 8, we will see a method of edge detection filtering that enhances such images. Should this project be revisited, we may use autoencoders directly, and compare their outputs with GANs.

Autoencoders are more complicated Neural Networks than the ones discussed earlier in the report, for example figure 1. Their basic structure is similar however, shown in figure 14, consisting of an input and output layer, and layers in between that are structured like a bottleneck [23]. Loss functions are used here too, ranging from the mean squared error which we have seen, to Kulback-Leibler Divergence (KLD) [24] for variational autoencoders.

The KLD compares two probabilistic distributions, and may be used to compare actual outputs, t_L, to the desired output, y_L. For two discrete distributions, P and Q, the KLD between them is given by:

$$D_{KL}(P||Q) = \Sigma_x P(x) log \frac{(P(x))}{(Q(x))} \tag{21}$$

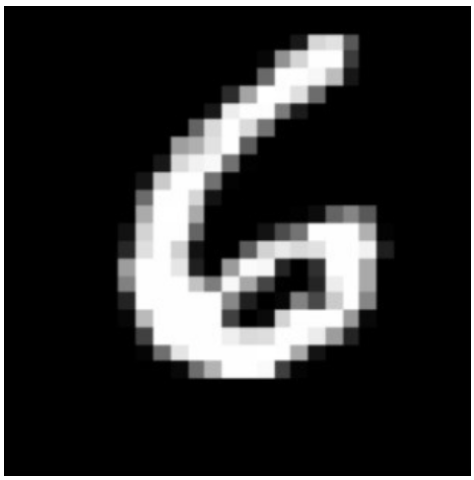


Figure 12: Original

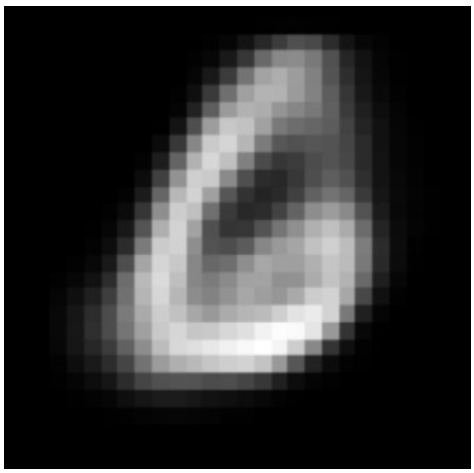


Figure 13: Reconstructed

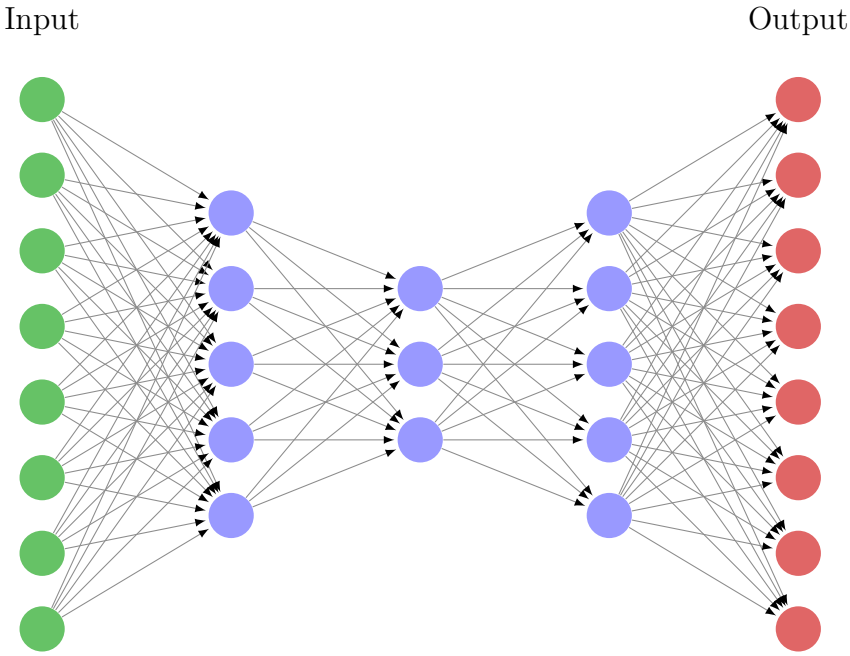


Figure 14: Autoencoder Structure

8 Edge Detection Filters

So far we have considered using ANNs to identify and classify handwritten digits. However, in my research plan (see appendices, 13.1) I laid out my motivation for researching image recognition. This was that, in year two of my degree, I produced a research poster on the Sobel Filter [2], section 13.2 in the appendices, an edge detection filter (EDF). Though I have deviated from this process, I want to touch upon such methods, as they may be used with neural networks, such as autoencoders, to refine and enhance images and outputs.

EDFs detect boundaries with significant differences in brightness, typically using grayscale. For example, figure 15 [25], which uses the canny edge detection algorithm. I propose the use of edge detection filters on the outputs of autoencoders, section 7.2.1, to refine images. The outputs may then be used, for example to create a font.

8.0.1 The Sobel Filter

The Sobel Filter, or the Sobel–Feldman Operator (SFO), uses gradient filters to refine images [26]. It builds upon earlier work estimating the gradient vector using the following masks [27]:

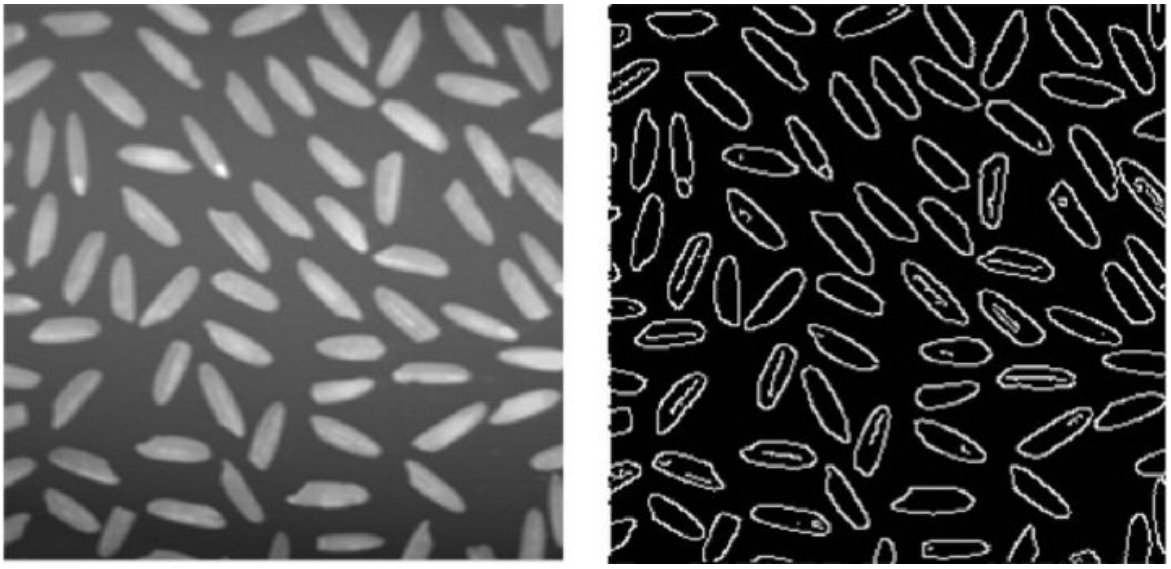


Figure 15: Edge Detection Example

$$\triangle_x = \begin{bmatrix} -1 & 0 & 1 \\ -a & 0 & a \\ -1 & 0 & 1 \end{bmatrix} \quad (22) \qquad \text{and} \qquad \triangle_y = \begin{bmatrix} -1 & a & -1 \\ 0 & 0 & 0 \\ 1 & a & 1 \end{bmatrix} \quad (23)$$

where $a > 0, a \in \mathbb{R}$

The SFO uses $a = 2$, i.e. the two 3x3 kernels:

$$\triangle_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad (24) \qquad \text{and} \qquad \triangle_y = \begin{bmatrix} -1 & 2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad (25)$$

We then use the kernels on the source image $\mathbf{A}$ as follows:

$$G_x = X * A, G_y = Y * A \quad (26)$$

where $*$ is the convolution operation.

The result of equation 26 describes the differentiation in grayscale values between pixels. A larger values means a more noticeable edge due to greater differences in value. Performing equation 26 on the entire image allows us to replace the previous grayscale values with the result. For example, the SFO applied to figure 13 returns figure 17. Although figure 17 is noticeably clearer than figure 13, it still has noise. To overcome this, we may us an averaging filter prior to the SFO [28].

In figures 16 and 18, you see the results of two other EDFs that perform, and produce outputs, similar to the SFO. The Robert Cross EDF uses two 2x2 kernels, and the Laplacian uses a 3x3 mask. Future work on this project may include the comparison of such methods on the results of an Auto Encoder.

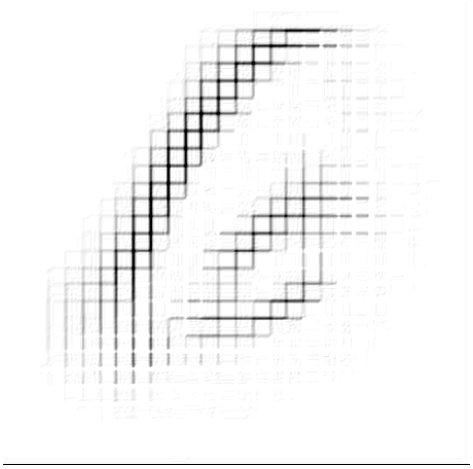


Figure 16: Figure 13 reconstructed using Roberts Cross EDF [29]

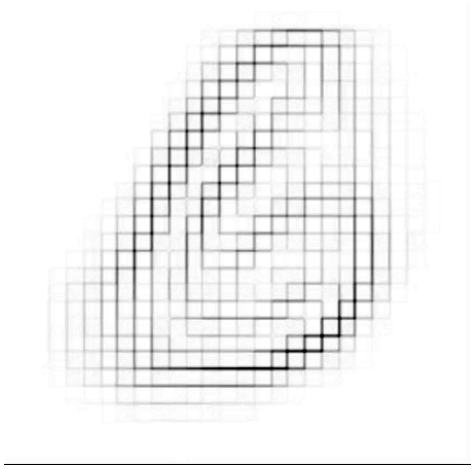


Figure 17: Figure 13 reconstructed using Sobel Feldman EDF [29]

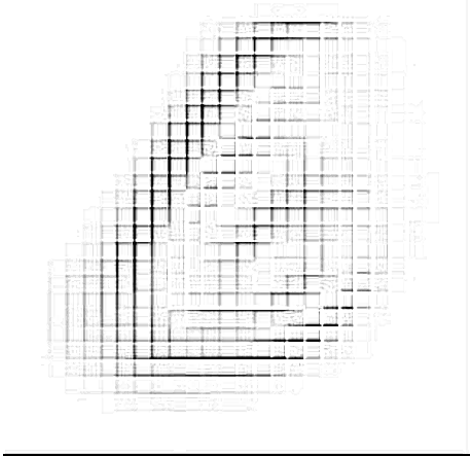


Figure 18: Figure 13 reconstructed using Laplacian EDF [29]

9 Interpolation

In this chapter, we explore how polynomial interpolation may be used for image recognition. Interpolation provides a means of estimating the function at intermediate points.

To begin, we may consider handwritten digits as graphs. Treating the interface as an axis, we can take the coordinates of points along the ink trace. For example, in figure 19 we have two handwritten characters. Figure 20 shows how we may record the coordinates, with the yellow dots showing the trace, and the red dots showing where the pen starts touching the interface, and where it stops.

In this section we explore methods using the coordinates to recognise images. Later,

in section 10.2, we look at how we may record the coordinates to create a database.

Figure 19: handwritten characters

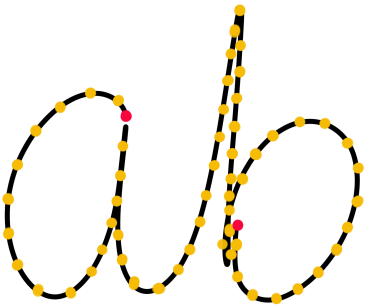


Figure 20: plots

9.1 Lagrange

The first method we look at is Lagrange interpolation to set the foundations for using interpolation in image recognition. Lagrange interpolation allows us to represent a collection of points as a polynomial using the following:

$$P(x) = \sum_{i=1}^{n} P_i(x) y_i \tag{27}$$

where $P_i(x) = \frac{f(x)}{(x-x_i)f'(x_i)}$ and $f(x) = (x - x_1)(x - x_2)...(x - x_n)$

We begin with an example.

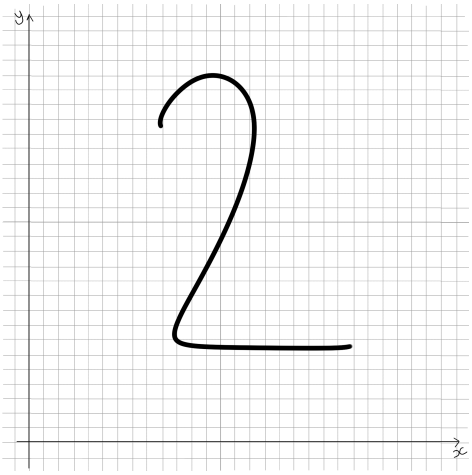


Figure 21

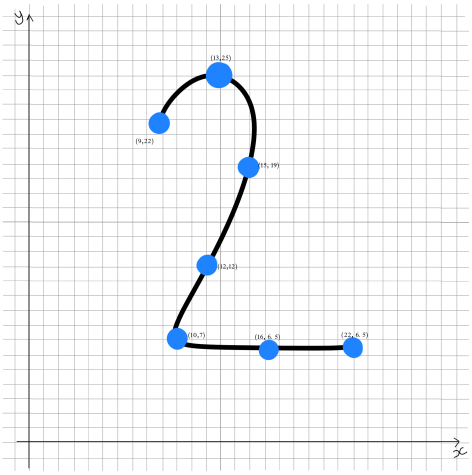


Figure 22

Example 7 *Consider figure 21. We begin by taking 7 random points, as displayed in figure 22:*

(9,22), (13, 25), (15, 19), (12, 12), (10, 7), (16, 6.5), (22, 6.5).

Using equation 27: $P(x) = -0.00885565x^6 + 0.747309x^5 - 25.6916x^4 + 460.525x^3 - 4538.94x^2 + 2332.1x - 48805.2$
(see section 13.3.2 for full calculation).

Considering the polynomial $P(x)$ from 7, we see that the graph does not exist within the area the plots are taken from (roughly the square connecting the points $(9, 6), (9, 25), (23, 25), (23, 6)$). This is likely down to our small selection of points. However, including more points may become problematic, as the increased points will increase the degree of our polynomial $P(x)$; increasing the difficulty and time for calculation. There is the possibility of using a programming language, such as Python [30] to overcome this. However, what is it we are looking for? We may be able to replicate the digits with polynomials to look similar - but that doesn't get us any closer to recognising the image other than with the human eye. Rather, we move onto another method of interpolation, Chebyshev polynomials.

9.2 Chebyshev

Chebyshev polynomials are orthogonal polynomials used in many function approximation situations. I have found numerous texts [31], [32] supporting the use of Chebyshev polynomials, especially for more complex functions. However, the main text I focus on is that by Char and Watt [5].

We will first explore what Chebyshev polynomials are, and some foundations necessary to use them in image recognition. A chebyshev polynomial of degree n is given by the formula $T_n(x) = \cos(n \cos^{-1} x)$, [33]. We may express this as:

$$T_n(\cos\theta) = \cos n\theta \tag{28}$$

such that $T_0(x) = 1$ $T_1(x) = x$ $T_2(x) = 2x^2 - 1$ $T_3(x) = 4x^3 - 3x$ ... See 13.3.4 for a step-by-step to the above.

Using equation 28, we may obtain *(see appendices, section 13.3.4)* the following recurrence relation:

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x) \tag{29}$$

9.2.1 Using Chebyshev Polynomials to approximate functions

A typical polynomial is of the form $f(x) = ax^n + bx^{n-1} + ... + yx + h$ where $a, b, y, h, n \in \mathbb{R}$. In essence, linear combinations of $1, x, x^2, x^3, ...$ To use Chebyshev polynomials, we consider our functions as linear combinations of $T_n(x)$.

For any function f(x'), we express it as [5]:

$$f(x') = \Sigma c_k T_k(u) \tag{30}$$

where $u = \frac{2x'-a-b}{b-a}, x' \in [a, b]$.

We use the transformation $u = \frac{2x'-a-b}{b-a}$, as it maps $x' \in [a, b]$ to $u \in [-1, 1]$, as required by the definition of Chebyshev polynomials, equation 28. Equation 30 allows us to solve for the coefficients, c_k; this can be done using Chebyshev nodes:

$$x_i = cos(\frac{2i-1}{2n}\pi), i = 1, ..., n \tag{31}$$

where x_i is the root of $T_n(x)$.

Example 8 *Let $f(x) = 2x^4 + 3x^3 - 2x^2 + x, x \in [-2, 1]$. We will approximate f(x) using a linear combination of Chebyshev polynomials for $N = 0, 1, 2, 3, 4$*
We use the translation $u = \frac{2x-a-b}{b-a} = \frac{2x-1}{3}$.

Using equation 31, we obtain the Chebyshev nodes (to 3.s.f):

- $u_1 = cos(\frac{1}{10}\pi) = 0.951$,
- $u_2 = cos(\frac{3}{10}\pi) = 0.588$,
- $u_3 = cos(\frac{5}{10}\pi) = 0$,
- $u_4 = cos(\frac{7}{10}\pi) = -0.588$,
- $u_5 = cos(\frac{9}{10}\pi) = -0.951$

Next, we use $u = \frac{2x-1}{3}$ to obtain the respective x values, and then input this in to $f(x)$ to obtain the corresponding values of y = f(x). Using values of f(x), and calculating our values of c_n from equation 30, we may approximate $f(x)$ as follows:
$f(x) = 10.5T_0(u) + 21.1T_1(u) + 11.5t_2(u) + 5.91T_3(u) + 1.07T_4(u)$, *or* $f(x) = 8.56x^4 + 23.64x^3 + 14.44x^2 + 3.37x + 0.07$

Full working in section 9 in the appendices.

9.2.2 Image Recognition using Chebyshev Polynomials

Starting with our handwritten digit, we take continuous plots as the writer draws, similar to 22, but with more points. However, we record the trace now as a sequence of (x_i, y_i, t_i), where $t_0 = 0, t_1 = 1$. I.e. the pen hits the interface at $t = 0$, and finishes/lifts off at $t = 1$.

We use linear interpolation on x and y separately from our traces, (x_i, y_i, t_i), to obtain $X(t) = x_i + \frac{x_{i+1}-x_i}{t_{i+1}-t_i}$ and $Y(t) = y_i + \frac{y_{i+1}-y_i}{t_{i+1}-t_i}, t_i \leq t < t_{i+1}$. Using this, we create the Chebyshev Series:

$$X(t) = \Sigma_{i=0}^{\infty} \alpha_i T_i(t), Y(t) = \Sigma_{i=0}^{\infty} \beta_i T_i(t) \tag{32}$$

From here, the reader may use the techniques discussed in section 9.2.1, and example 8, to obtain the coefficients α_i and β_i for equation 32. The series approximations of X and Y may then be compared with others and classified, using techniques not covered in this report. Future work beyond this report should expand upon this.

10 Creating Data Sets

As discussed in the introduction, section 1.2, we have used the MNIST database [7] for handwritten digits. We will discuss now how to capture such data ourselves, from the grayscale values of 28 x 28 pixel images, to coordinates as seen in 10.2.

10.1 Grayscale Pixel Values

To capture data manually, we may use Excel, or another similar worksheet tool. For example, figure 23 shows an excel spreadsheet restricted to 28 rows and 28 columns, with possible inputs $[0, 1]$. We can use this to represent images and their pixel grayscale values by 'drawing' the digit directly into the spreadsheet 24, or by putting your image in the background and inputting the corresponding values, as done in figure 24 using figure 21. This, however, is not practical.

To automate this process, increasing processing time and removing human error, we may use a coding language, such as Python, and import the pixel values.

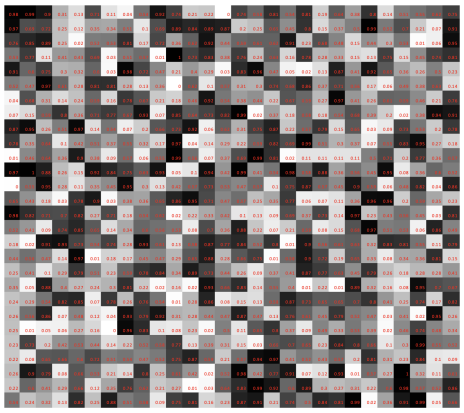


Figure 23: Random grayscale image in excel

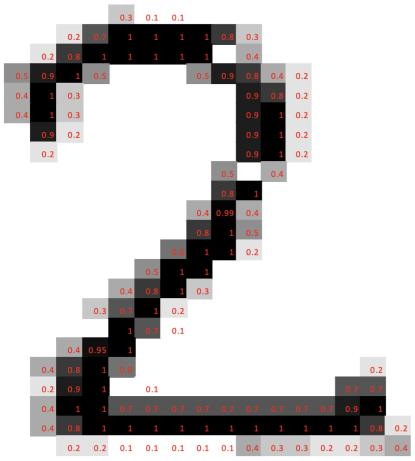


Figure 24: Digit 2 in Excel

10.2 Graph Plots

For graph plotting, there are tools [34] to manually record coordinates from images. Alternatively, we can overlay an xy axis over images and determine their coordinates. When requiring points in the form (x_i, y_i, t_i), it is also necessary to record the time of a trace, and then normalise this to be within the range $[0, 1]$. There are, however, tools to automate this process. We shall explore a particular example next, section 10.2.1.

10.2.1 InkML

InkML uses an electronic pen-based interface, such as a tablet, to track pen movements, recording their (x, y) coordinates to create a sequence. An example of an InkML file is below [6]. We may then collate this data with several handwriting samples to create our database ready for interpolation.

```
<ink xmlns="http://www.w3.org/2003/InkML">
   <trace>
      10 0, 9 14, 8 28, 7 42, 6 56, 6 70, 8 84, 8 98, 8 112, 9 126, 10 140,
      13 154, 14 168, 17 182, 18 188, 23 174, 30 160, 38 147, 49 135,
      58 124, 72 121, 77 135, 80 149, 82 163, 84 177, 87 191, 93 205
   </trace>
   <trace>
      130 155, 144 159, 158 160, 170 154, 179 143, 179 129, 166 125,
      152 128, 140 136, 131 149, 126 163, 124 177, 128 190, 137 200,
      150 208, 163 210, 178 208, 192 201, 205 192, 214 180
   </trace>
   <trace>
      227 50, 226 64, 225 78, 227 92, 228 106, 228 120, 229 134,
      230 148, 234 162, 235 176, 238 190, 241 204
   </trace>
   <trace>
      282 45, 281 59, 284 73, 285 87, 287 101, 288 115, 290 129,
      291 143, 294 157, 294 171, 294 185, 296 199, 300 213
   </trace>
   <trace>
      366 130, 359 143, 354 157, 349 171, 352 185, 359 197,
      371 204, 385 205, 398 202, 408 191, 413 177, 413 163,
      405 150, 392 143, 378 141, 365 150
   </trace>
</ink>
```

11 Conclusion

We have seen how we may train and use ANNs to classify digits from a dataset throughout this report. We explored how to structure a practical ANN to cater to our needs in section 3 and created an optimised network in example 4. Using stochastic gradient descent in section 5.2.2, we saw how we might use the proposed network to develop a NN from our set of training data.

In section 7, we saw the different types of neural networks and how they may be used within image recognition. We saw the use of autoencoders to regenerate digits from our network post-training. Later, in section 8, we saw how we might use edge detection filters to enhance the outputs of autoencoders. Future studies may focus on the impact of combining the two processes.

For interpolation methods in section 9, I focussed on Chebyshev polynomials. Given more time, I would have determined a process to use these from the initial data sets to the final classification result. I would recommend future studies include more profound research on this, as well as exploring other methods of interpolation, such as splines which were mentioned by Marsland [3].

Additionally, there are factors within this project that could affect the classification process that we have accepted following the recommendation from literature. For example, we use the sigmoid function, section 4.3.1, for our ANNs. However, we did not practise using alternatives, such as ReLU, to assess the impact mathematically. Future studies may compare methods such as this and their suitability. Additionally, variables such as η and the number of nodes should be scrutinised similarly.

From the report's contents, we have a robust process that is statistically proven to classify handwritten digits: neural networks. In hindsight, I should have spent more time on interpolation. This appears to be an emerging method with more room to develop by changing the interpolation method or adapting the parameters used.

This project aimed to understand how mathematics may be used to process and classify images. We have established two methods to classify images: interpolation and neural networks and developed a thorough understanding of these. To better understand the two methods' performance compared, further studies may address the process of image recognition from start to end, using additional tools such as InkML and autoencoders.

Overall, I hope you have developed an understanding of the variety of methods avail-

able to mathematically categorise handwriting. Thank you for reading.

12 Acknowledgements

I want to take this opportunity to thank my partner, Sam Robertson, for keeping me sane and on track with my studies and for simply everything you do for me. Your enthusiasm when listening to the mathematics of image processing cannot be matched. I can't wait to return the favour and support you throughout your physics degree, and I look forward to seeing your final year report within the school in the years to come. Again, thank you.

I'd also like to thank my parents, Sharon and Wes, for their support throughout my undergraduate studies. From helping with applications to University in 2017 to supporting me with love and encouragement to return to studies in 2019.

Finally, thank you to my best friends, Anya and Oscar, for always listening to my maths orientated talks and providing endless emotional support.

I cannot thank you all enough for the love and support you have given me.

13 Appendices

13.1 Research Plan

16630927 - MTH3007M

Mathematics of Image Processing and Recognition: Research Plan

Project Description

Image processing and recognition are about breaking down images into numerical descriptors and using this to recognise patterns. For example, there are many systems now that can spot a face from a photograph, such as Microsoft's face detection and recognition technologies [1]. The wide range of applications behind this topic is my primary motivation for selecting this project title.

This project will research the mathematics behind image processing and use technology to practice image processing and pattern recognition. I have divided this topic into three sections; the mathematics behind feature extraction (converting images to appropriate numerical descriptors), pattern recognition, and applications. Within the applications section, I intend to look at the mathematical foundations and how this has evolved with technology, then move on to current applications and intend to work on a practical application using a computer language, such as Python or C++.

The project was provided with the example application of categorisation of Atomic Force Microscope images. However, I would like to focus on everyday examples such as medical or social uses. We have already introduced facial recognition [1], which is also helpful for unlocking electronic devices such as our phones. Image recognition is also used in traffic management, such as road sign recognition [2]. Medically there are applications in diagnosis, such as organ detection in tumour patients [3], and refining ultrasound images [4].

In 2021 I undertook the module MTH2006M: Industrial And Financial Mathematics, where we were introduced to industrial applications of mathematics. One that particularly fascinated me was a talk given by Natasha Maurits, UMCG, 23 February 2021 [4], on the enhancement of ultrasound images using a feature extraction method, the Sobel Operator, to diagnose ALS. The Sobel Operator is an edge detection filter that can take noisy images and refine them, allowing for a more definite diagnosis. I chose this application among others to research and finalised this with a poster on the topic.

Literature Review

Bishop states that understanding statistical pattern recognition is necessary to understand neural networks [5]. As such, this book provides a solid statistics foundation, focusing on Baye's Theorem (chapter 1), probability density estimation (chapter 2), and single-layer networks (chapter 3). I will be starting my project research by studying these chapters. Additionally, this book will research feature extraction (chapter 7) and pattern recognition (chapter 8: learning and recognition).

Pattern Classification [6] is a book recommended by Bishop [5]. It, too, has a statistical base but focuses more on techniques for this project. In summary, it has Bayesian Decision Theory (chapter 2), Non-Parametric Techniques (chapter 4), Neural Networks (chapter 6), machine-learning (chapter 9), and more. Pattern Recognition [7] is a book similar to this. It starts with Bayes decision theory, though it focuses more on feature extraction and has several chapters on cluster analysis. This will not necessarily be a book to start with, but I see this being useful when researching pattern recognition. I anticipate both of these books being the main resources of the project.

I found it necessary to find a preliminary report on neural networks, as they may be a vital part of my project towards the computer application section. They are based on the human brain and allow computers to recognise patterns [8]. I also found a helpful article [9] from a software developer that explains neural networks from a mathematical standpoint and explains them well too. These resources will be necessary to support my understanding alongside the books already mentioned.

Another book I have found, Intelligent Biometric Techniques in Fingerprint and Face Recognition [10], focuses on applications. This application can be used in security and social applications such as organising photographs by faces [1]. I intend to read this book alongside others, as it does not seem as maths-intense as it is descriptive of processes, to provide a thorough understanding of an application.

When it comes to the practical side of my project, I have found resources from Mathworks on Image Filtering [11], and also region of interest (ROI) based processing [12]. Resources such as this will help explore techniques and different types of feature extraction.

We have already introduced facial recognition [1], which is also helpful for unlocking electronic devices such as our phones. Image recognition is also used in traffic management, such as road sign recognition [2]. Medically there are applications in diagnosis, such as organ detection in tumour patients [3] and refining ultrasound images [4].

Conducting Research

The majority of my research will be conducted through reading the material mentioned in the literature review and any new resources I find and then collating and relating the information found to be relevant for the project. Towards the end of the module, I will be moving to computer applications, using Python, Julia, and C++. My project report will be written in LaTeX using the online editor Overleaf [13].

Supervisor Meetings

I have arranged to meet with my supervisor, Dr Matthew Watkins, every Tuesday (apart from Christmas break) between 10:30 - 10:50.

Equipment and Costs

There are no requirements beyond standard equipment already available. I will be using a laptop, electronic tablet, and my logbook to conduct my research, alongside books from the library. I also have access to online resources such as library databases that the university subscribes to. When I cannot use my laptop, I will be using the computers in the computer lab INB2305, with access to all necessary C++, Python, and MATLAB software. For storage, I will use online cloud storage, such as my OneDrive university.

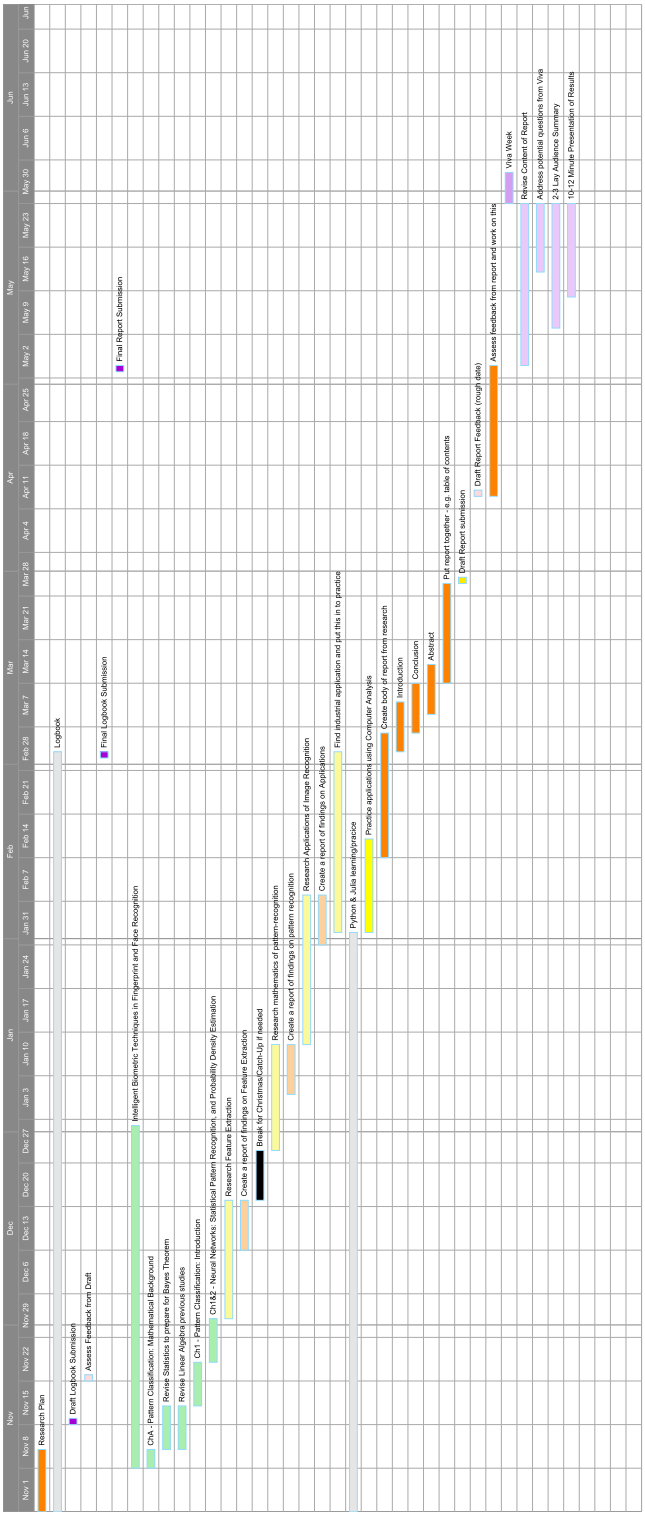
As my research relies heavily on technology (project written in LaTeX, research done on my laptop or tablet, using computer software for applications), I will be saving work onto my university OneDrive wherever possible.

Action Plan

My research will begin with reviewing the literature mentioned, particularly gaining a solid mathematical basis in statistics and linear algebra, as recommended by my supervisor. I then plan to research feature extraction, followed by pattern recognition, and then applications. Afterwards, my research will culminate in putting an application into practice, using those applications found in my research.

Alongside my research, I will continue learning Python and Julia for the computer application section as recommended by my supervisor. I will, too, be building upon my Python and C++ knowledge through my current study of the module MTH3007M - Numerical Methods.

Overleaf is a copy of the action plan I have created using Smartsheet [14]. You can also view a larger version, spread across three pages, in the appendices.


Nov
Dec
Jan
Feb
Mar
Apr
May
Jun
Nov 1
Nov 8
Nov 15
Nov 22
Nov 29
Dec 6
Dec 13
Dec 20
Dec 27
Jan 3
Jan 10
Jan 17
Jan 24
Jan 31
Feb 7
Feb 14
Feb 21
Feb 28
Mar 7
Mar 14
Mar 21
Mar 28
Apr 4
Apr 11
Apr 18
Apr 25
May 2
May 9
May 16
May 23
May 30
Jun 6
Jun 13
Jun 20
Jun
Research Plan
Logbook
Draft Logbook Submission
Assess Feedback from Draft
Final Logbook Submission
Final Report Submission
Intelligent Biometric Techniques in Fingerprint and Face Recognition
Ch4 - Pattern Classification: Mathematical Background
Revise Statistics to prepare for Bayes Theorem
Revise Linear Algebra/previous studies
Ch1 - Pattern Classification: Introduction
Ch1&2 - Neural Networks: Statistical Pattern Recognition, and Probability Density Estimation
Research Feature Extraction
Create a report of findings on Feature Extraction
Break for Christmas/Catch-Up if needed
Research mathematics of pattern-recognition
Create a report of findings on pattern recognition
Research Applications of Image Recognition
Create a report of findings on Applications
Find Industrial application and put this in to practice
Python & Julia learning/practice
Practice applications using Computer Analysis
Create body of report from research
Introduction
Conclusion
Abstract
Put report together - e.g. table of contents
Draft Report submission
Draft Report Feedback (rough date)
Assess feedback from report and work on this
Viva Week
Réevse Content of Report
Address potential questions from Viva
2-3 Lay Audience Summary
10-12 Minute Presentation of Results

References

[1] "Group photos by faces", Support.microsoft.com, 2021. [Online]. Available: https://support.microsoft.com/en-us/windows/group-photos-by-faces-1ab09703-f0a6-5835-d27b-58672b23fdd2. [Accessed: 07- Nov- 2021].

[2] Gomes, Samuel Rebouças, Elizângela Cavalcanti Neto, Edson Papa, João Albuquerque, V.H.C. Filho, Pedro Pedrosa & Tavares, Joao. (2017). Embedded real-time speed limit sign recognition using image processing and machine learning techniques. Neural Computing and Applications. 28. 573–584. 10.1007/s00521-016-2388-3.

[3] K. Zhou, Medical Image Recognition, Segmentation and Parsing: Machine Learning and Multiple Object Approaches. London: Academic Press, 2015, pp. Pages 123-153.

[4] N. Maurits, Mathematics in Medicine, Guest lecture for Industrial And Financial Mathematics. 2021.

[5] C. M. Bishop, Neural Networks for Pattern Recognition, 1st ed. Oxford: Oxford University Press, 1995.

[6] R. O. Duda, P. E. Hart and D. G. Stork, Pattern Classification, 2nd ed. New York: Wiley, 2001.

[7] S. Theodoridis and K. Koutroumbas, Pattern Recognition, 4th ed. Amsterdam; London: Elsevier/Academic Press, 2009.

[8] "What are Neural Networks?", Ibm.com, 2020. [Online]. Available: https://www.ibm.com/uk-en/cloud/learn/neural-networks. [Accessed: 07- Nov- 2021].

[9] D. S K, "A Gentle Introduction To Math Behind Neural Networks", Towards Data Science, 2020. [Online]. Available: https://towardsdatascience.com/introduction-to-math-behind-neural-networks-e8b60dbbdeba. [Accessed: 07- Nov- 2021].

[10] L. Jain, U. Halici, I. Hayashi, S. Lee and S. Tsutsui, Intelligent Biometric Techniques in Fingerprint and Face Recognition, 1st ed. Boca Raton: CRC Press, 1999.

[11] "Image Filtering", MathWorks, 2021. [Online]. Available: https://uk.mathworks.com/help/images/linear-filtering.html?s_tid=CRUX_lftnav. [Accessed: 07- Nov- 2021].

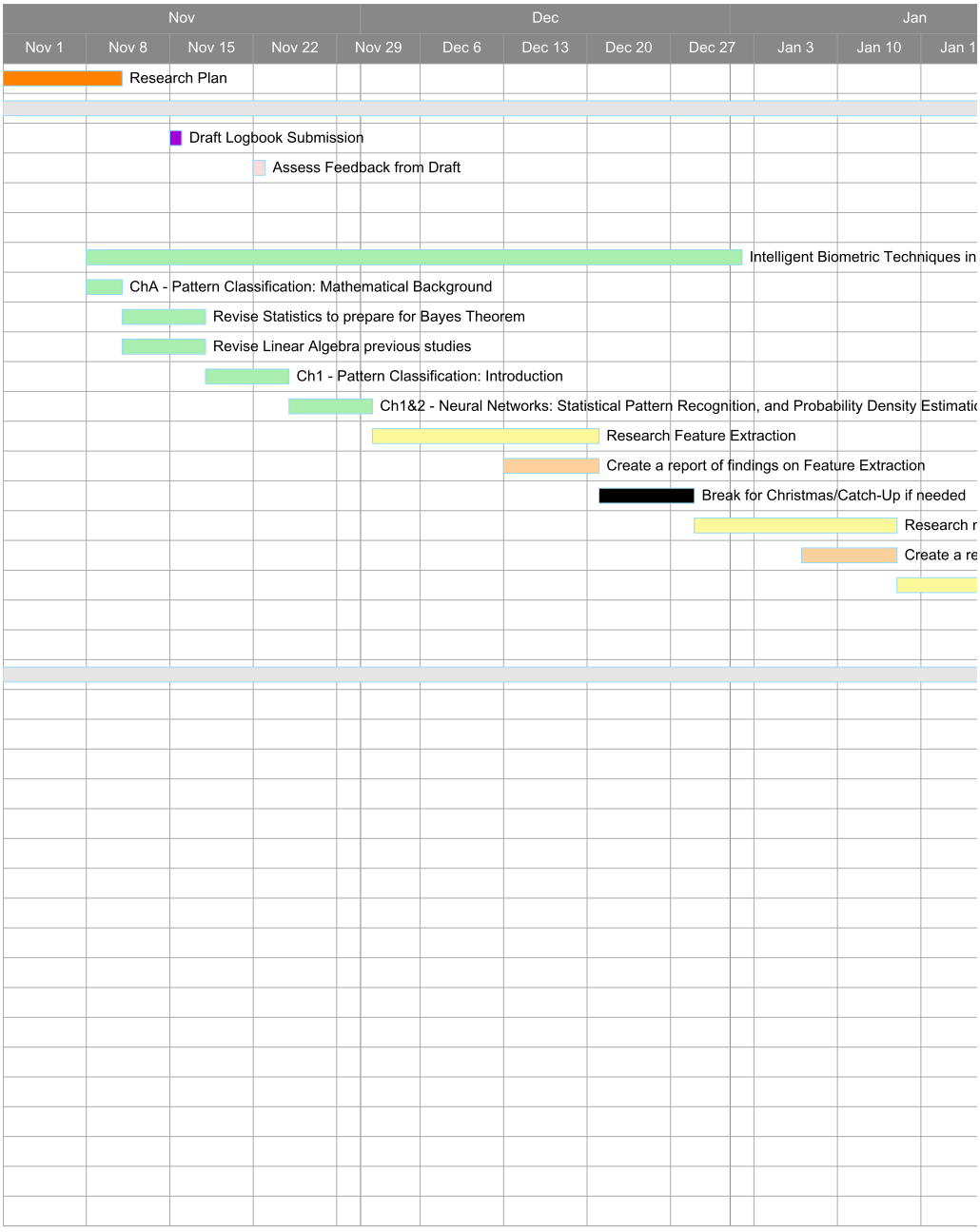
[12] "ROI-Based Processing", MathWorks, 2021 [Online]. Available: https://uk.mathworks.com/help/images/roi-based-processing.html [Accessed: 7 November 2021]

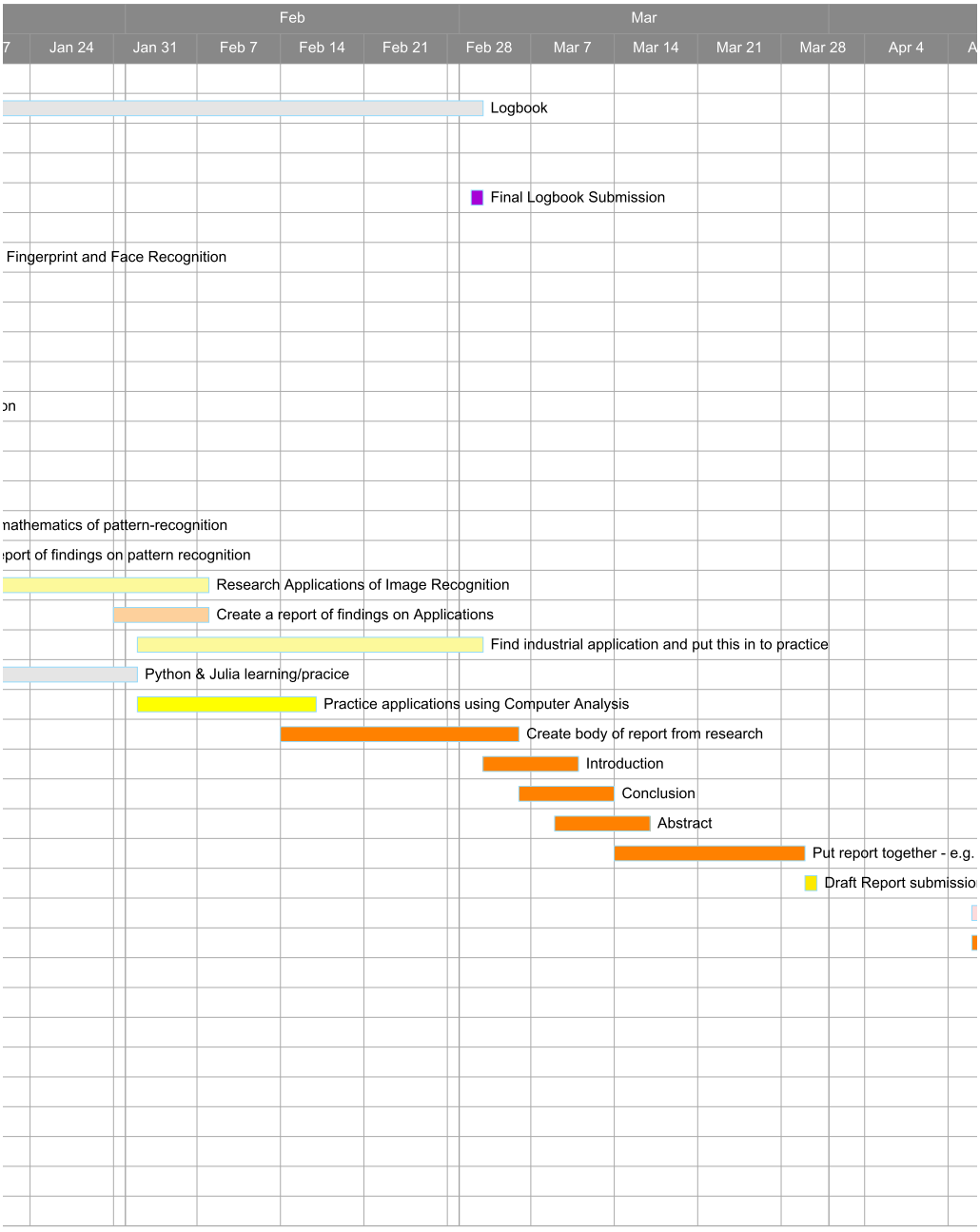
[13] "Overleaf, Online LaTeX Editor", Overleaf.com, 2021. [Online]. Available: https://www.overleaf.com/. [Accessed: 07- Nov- 2021].

[14] "Smartsheet Browser Compatibility — Smartsheet", App.smartsheet.com, 2021. [Online]. Available: https://app.smartsheet.com. [Accessed: 07- Nov- 2021].

Apendices

The next three pages contain the enlarged action plan [14].


Nov
Dec
Jan
Nov 1
Nov 8
Nov 15
Nov 22
Nov 29
Dec 6
Dec 13
Dec 20
Dec 27
Jan 3
Jan 10
Jan 1
Research Plan
Draft Logbook Submission
Assess Feedback from Draft
Intelligent Biometric Techniques in
ChA - Pattern Classification: Mathematical Background
Revise Statistics to prepare for Bayes Theorem
Revise Linear Algebra previous studies
Ch1 - Pattern Classification: Introduction
Ch1&2 - Neural Networks: Statistical Pattern Recognition, and Probability Density Estimatic
Research Feature Extraction
Create a report of findings on Feature Extraction
Break for Christmas/Catch-Up if needed
Research r
Create a re


Feb
Mar
Jan 24
Jan 31
Feb 7
Feb 14
Feb 21
Feb 28
Mar 7
Mar 14
Mar 21
Mar 28
Apr 4
Logbook
Final Logbook Submission
Fingerprint and Face Recognition
Research Applications of Image Recognition
Create a report of findings on Applications
Find industrial application and put this in to practice
Python & Julia learning/pracice
Practice applications using Computer Analysis
Create body of report from research
Introduction
Conclusion
Abstract
Put report together - e.g.
Draft Report submissio


Apr
May
Jun
pr 11
Apr 18
Apr 25
May 2
May 9
May 16
May 23
May 30
Jun 6
Jun 13
Jun 20
Jun
Final Report Submission
table of contents
n
Draft Report Feedback (rough date)
Assess feedback from report and work on this
Viva Week
Revise Content of Report
Address potential questions from Viva
2-3 Lay Audience Summary
10-12 Minute Presentation of Results

13.2 Sobel Filter Poster

13.3 Calculations

13.3.1 Proving equation 14

$z^L = w^L a^{L-1} + b^L \Rightarrow \frac{\partial z^L}{\partial w^L} = a^{L-1}$
$a^L = \sigma(z^L) \Rightarrow \frac{\partial a^L}{\partial z^L} = \sigma'(z^L)$
$C_0 = (a^L - y)^2 \Rightarrow \frac{\partial C_0}{\partial a^L} = 2(a^L - y)$

13.3.2 Example 7 calculations

$P(x) = \frac{(x-13)(x-15)(x-12)(x-10)(x-16)(x-22)}{(9-13)(9-15)(9-12)(9-10)(9-16)(9-22)}22+$
$\frac{(x-9)(x-15)(x-12)(x-10)(x-16)(x-22)}{(13-9)(13-15)(13-12)(13-10)(13-16)(13-22)}25+$
$\frac{(x-9)(x-13)(x-12)(x-10)(x-16)(x-22)}{(15-9)(15-13)(15-12)(15-10)(15-16)(15-22)}19+$
$\frac{(x-9)(x-13)(x-15)(x-10)(x-16)(x-22)}{(12-9)(12-13)(12-15)(12-10)(12-16)(12-22)}12+$
$\frac{(x-9)(x-13)(x-15)(x-12)(x-16)(x-22)}{(10-9)(10-13)(10-15)(10-12)(10-16)(10-22)}7+$
$\frac{(x-9)(x-13)(x-15)(x-12)(x-10)(x-22)}{(16-9)(16-13)(16-15)(16-12)(16-10)(16-22)}6.5+$
$\frac{(x-9)(x-13)(x-15)(x-12)(x-10)(x-16)}{(22-9)(22-13)(22-15)(22-12)(22-10)(22-16)}6.5$

$P(x) = \frac{(x-9)(x-15)(x-12)(x-10)(x-16)(x-22)}{-648}25+$
$\frac{(x-9)(x-13)(x-12)(x-10)(x-16)(x-22)}{1260}19+$
$\frac{(x-9)(x-13)(x-15)(x-10)(x-16)(x-22)}{720}12+$
$\frac{(x-9)(x-13)(x-15)(x-12)(x-16)(x-22)}{-2160}7+$
$\frac{(x-9)(x-13)(x-15)(x-12)(x-10)(x-22)}{-3024}6.5+$
$\frac{(x-9)(x-13)(x-15)(x-12)(x-10)(x-16)}{589680}6.5$

$P(x) = \frac{(x-9)(x-13)(x-15)(x-12)(x-10)(x-16)(x-22)}{6552(x-9)}22+$
$\frac{(x-9)(x-13)(x-15)(x-12)(x-10)(x-16)(x-22)}{-648(x-13)}25+$
$\frac{(x-9)(x-13)(x-15)(x-12)(x-10)(x-16)(x-22)}{1260(x-15)}19+$
$\frac{(x-9)(x-13)(x-15)(x-12)(x-10)(x-16)(x-22)}{720(x-12)}12+$
$\frac{(x-9)(x-13)(x-15)(x-12)(x-10)(x-16)(x-22)}{-2160(x-10)}7+$
$\frac{(x-9)(x-13)(x-15)(x-12)(x-10)(x-16)(x-22)}{-3024(x-16)}6.5+$
$\frac{(x-9)(x-13)(x-15)(x-12)(x-10)(x-16)(x-22)}{589680(x-22)}6.5$

$P(x) = (x-9)(x-13)(x-15)(x-15)(x-12)(x-10)(x-16)(x-22)(\frac{22}{6552(x-9)}+$
$\frac{25}{-648(x-13)} + \frac{19}{1260(x-15)} + \frac{(12}{720(x-12)} + \frac{7}{-2160(x-10)} + \frac{6.5}{-3024(x-16)} + \frac{6.5}{589680(x-22)})$

$P(x) = -0.00885565x^6 + 0.747309x^5 - 25.6916x^4 + 460.525x^3 - 4538.94x^2 + 2332.1x -$
48805.2

13.3.3 Deducing the Chebyshev polynomial, equation 28, section 9.2

$T_0(\cos\theta) = \cos(0) = 1 \implies T_0(x) = 1$ $T_1(\cos\theta) = \cos(\theta) \implies T_1(x) = x$ $T_2(\cos\theta) = \cos(2\theta) = 2\cos^2(\theta) - 1 \implies T_2(x) = 2x^2 - 1$ $T_3(\cos\theta) = \cos(3\theta) = 4\cos^3(\theta) - 3\cos(\theta) \implies T_3(x) = 4x^3 - 3x$

13.3.4 Deducing the Chebyshev Recurrence Relation, equation 29, section 9.2

$T_{n+1}(\cos\theta) = \cos((n+1)\theta)$ $\cos((n+1)\theta) = \cos n\theta\cos\theta - \sin n\theta\sin\theta \implies \cos(n+1)\theta + \cos(n-1)\theta = 2\cos\theta\cos n\theta \implies T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$

13.3.5 Full workings of equation 8

Example 9 $f(x) = 2x^4+3x^3-2x^2+x, x \in [-2,1]$ *Using the translation* $u = \frac{2x-a-b}{b-a} = \frac{2x-1}{3}$

We then have the following Chebyshev nodes (to 3.s.f): $u_1 = cos(\frac{1}{10}\pi) = 0.951$, $u_2 = cos(\frac{3}{10}\pi) = 0.588$, $u_3 = cos(\frac{5}{10}\pi) = 0$, $u_4 = cos(\frac{7}{10}\pi) = -0.588$, $u_5 = cos(\frac{9}{10}\pi) = -0.951$,

Using the relation between u and x, we may obtain the relevant x values, and then input this in to f(x) to obtain the corresponding values of y = f(x) : $y_1 = 43.5, y_2 = 12.8, y_3 = 0.500, y_4 = -0.797, y_5 = -5.56$

We then use this to calculate our values of c_n *from equation 30:* $c_0 =$ *average* $T_0y = 10.5$ $c_1 = 2*$ *average* $T_1y = 2*$ *average* $u * y = 21.1$ $c_2 = 2*$ *average* $T_2y = 2*$ *average* $(2u^2-1) * y = 11.5$ $c_3 = 2*$ *average* $T_3y = 2*$ *average* $(4u^3-3u) * y = 5.91$ $c_4 = 2*$ *average* $T_4y = 2*$ *average* $(8u^4-8u^2+1) * y = 1.07$

Using these values, we may approximate $f(x)$ *as follows:* $f(x) = 10.5T_0(u)+21.1T_1(u)+11.5t_2(u)+5.91T_3(u)+1.07T_4(u)$, *or* $f(x) = 8.56x^4+23.64x^3+14.44x^2+3.37x+0.07$

13.4 Code

13.4.1 Perceptron Network Code

```
//Code for simple 'AND'

#include <stdlib.h> //for our random numbers
#include <math.h> //for exp
#include <iostream>
using namespace std;

//a function for the sigmoid function
double sigmoid_function(double x) {
   return 1 / (1 + exp(-1 * x));
}

int main()
{

   //cout << w[0] << endl; //w[0] corresponds to A, w[1] corresponds to B,
       w[2] is for the bias
   //cout << w[1] << endl;
   //cout << w[2] << endl;

   //defining all variables here to make code clearer
   int expected_output;
   //weights for a&b, and the bias
   double w_a = rand() % 10;
   double w_b = rand() % 10;
   double bias = rand() % 10;

   //a_L = activation
   double z_L, a_L;
   double delta_w_a, delta_w_b, delta_b; //dC/dw_a, dC/dw_B, dC/db


   for (int i = 0; i < 50000; i++) {

      int A = rand() % 2; //Assign 0, or 1, to A&B
      int B = rand() % 2;

      if (A == 0 && B == 0) {
```

```
            expected_output = 0;
        }
        else {
            expected_output = 1;
        }

        cout << "A = " << A << endl;
        cout << "B = " << B << endl;

        cout << "w_a = " << w_a << endl;
        cout << "w_b = " << w_b << endl;
        cout << "b = " << bias << endl;

        z_L = w_a * A + w_b * B + bias;
        a_L = sigmoid_function(z_L);
        cout << "Expected output: " << expected_output << endl;
        cout << "----VS----" << endl;
        cout << "Network output: " << a_L << endl;

        //changes to be made
        delta_w_a = 2 * A * a_L * (1 - a_L) * (a_L - expected_output);
        delta_w_b = 2 * B * a_L * (1 - a_L) * (a_L - expected_output);
        delta_b = 2 * a_L * (1 - a_L) * (a_L - expected_output);

        w_a -= delta_w_a;
        w_b -= delta_w_b;
        bias -= delta_b;


    }
}
```

13.4.2 Multilayer Network Code

```
#include <iostream>
#include <algorithm> // for transform
#include <functional> // for plus
#include <cmath>
#include <math.h>
#include <Dense>

using namespace std;
using namespace Eigen;

//sigmoid function
double sig_fntion(double x) {
    return 1 / (1 + exp(-1 * x));
}

int main()
{
    //describing the NN
    int no_of_layers = 3; //including input, output, and hidden
    int neurons_input = 4; //number of input neurons, labelled A, B, C, D
    int neurons_output = 2; //outputs labelled w and b respectively
    int neurons_h1 = 3; //number in hidden layer, labelled 0 - 2
    int a = neurons_h1 + neurons_output; //the amount of activations from
        the activation function, input neuron values are not activations here

    double A, B, C, D; //our inputs

    int N_w, N_b;
    N_w = neurons_input * neurons_h1 + neurons_h1 * neurons_output;
        //number of weights
    N_b = neurons_h1 + neurons_output; //number of biases


    VectorXd w(N_w), b(N_b); //w_0A^L-1, w_0B^L-1, w_0C^L-1, w_0D^L-1,
        w_1A^L-1, etc.
     //vector displaying ALL weights
    for (int i = 0; i < N_w; i++) {
        w(i) = rand() % 10;
    }
```

```
//weights = (w_0A, w_0B, w_0C, w_0D, w_1A, w_1B, w_1C, w_1D, w_2A,
    w_2B, w_2C, w_2D, w_w0, w_w1, w_w2, w_b0, w_b1, w_b2)

//vector displaying ALL biases
for (int k = 0; k < N_b; k++) {
    b(k) = rand() % 10;
}

int out = 2; //no of output neurons
VectorXd e_out(out); //our expected outputs
// e = ['white' 'black'] - 2 neurons, one depicts white, the other
    depicts black

cout << "Beginning program. Intial values of weights: " << endl;
cout << w << endl;
cout << "Initial biases: " << endl;
cout << b << endl;



for (int i = 0; i < 100000; i++) {

    //our input neurons are A B C D
    A = rand() % 2;
    B = rand() % 2;
    C = rand() % 2;
    D = rand() % 2;

    double average_grayscale_value; //the inputs are grayscale values
    average_grayscale_value = (A + B + C + D) / 4;


    //declare what the outputs should be
    if (average_grayscale_value > 0.5) {
        e_out << 0, 1; //image mostly black
    }
    else {
        e_out << 1, 0; //image mostly white
    }
```

```
//define the activations
VectorXd activations(a);
VectorXd z(a); //to work out activations
for (int q = 0; q < a; q++) {
    activations(q) = 0;
    z(q) = 0;
}

//hidden layer:
for (int m = 0; m < neurons_h1; m++) {
    z(m) = w(4 * m) * A + w(4 * m + 1) * B + w(4 * m + 2) * C + w(4
        * m + 3) * D;
}
//OUTPUT LAYER
z(3) = w(12) * z(0) + w(13) * z(1) + w(14) * z(2);
z(4) = w(15) * z(0) + w(16) * z(1) + w(17) * z(2);


//therefore all activations:
for (int t = 0; t < a; t++) {
    activations(t) = sig_fntion(z(t));
}



//GRADIENT DESCENT
VectorXd deltaC_w(N_w), deltaC_b(N_b); //delta cost for C wrt w's,
    and then b's
 //assign zeros to the above vectors to allow for values to be added
for (int l = 0; l < N_w; l++) {
    deltaC_w(l) = 0;
}
for (int f = 0; f < N_b; f++) {
    deltaC_b(f) = 0;
}

//FOR THE OUTPUT LAYER
// i = all weights - (number of activations)

    //for the 'white' output neuron
for (int g = N_w - a; g < N_w - a + neurons_h1; g++) {
```

```
        deltaC_w(g) = 2 * w(g - 13) * activations(3) * (1 -
            activations(3)) * (activations(3) - e_out(0));
    }
    //a(3) = activation of the 'white' neuron

    //bias
    deltaC_b(3) = 2 * activations(3) * (1 - activations(3)) *
        (activations(3) - e_out(0));

    //for the 'black' output neuron
    for (int h = N_w - a + neurons_h1; h < N_w; h++) {
        deltaC_w(h) = 2 * w(h - 16) * activations(4) * (1 -
            activations(4)) * (activations(4) - e_out(1));
    }

    //bias
    deltaC_b(4) = 2 * activations(4) * (1 - activations(4)) *
        (activations(4) - e_out(1));

    //HIDDEN LAYER

    VectorXd sum(neurons_h1); //sum = change in C wrt to a^L-1
    sum(0) = w(12) * activations(3) * (1 - activations(3)) * 2 *
        (activations(3) - e_out(0)) + w(15) * activations(4) * (1 -
        activations(4)) * 2 * (activations(4) - e_out(1));
    sum(1) = w(13) * activations(3) * (1 - activations(3)) * 2 *
        (activations(3) - e_out(0)) + w(16) * activations(4) * (1 -
        activations(4)) * 2 * (activations(4) - e_out(1));
    sum(2) = w(14) * activations(3) * (1 - activations(3)) * 2 *
        (activations(3) - e_out(0)) + w(17) * activations(4) * (1 -
        activations(4)) * 2 * (activations(4) - e_out(1));

    //for weights connecting input and hidden
    for (int i = 0; i < (neurons_h1 * neurons_input); i++) {

        if (i < neurons_input) { //w_0A, w_0B, w_0C, w_0D
            int j = 0;
            if (i % neurons_input == 0) {
                deltaC_w(i) = A * activations(j) * (1 - activations(j)) *
                    sum(j);
            }
```

```
        else if (i % neurons_input == 1) {
            deltaC_w(i) = B * activations(j) * (1 - activations(j)) *
                sum(j);
        }
        else if (i % neurons_input == 2) {
            deltaC_w(i) = C * activations(j) * (1 - activations(j)) *
                sum(j);
        }
        else if (i % neurons_input == 3) {
            deltaC_w(i) = D * activations(j) * (1 - activations(j)) *
                sum(j);
        }
    }

    else if (i < 2 * neurons_input) { //w_1A, w_1B, w_1C, w_1D,
        int j = 1;
        if (i % neurons_input == 0) {
            deltaC_w(i) = A * activations(j) * (1 - activations(j)) *
                sum(j);
        }
        else if (i % neurons_input == 1) {
            deltaC_w(i) = B * activations(j) * (1 - activations(j)) *
                sum(j);
        }
        else if (i % neurons_input == 2) {
            deltaC_w(i) = C * activations(j) * (1 - activations(j)) *
                sum(j);
        }
        else if (i % neurons_input == 3) {
            deltaC_w(i) = D * activations(j) * (1 - activations(j)) *
                sum(j);
        }
    }

    else if (i < 3 * neurons_input) { //w_2A, w_2B, w_2C, w_2D,
        int j = 2;
        if (i % neurons_input == 0) {
            deltaC_w(i) = A * activations(j) * (1 - activations(j)) *
                sum(j);
        }
        else if (i % neurons_input == 1) {
```

```
            deltaC_w(i) = B * activations(j) * (1 - activations(j)) *
                sum(j);
        }
        else if (i % neurons_input == 2) {
            deltaC_w(i) = C * activations(j) * (1 - activations(j)) *
                sum(j);
        }
        else if (i % neurons_input == 3) {
            deltaC_w(i) = D * activations(j) * (1 - activations(j)) *
                sum(j);
        }
    }
}

for (int i = 0; i < neurons_h1; i++) {
    deltaC_b(i) = activations(i) * (1 - activations(i) * sum(i));
}


//IMPLEMENT CHANGES
for (int i = 0; i < N_w; i++) {
    w(i) -= deltaC_w(i);
}

for (int i = 0; i < N_b; i++) {
    b(i) -= deltaC_b(i);
}

double cost_weights = 0;
double cost_bias = 0;

for (int i = 0; i < N_w; i++) {
    cost_weights += deltaC_w(i);
}

for (int i = 0; i < N_b; i++) {
    cost_bias += deltaC_b(i);
}

double cost_total = cost_weights + cost_bias;
```

```
        cout << "Cost function = " << cost_bias << endl;
    }

    cout << "Learning complete, values of weights and biases: " << endl;
    cout << "w = " << endl;
    cout << w << endl;

    cout << "b = " << endl;
    cout << b << endl;


}
```

13.5 Sobel Filter Poster

How ultra is sound? Quantifying images to refine diagnosis

Adelaide Baron – 16630927

Introduction

Ultrasound scans have many uses in medicine as it allows us to see into the body. By placing a probe emitting high frequency sound waves against a part of the body, echoes received by the probe are used to create images like those in figure 1.

Ultrasound and ALS

Ultrasound can be used to diagnose certain muscle-related issues; such as ALS and Myositis[2], and to monitor diseases as they progress[3]. ALS, also known as Motor Neuron Disease, stands for Amyotrophic Lateral Sclerosis and is a nerve[4] disease, wherease Mysositis is muscle disease[5]. Both can be detected by ultrasound, as displayed in figure 1. Using various methods, such as statistics and image analysis, we can quantify the ultrasound image, and use this to refine our diagnosis. Here, we will look at three quantities.

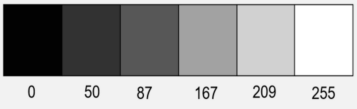


Figure 2
Image: (Color \ Processing.org, 2021)

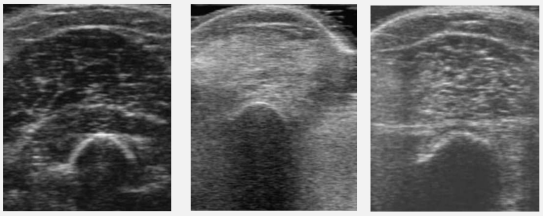


Figure 1
From left to right: healthy muscle, muscle affected by a muscle disease (Mysositis), and a muscle affected by ALS [2].

Quantifying ultrasound images:

1. **Whiteness**
The whiteness is the mean (grayscale, see figure 2) value of the pixels in the image (or representative part of). The higher the whiteness value, the more homogeneously white the image is[2].
2. **White-area index**
This is a measurement of how many truly white pixels are in the ultrasound image. It is normalised as otherwise the value would depend upon image size[2].
3. **Inhomogeneity index**
An inhomogeneous image is one with a lot of boundaries between light and darker pixels. One way to make this clearer is using a sobel filter (read 'Sobel Filter' to the right). The *inhomogeneity index* is a measurement of the boundaries between dark and light pixels, the higher the index, the more boundaries there are[2].

Sobel Filter

It is an edge detection filter. By reviewing the gradient between adjacent pixels, we assign pixels a number from the filter dependant upon the gradient. The pixels are then replaced by their number and a new image is created – figure 3.
The number of white pixels, which is then normalised to account for image size, is the *inhomogeneity index*

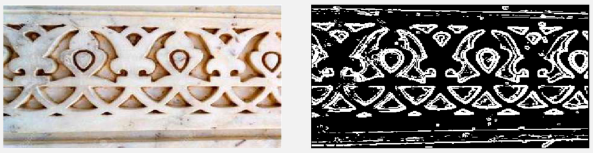


Figure 3
Image before and after the Sobel filter
Image: [1] GeeksforGeeks

Using the values

It has been found, through descriptive statistics[3], that the best value to distinguish nerve diseases is the inhomogeneity, whereas to distinguish a muscle disease it is best to use the value of whiteness, and the white-area index.

References

1.^ GeeksforGeeks. 2020. *MATLAB - Image Edge Detection using Sobel Operator from Scratch.* https://www.geeksforgeeks.org/matlab-image-edge-detection-using-sobel-operator-from-scratch/ [Accessed 5 April 2021].
2.^ Mathematics in Medicine, 2021 Panapto, Guest lecture for Industrial And Financial Mathematics - MTH2006M-2021, uploaded by Natasha Maurits, UMCG, 23 February 2021.
3.^ Motor neurone disease, 2021 https://www.nhs.uk/conditions/motor-neurone-disease/ [Accessed 11 April 2021].
4.^ NHS Choices, 2019. Motor neurone disease. [online] NHS. Available at: https://www.nhs.uk/conditions/motor-neurone-disease/ [Accessed 4 May 2021].
5.^ Myositis (polymyositis and dermatomyositis) - NHS, 2021. https://www.nhs.uk/conditions/myositis/ [Accessed 11 April 2021].

References

[1] M.A. Nielsen, *Neural Networks and Deep Learning*, Determination Press, 2015. [Ebook] Available: http://neuralnetworksanddeeplearning.com/chap1.html

[2] A. Baron, "How Ultra is Sound? Quantifying Images to Refine Diagnosis", Apr. 2021.

[3] S. Marsland, *Machine Learning: An Algorithmic Perspective*, 2nd ed., NY USA, Chapman and Hall, 2014.

[4] D. Stathakis, (2009). "How many hidden layers and nodes?". Internal Journal of Keynote Sensing, 30:8, 2133-2147, DOI: 10.1090/01431160902549278

[5] B. W. Char, S. M. Watt, "Representing and Characterizing Handwritten Mathematical Symbols through Succicnt Functional Approximation," in *Ninth International Conference on Document Analysis and Recognition (ICDAR)* , 2007, pp. 1198-1202.

[6] Y. Chee, K. Franke, M. Fourmentin, S. Madhvanath, J. Magaña, G. Pakos, G. Russell, M. Selvaraj, G. Seni, C. Tremblay, L. Yaeger, "Ink Markup Language (InkML)", W3C, Sep. 20, 2011. [Online] Available: https://www.w3.org/TR/InkML/

[7] L. Den, "The MNIST database of handwritten digit images for machine learning research." *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp 141 - 142, 2012.

[8] A. Bain, *Mind and Body: The Theories of Their Relation.* NY U.S.A.: D. Appleton and Company, 1873.

[9] W. James. *The Principles of Psychology*, NY U.S.A: H. Holt and Company, 1890.

[10] W. S. McCulloch, W. Pitts, "A Logical Calculus of Ideas Immanent in Nervous Activity", *Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, Dec. 1943.

[11] C. M. Bishop, *Neural networks for pattern recognition*, Oxford, UK: Oxford University Press, 1995.

[12] [FirstNNbegin] A. Arnx. (2019, 01.13). First neural network for beginners explained (with code), [Online] Available: https://towardsdatascience.com/first-neural-network-for-beginners-explained-with-code-4cfd37e06eaf

[13] J. Heaton, *Introduction to Neural Networks for Java*, 2nd ed., Chesterfield, MO: Heaton Research Inc, 2008.

[14] H. Samson, "Getting to know Activation Functions in Neural Networks," Towards Data Science, Jun 25, 2020. [Online] Available: https://towardsdatascience.com/getting-to-know-activation-functions-in-neural-networks-125405b67428: :text=Binary%20step%20function%20is%20a,said%20threshold%20neuron%20is%20deactivated.

[15] BP. Baheti, "12 Types of Neural Network Activation Functions: How to Choose," *Deep Learning*, Accessed: May 1, 2022. [Online] Available: https://www.v7labs.com/blog/neural-networks-activation-functionschoose-activation-function

[16] D. Yarotsky, "Error bounds for approximations with deep ReLU networks," *Neural Networks*, vol. 94, pp 103-114, Oct. 2017.

[17] N. Ketkar, "Stochastic Gradient Descent" in: *Deep Learning with Python*, Berkeley, CA, Apress, 2017.

[18] J. Brownlee, " Weight Initialization for Deep Learning Neural Networks," *Machine Learning Mastery,* Feb. 3, 2021. [Online] Available: https://machinelearningmastery.com/weight-initialization-for-deep-learning-neural-networks/: :text=Historically%2C%20weight%20initialization%20follows%20simple,the%20range%20%5B%2D1%2C%201%5D

[19] Aravindpai, "CNN vs. RNN vs. ANN – Analyzing 3 Types of Neural Networks in Deep Learning", *Analytics Vidhya*, Feb. 17, 2020. [Online Available: https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning/

[20] W. Yin, K. Kann, M Yu, H. Schutze, *Comparative Study of CNN and RNN for Natural Language Processing*, LMU Munich, Germany, and IBM Research, U.S.A, Feb. 7, 2017.

[21] A. Poznanski, L. Wolf, "CNN-N-Gram for Handwriting Word Recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, U.S.A, pp. 2305-2314, 2016. [Online] Available: https://openaccess.thecvf.com/content_cvpr_2016/html/Poznanski_CNN-N-Gram_for_Handwriting_CVPR$_2$016_*paper.html*

[22] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, A. A. Bharath, "Generative Adversarial Networks: An Overview," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp 53 - 65, Jan. 2018

[23] W. Badr, Auto-Encoder: "What Is It? And What Is It Used For? (Part 1)," *Towards Data Science*, Apr. 22, 2019. [Online] Available: https://towardsdatascience.com/auto-encoder-what-is-it-and-what-is-it-used-for-part-1-3e5c6f017726

[24] S. Kullback, R.A. Leibler, "On information and sufficiency," *Annals of Mathematical Statistics*, vol. 22, no. 1, pp 79-86, 22 (1): 79–86, Mar. 1951.

[25] T. Yang, Y. H. Qiu, "Improvement and implementation for Canny edge detection algorithm" in *Seventh International Conference on Digital Image Processing (ICDIP)*, Jul. 6, 2015.

[26] I. Sobel, *An Isotropic 3x3 Image Gradient Operator*, Presentation at Stanford A.I. Project, 1968. [Online] Available: https://www.researchgate.net/publication/239398674_An_Isotropic_3x3_Image_Gradient_Operator

[27] D. Ziou, S. Tabbone, "Edge detection techniques: An overview," *International Journal of Patttern Recognition and Image Analysis*, vol. 4, pp 537-559, 1998.

[28] S. Sodha, "An Implementation of Sobel Edge Detection," *Project Rhea*, 2017. [Online] Available: https://www.projectrhea.org/rhea/index.php/An_Implementation_of_Sobel_Edge_Detection

[29] Online Edge Detection, *Pine Tools*. [Online], Available: https://pinetools.com/image-edge-detection

[30] Q. Kong, T. Siauw, A. Bayen, *Python Programming and Numerical Methods*, 1st ed., Berkley CA: Academic Press, 2020.

[31] "Chebyshev Polynomials - Definition and Properties," *Brilliant*, Accessed Feb. 20, 2022, [Online] Available: https://brilliant.org/wiki/chebyshev-polynomials-definition-and-properties/: :text=The%20Chebyshev%20polynomials%20are%20a,solving%20polynomials%20and%20approximating%20functions.

[32] J. P. Berrut, L. N. Trefethen, "Barycentric Lagrange Interpolation," *SIAM Review*, vol. 46, no. 3, pp 501-517, 2004. [Online] Available: https://people.maths.ox.ac.uk/trefethen/barycentric.pdf

[33] W. H. Press, *Numerical Recipes: The Art Of Scientific Computing*, 3rd ed., Cambridge, UK: Cambridge University Press, 2007.

[34] Wolfram Language & System Documentation Center, Get Coordinates from an Image, accessed 20 April 2022. [Online] Available: https://reference.wolfram.com/language/workflow/GetCoordinatesFromAnImage.html