



Candidates must complete this page and then give this cover and their final version of the extended essay to their supervisor.

Candidate session number

Candidate name

School name

Examination session (May or November)

May

Year

2015

Diploma Programme subject in which this extended essay is registered: Computer Science

(For an extended essay in the area of languages, state the language and whether it is group 1 or group 2.)

Title of the extended essay: Neural Network vs. Vector Based Algorithm

A comparison of speed and accuracy when identifying numbers

Candidate's declaration

This declaration must be signed by the candidate; otherwise a mark of zero will be issued.

The extended essay I am submitting is my own work (apart from guidance allowed by the International Baccalaureate).

I have acknowledged each use of the words, graphics or ideas of another person, whether written, oral or visual.

I am aware that the word limit for all extended essays is 4000 words and that examiners are not required to read beyond this limit.

This is the final version of my extended essay.

Candidate's signature: _____

Date: 2015-03-04

Supervisor's report and declaration

The supervisor must complete this report, sign the declaration and then give the final version of the extended essay, with this cover attached, to the Diploma Programme coordinator.

Name of supervisor (CAPITAL letters) _____

Please comment, as appropriate, on the candidate's performance, the context in which the candidate undertook the research for the extended essay, any difficulties encountered and how these were overcome (see page 13 of the extended essay guide). The concluding interview (viva voce) may provide useful information. These comments can help the examiner award a level for criterion K (holistic judgment). Do not comment on any adverse personal circumstances that may have affected the candidate. If the amount of time spent with the candidate was zero, you must explain this, in particular how it was then possible to authenticate the essay as the candidate's own work. You may attach an additional sheet if there is insufficient space here.

The candidate conducted substantial research into the relevant background concepts and was surprised to discover how far back the literature extends.

The candidate has a particular interest in the functioning of the human brain, so he embraced the intellectual challenge of coding a neural network even though he decided not to include that work in his essay. (Note: while not clearly indicated in the text, the candidate made very clear in the *viva voce* that the neural-network algorithm used in his investigation was _____ s and not his.)

I was impressed by the zeal with which the candidate worked and gratified to discover how much he had learnt.

This Extended Essay has
been checked for originality
on turnitin.com

This declaration must be signed by the supervisor; otherwise a mark of zero will be awarded.

I have read the final version of the extended essay that will be submitted to the examiner.

To the best of my knowledge, the extended essay is the authentic work of the candidate.

As per the section entitled "Responsibilities of the Supervisor" in the EE guide, the recommended number of hours spent with candidates is between 3 and 5 hours. Schools will be contacted when the number of hours is left blank, or where 0 hours are stated and there lacks an explanation. Schools will also be contacted in the event that number of hours spent is significantly excessive compared to the recommendation.

I spent hours with the candidate discussing the progress of the extended essay.

Supervisor's signature: _____

Date: 2015-03-04

Assessment form (for examiner use only)

Candidate session number	
--------------------------	--

Achievement level

Criteria	Examiner 1 maximum	Examiner 2 maximum	Examiner 3
A research question	2	<input style="width: 30px; height: 20px;" type="text" value="2"/>	<input style="width: 30px; height: 20px;" type="text"/>
B introduction	2	<input style="width: 30px; height: 20px;" type="text" value="1"/>	<input style="width: 30px; height: 20px;" type="text"/>
C investigation	4	<input style="width: 30px; height: 20px;" type="text" value="3"/>	<input style="width: 30px; height: 20px;" type="text"/>
D knowledge and understanding	4	<input style="width: 30px; height: 20px;" type="text" value="4"/>	<input style="width: 30px; height: 20px;" type="text"/>
E reasoned argument	4	<input style="width: 30px; height: 20px;" type="text" value="3"/>	<input style="width: 30px; height: 20px;" type="text"/>
F analysis and evaluation	4	<input style="width: 30px; height: 20px;" type="text" value="3"/>	<input style="width: 30px; height: 20px;" type="text"/>
G use of subject language	4	<input style="width: 30px; height: 20px;" type="text" value="3"/>	<input style="width: 30px; height: 20px;" type="text"/>
H conclusion	2	<input style="width: 30px; height: 20px;" type="text" value="2"/>	<input style="width: 30px; height: 20px;" type="text"/>
I formal presentation	4	<input style="width: 30px; height: 20px;" type="text" value="2"/>	<input style="width: 30px; height: 20px;" type="text"/>
J abstract	2	<input style="width: 30px; height: 20px;" type="text" value="0"/>	<input style="width: 30px; height: 20px;" type="text"/>
K holistic judgment	4	<input style="width: 30px; height: 20px;" type="text" value="3"/>	<input style="width: 30px; height: 20px;" type="text"/>
Total out of 36	<input style="width: 60px; height: 25px;" type="text"/>	<input style="width: 60px; height: 25px;" type="text" value="26"/>	<input style="width: 60px; height: 25px;" type="text"/>

Name of examiner 1: _____ Examiner number: _____
(CAPITAL letters)

Name of examiner 2: _____ Examiner number: _____
(CAPITAL letters)

Name of examiner 3: _____ Examiner number: _____
(CAPITAL letters)

IB Assessment Centre use only: B: _____

IB Assessment Centre use only: A: _____

Extended Essay in Computer
Science Group 5

Neural Network vs. Vector Based Algorithm

*A comparison of speed and accuracy when
identifying numbers*

Session;
May 2015

Word Count;
3706

Abstract

There are many different types of algorithms that can recognize hand writing, two distinct ones that are investigated in this paper are a method that uses the shape of the written digit (the stroke of the pen if you will) to figure out what the digit is, and the other emulates the brain to try and detect patterns a use those to recognize a written digit. The speed and accuracy of the algorithms is analysed and the stroke algorithm is determined to be faster while lacking the accuracy of the neural network which makes it ideal for recognition in a controlled environment while a Neural Network is better when the condition and quality of input cannot be controlled or counted on.

Minimal methodology ✓
 $J = 0$

(Word Count: 121)

Table of Contents

Introduction	1.0
Uses of Character Recognition	1.1
Neural Networks	1.2.0
Algorithm.....	1.2.1
Training the Network	1.2.2
Stroke Based Recognition.....	1.3.0
Algorithm.....	1.3.1
Training.....	1.3.2
Results	2.0
Theoretical Results	2.1
Testing Environment	2.2
Actual Results	2.3
Without Distortion	2.3.1
With Distortion.....	2.3.2
Conclusion	3.0
Experimental Problems and Future Improvements.....	3.2
Works Cited	4.0
Appendices	5.0

Introduction is too long B=1

1.0 Introduction

In the past few decades computers have been getting more and more powerful as a result they have been able to perform many tasks that humans are not very good at such as doing complex math problems or being very exact for extended periods of time. This makes them perfect for a huge range of tasks from word processing to running nuclear power plants. But computers until recently have never been able to come close to humans when it comes to any sort of pattern recognition. Advancements in computer processing power have let computer scientists develop algorithms that can emulate the human ability to recognize complex patterns this field of artificial intelligence is known as computer vision. In this paper I will be looking at two different types of handwriting recognition algorithms; the Neural Network and the image analysis approach.

1.1 Uses of Handwriting Recognition

Handwriting recognition or OCR (optical character recognition) is a branch of computer vision research that has countless applications in many areas in our everyday lives, for example for students taking notes it would be very convenient if you went to jot something down and all you had to do was take a picture of it and it would be saved as a text file. Or with the tablets with the included stylus which have become more popular recently it could be very convenient if you wrote a note with the stylus and it was immediately converted into text. There are even companies such as myScript who have recognition algorithms for translating sheet music from paper to the computer screen. Despite advancements in monitors, keyboards, and mice a pen is still the most effective and general way to get your ideas across so it seems like a necessary step to have the utility of the pen being able to interface with the computer.

1.2.0 Neural Networks

The first algorithm that I am looking at is the more complex of the two. The basic type of algorithm is called a “neural network” and as the name implies the neural network is largely based on the structure of the brain. The actual theory behind the networks is fairly basic, but the easiest way to understand them is to understand the brain. The brain in a very basic sense is many cells called neurons that are connected together with another type of cell called dendrites. (Structure of the Neuron) On their own these Neurons can't do much but when working together they can accomplish a great deal, Neural Networks operate on the same principal of a great many simple components working together towards something more.

While neural network algorithms try to emulate their biological counterparts the neurons in our head are very effective at many things that silicone (artificial) neurons are not very good at. Most notably handling imperfect data meaning data that is has unexpected features or errors.

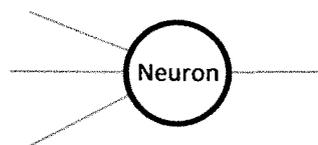


Figure 1: An Artificial Neuron

At its core much like the brain the neural network is built with neurons which alone are very simple machines that can't do much but when put together have the ability to compute problems that are very complex for many computers today with conventional algorithms. The artificial neurons have a very similar structure to the biological neurons in the brain with many different inputs attaching to the neuron and one singular output leaving the neuron. (Colin)

1.2.1 The Algorithm

For the investigation I am using open source neural network software developed by Mike O'Neil, it is a five-layer convolutional neural network (a neural network designed for handwriting recognition). This basically means that the network gradually breaks this image down into smaller and smaller pieces. Which is a concept that doesn't make sense until the basic neural network algorithm is explained.

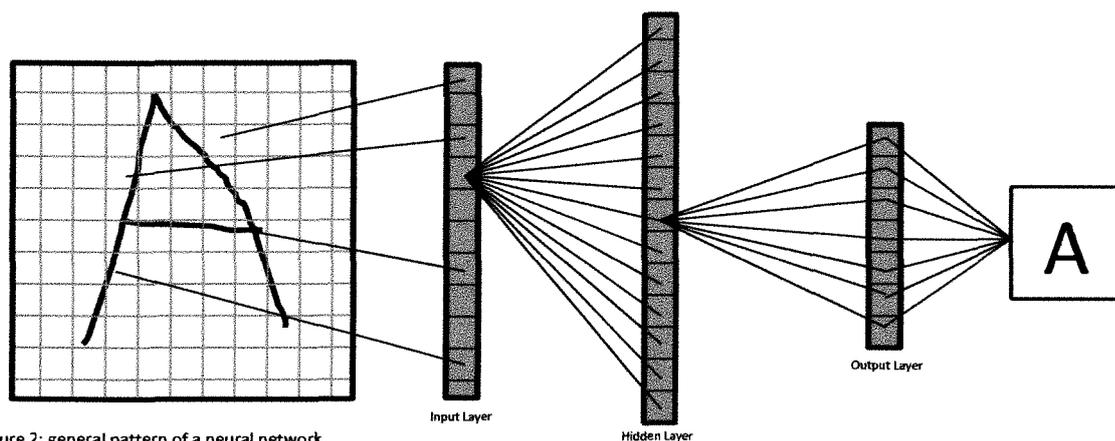


Figure 2: general pattern of a neural network

In general neural networks look like the image above, the neurons are organized into layers, the first layer known as the input layer consists of a series of neurons that accept a value based on the greyscale value of a pixel (generally a value from -1 to 1). If this value passes a certain threshold then the neuron outputs a signal. This signal will generally go to all of the neurons in the hidden layer this means that each neuron will be taking the entire image into account when calculating whether or not to output a signal. This is the stage at which the power of the neural network can be seen, the inputs of the neural network each have a weight associated with them, meaning that a number that augments the normal input received by the neuron by a number that is set by the programmer. Much like the input layer the neurons in the hidden layer will also output a signal to the output layer if the weighted inputs from the input layer are above a certain threshold. Finally the Output layer accepts inputs from the hidden layer and then each one will vote for a final output, meaning that once again weighted inputs from each of the neurons in the hidden layer are taken into account and when all compounded they are above a certain value again set by the programmer the neuron will vote for a certain final output. The compounding of the weights plus the inputs can be written as the following mathematical formula. (Colin)

$$y_n^i = \sum_{l=0}^{C_{n-1}} w_n^{il} \cdot x_{n-1}^l$$

Where, y_n^i is the compounded input,

w_n^{il} is the weight of input l ,

x_{n-1}^l is the raw input at input l ,

And C_{n-1} is the number of neurons on the previous layer

The outline is fairly constant throughout most designs of neural networks but the specifics of there are specific differences between each network. More specifically in the convolutional network that I examined for my investigation has the main difference of being that they take into account that letters and number won't always be written the same way each time, meaning that there will always be some non-uniform shifting and rotation of the inputs that can confuse a standard Neural Network (figure 2). What the convolutional network does is it takes the inputs and it maps them as features (end points, straight edges, curves) of a larger shape. (Haffner)

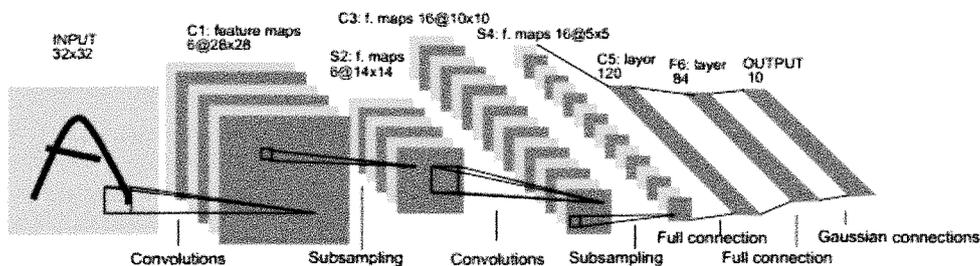


Figure 3: the design of a typical convolutional Neural Network (Haffner)

The design of a convolutional Neural Network (figure 3) shows a few of the tricks that the convolutional neural network uses to find out the features of the given input. The first is the convolution; this is basically a way of separating the image into feature maps without losing any data. Notice how there are two colours of feature maps in figure 3 this is another trick that allows the image to be shifted, the feature maps each contain part of the image with some overlap the but some of the feature maps (the ones that are the same colour in the diagram) have the same weights associated with their inputs. Meaning that they are looking for the same features in different places which as I mentioned before means that the letter can be shifted but still be recognized. After a feature has been detected by the feature map for example a curving endpoint on the upper right hand side of at least one of the maps its precise location doesn't matter too much and it shouldn't because looking for precise locations is what we are trying to avoid with the convolutional neural network. The easiest way to ignore the location of the features is to reduce the resolution of the image in the first set of feature maps, there are many ways to do this but the way that was used in the algorithm for my investigation was to simply average

no
colours
o.p.

problems with the language
e.p.

every 2 outputs to and as a result reduce each feature map down to one with the same information just less precise. This process is repeated until the feature map is too small to continue or the programmer decides that they are going to stop. In this specific case it was decided that after the feature map was reduced to a 5x5 area that the architecture of the Neural Network would shift to that of a Feed Forward Neural Network (figure 2). It is also important to note that during the process of subsampling and the convolutions the same ideas applied in terms of computing outputs from inputs as was used in figure 1 but the architecture of the network is different. (Heffner)

11.7
c-p

1.2.2 Training the Network

I stated earlier that the output threshold values and the input weights are set by the programmer and while it is possible that a programmer sets these values, it is more likely that they are set by a training algorithm. In general the training algorithm that is used is known as the back propagation algorithm, the basic idea behind it is relatively simple, if you know by how much the Neural Network is off the desired output you can fix it.

The basic algorithm works as follows starting with the output layer and working backwards you calculate by how much the current layer is off, increment closer to your desired value and then taking into account the change you just made calculate by how much the next output is off of your target. You repeat this basic procedure until you reach the input layer. After which you start again until any error that existed is gone (the input gives the right output). (LeCun)

The algorithm that I used splits this task into three different parts;

The first is

```
// calculate equation (3): dErr_wrt_dYn = F'(Yn) * dErr_wrt_Xn
for ( ii=0; ii<m_Neurons.size(); ++ii )
{
    output = m_Neurons[ ii ]->output;

    dErr_wrt_dYn[ ii ] = DSIGMOID( output ) * dErr_wrt_dXn[ ii ];
}

// calculate equation (4): dErr_wrt_Wn = Xnml * dErr_wrt_Yn
// For each neuron in this layer, go through
// the list of connections from the prior layer, and
// update the differential for the corresponding weight

ii = 0;
for ( nit=m_Neurons.begin(); nit<m_Neurons.end(); nit++ )
{
    NNNeuron& n = *(*nit); // for simplifying the terminology

    for ( cit=n.m_Connections.begin(); cit<n.m_Connections.end(); cit++ )
    {
        kk = (*cit).NeuronIndex;
        if ( kk == ULONG_MAX )
        {
            output = 1.0; // this is the bias weight
        }
        else
        {
            output = m_pPrevLayer->m_Neurons[ kk ]->output;
        }

        dErr_wrt_dWn[ (*cit).WeightIndex ] += dErr_wrt_dYn[ ii ] * output;
    }

    ii++;
}
}
```

H -
c.p.

In this part you build a map of the completed neural network and decide by how much the weight on each input will be shifted in the current layer.

In the second part the amount that the weights of the next layer have to be shifted by is calculated.

```
// calculate equation (5): dErr_wrt_Xnm1 = Wn * dErr_wrt_dYn,
// which is needed as the input value of
// dErr_wrt_Xn for backpropagation of the next (i.e., previous) layer
// For each neuron in this layer

ii = 0;
for ( nit=m_Neurons.begin(); nit<m_Neurons.end(); nit++ )
{
    NNNeuron& n =>(*nit); // for simplifying the terminology

    for ( cit=n.m_Connections.begin();
          cit<n.m_Connections.end(); cit++ )
    {
        kk=(*cit).NeuronIndex;
        if ( kk != ULONG_MAX )
        {
            // we exclude ULONG_MAX, which signifies
            // the phantom bias neuron with
            // constant output of "1",
            // since we cannot train the bias neuron

            nIndex = kk;

            dErr_wrt_dXnm1[ nIndex ] += dErr_wrt_dYn[ ii ] *
                m_Weights[ (*cit).WeightIndex ]->value;
        }
    }

    ii++; // ii tracks the neuron iterator
}
}
```

Finally in the third part the weights on the inputs of the current layer are changed based on what was calculated in part 1.

```
// calculate equation (6): update the weights
// in this layer using dErr_wrt_dW (from
// equation (4) and the learning rate eta

for ( jj=0; jj<m_Weights.size(); ++jj )
{
    oldValue = m_Weights[ jj ]->value;
    newValue = oldValue.dd - etaLearningRate * dErr_wrt_dWn[ jj ];
    m_Weights[ jj ]->value = newValue;
}
}
```

1.3.0 Stroke Based Recognition

The second algorithm that is going to be evaluated is stroke based recognition this algorithm is much easier to understand than Neural Networks, the algorithm itself tries to emulate what is known as "online" recognition or the process of translating characters as they are being written. But as it is an offline algorithm like the Neural Network there needs to be a fair amount of pre-processing before the character can be recognized.

gives the code but does not analyse 7
any specific parts of it F-
o.A

1.3.1 The Algorithm

The second algorithm is one that I created for the purpose of the investigation, it has two main components. The first is pre-processing where the raw image file is turned into a series of vectors that can be evaluated and the second is the actual image recognition phase in which the processed vectors are compared to templates and the one that most closely matches the image is outputted as the answer.

* } The pre-processing phase takes the longest out of both phases; the algorithm takes the image and finds all of the straight lines contained within it¹ after it has found all of the lines the algorithm creates vectors representing the digit and stores the created vectors as the shape. The templates are all based around 8 vectors because that is the most number of vectors that you need to accurately represent any number or letter in the English language. Most of the time when the image is processed there will be around 500 different straight line vectors due to the curves in most inputs.

This means that the vectors have to be run through an algorithm that simplifies them. First each vector is run through a pruning algorithm that removes any parallel vectors that are too close together, this appears in circumstances when there is a hole in a line or there was an error when the program was creating the vector representation of the image. On top of removing random error that could later affect results the pruning algorithm also makes the simplification that we will later do easier as any extra vectors will be quickly removed and I won't have to watch out for them later.

The next part of the simplification process is the stage where vectors are placed together, when there are 500 vectors most of them will only be a few pixels big and as a result they need to be placed with their neighbours. The easiest way to do this is to find 2 neighbours that have a very similar direction and are also very short and add them together. This process is repeated until there are 8 vectors left over.

```
while(shape.size() > 8)
{
    for(int i = 0; i < shape.size(); i++)
    {
        for(int x = i+1; x < shape.size()-1; x++)
        {
            //if the vector x is immediately after vector i
            if(x != i && (shape.get(i).xEnd == shape.get(x).xStart+1
                || shape.get(i).xEnd == shape.get(x).xStart-1))
            {
                if(shape.get(i).yEnd == shape.get(x).yStart+1
                    || shape.get(i).yEnd == shape.get(x).yStart-1)
                {
                    if(shape.get(i).getLength() <= 2)
                    {
                        //and if the direction that the vectors are facing are similar
                        if(shape.get(i).getDirection() == shape.get(x).getDirection()+1
                            || shape.get(i).getDirection() == shape.get(x).getDirection()-1)
                        {
                            //add the vectors together
                            shape.get(x).xStart = shape.get(i).xStart;
                            shape.get(x).yStart = shape.get(i).yStart;
                            shape.remove(i);
                        }
                    }
                }
            }
        }
    }
}
```

Now that the pre-processing phase is over the algorithm simply compares templates for each possible output to the processed image and if it finds a match it will use that as its output.

1.3.2 Training

The idea behind the training algorithm for this software is actually very similar to the back propagation algorithm used with Neural Networks. The same idea that if I know what my output should be then I can figure out my inputs also applies except instead of a general purpose series of equations the training algorithm for the stroke based system is a little more directed. First the algorithm creates the simple vector representation of a shape, then it uses that 8 vector representation to create a template, it then tests this template against more of the same shape measuring the rate of success and failure based on these successes and failures the programmer has to go into the code and change the biases of the program to accommodate the change. These biases can be anything from how close the directions of the vectors need to be to each other to be considered close enough to simplify to the preferred length of the simplified shape's vectors.

2.0 Results

2.1 Theoretical Results

When coming up with what algorithms I was going to be testing I decided that I should compare two algorithms that were as different as possible. So as to see if one was in general very good or if it was just good under specific circumstances or in a specific environment.

As a result both algorithms have their advantages and disadvantages, for example the stroke based recognition algorithm uses a training method which result in much less time required to learn new digits or shapes if there is some degree of difference from shapes that are already known while the neural network requires about the same amount of time to learn any new shape but the time it takes to learn the shape is anywhere from 650-750 seconds depending on the starting weights of the neurons and how much they need to change. In addition the neural network is designed for parallel processing and requires it to truly run efficiently while the stroke based algorithm is designed to run on current silicone processors which are much better at doing linear math problems. The stroke based recognition algorithm was also designed to be as lightweight as possible in order to again offer something different from the resource heavy Neural Network. But the fact that it is so light weight means that the stoke based algorithm does not have a feature detector which means that when shown shapes whose parts are too bent out of shape or the shapes are drawn slightly wrong, the algorithm proportion of wrong output can be expected to go up.

I think that as a result of the lack of a feature detector but the more efficient use of resources as a result the stroke based recognition algorithm will be faster and more efficient for the shapes that are of a

“higher quality” but when shapes that are not as ideal come up I think that the Neural Network will be more effective.

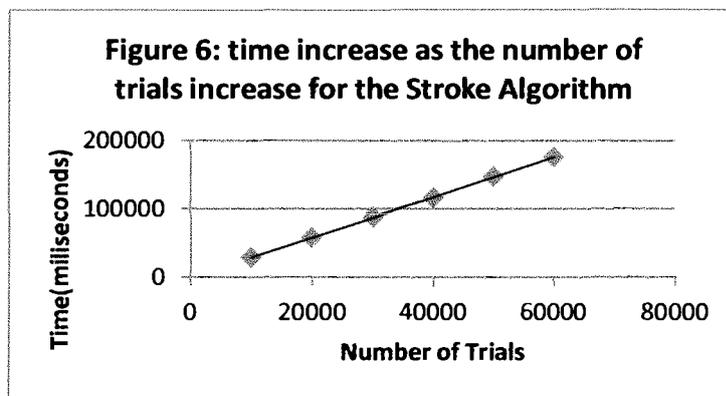
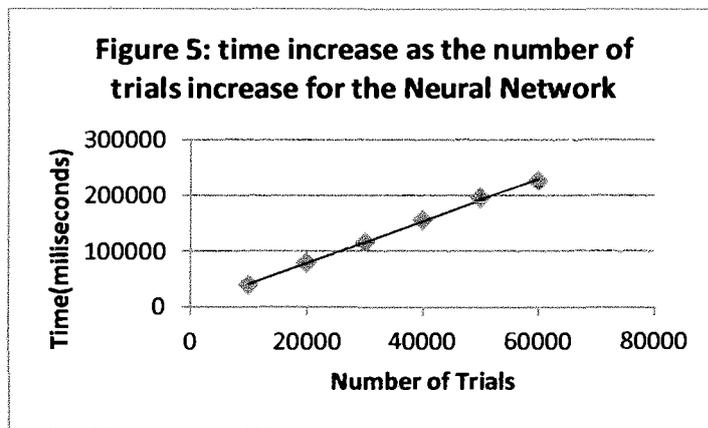
2.2 Environment

For the experiment I will close all non-essential processes on my computer and give the running algorithm priority I will also limit each algorithm to one core of my processor to give each one equal footing as I cannot guarantee the ability of both algorithms to run efficiently over multiple cores.

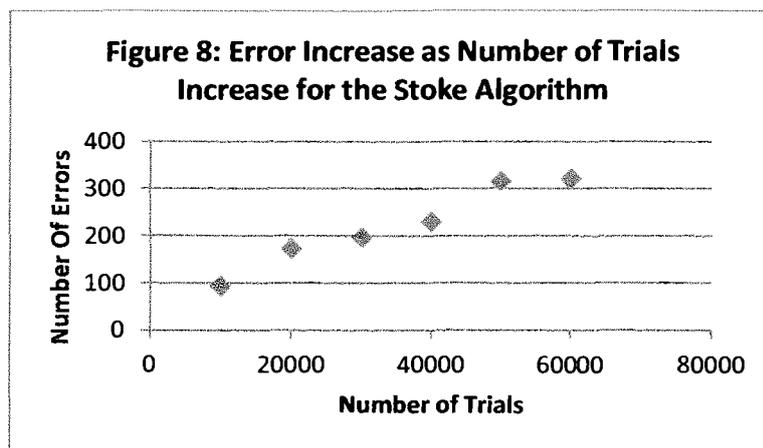
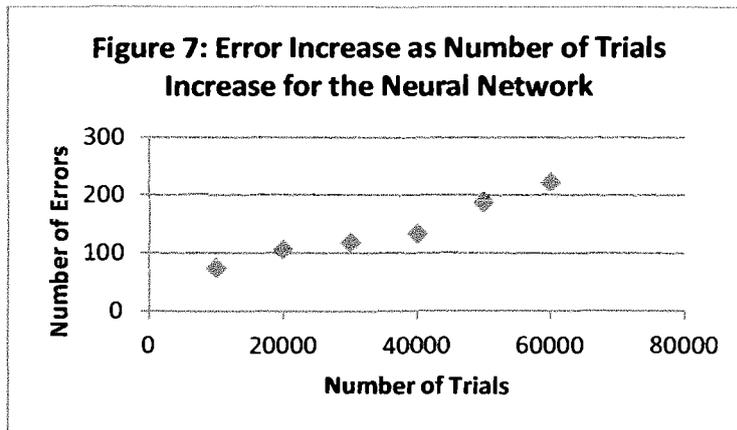
There will be 2 different tests for each algorithm under 2 different conditions. The first condition is without distortion, meaning the digits inputted by the computer will all be relatively straight and not be too hard to read. The second condition is with distortion, meaning that the digits will all be twisted or distorted in some way to make them harder to read. The first test will be a look at how both algorithms scale, meaning how the time it takes for them to process data as the amount of data increases. The second test will be by how much will the number of errors increase as the amount of data that is being processed increases.

2.3 The Actual Results

2.3.1 Without Distortion

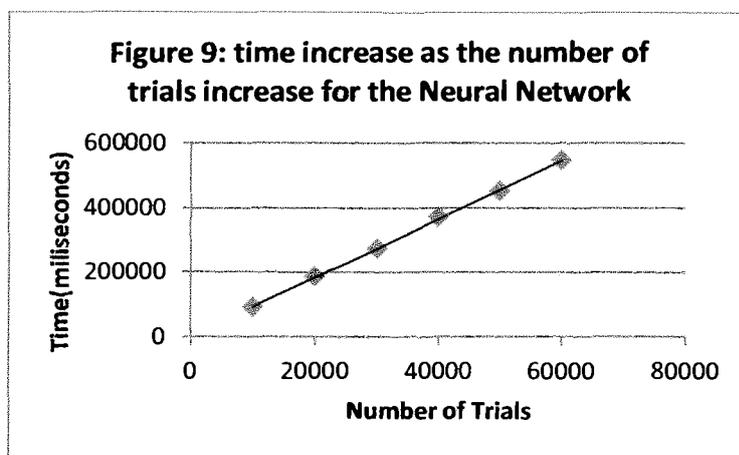


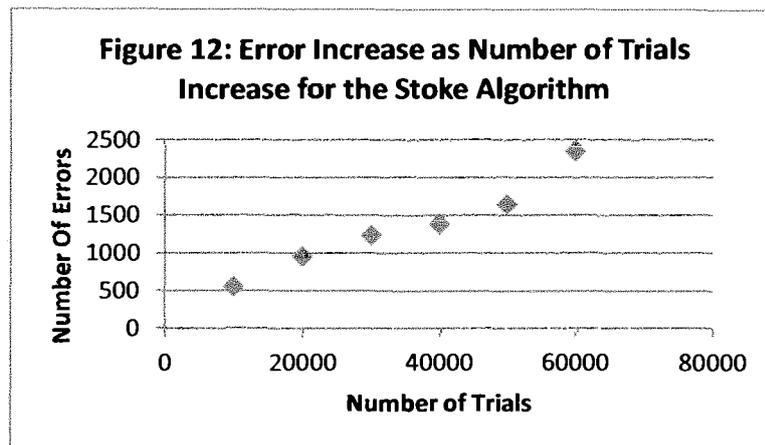
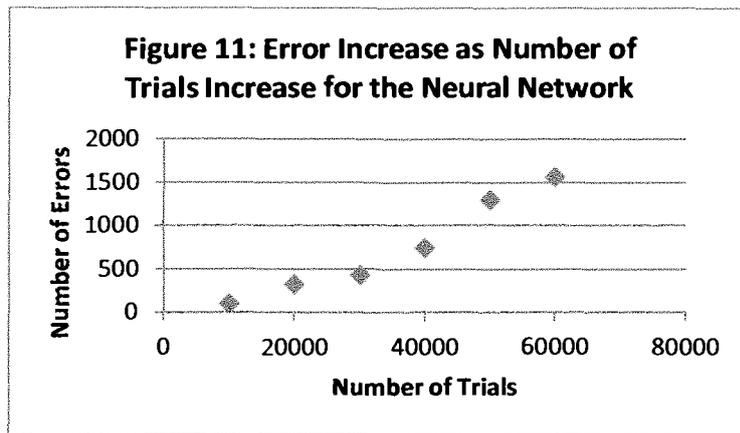
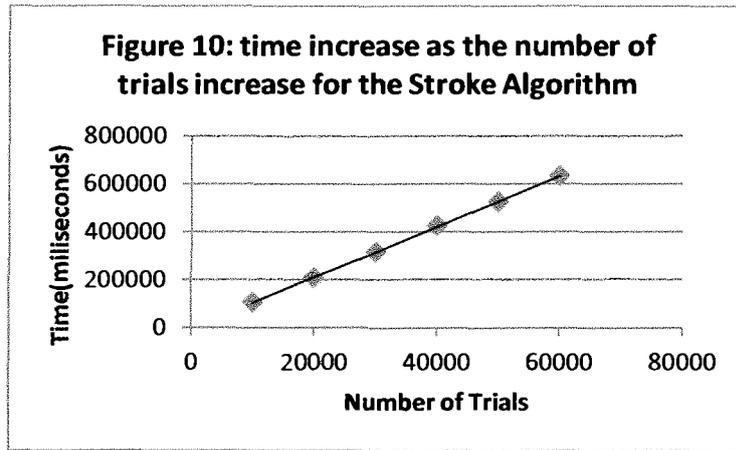
The first series of trials that were conducted were to see the increase in time to complete all recognition problems as the number of problems increases.



The second series of trials measures if the number of errors had anything to do with the number of trials.

2.3.2 With Distortion





3.0 Conclusion

From the tests that I conducted I can conclude that there are a few common sense conclusions that I can draw from my data, the first is that as the number of trials increases the number of errors also increases, this makes sense as the numbers were not in any particular order within the database so it makes sense

that there wouldn't be any particular pattern to the size of the increase. Also the length of time it takes to recognize the digits is directly proportional to the total number of trials. There is some fluctuation but that makes sense because different numbers take different amounts of time to analyse for example a 1 only requires a few vectors to be created and as a result the algorithm will run much faster than with something like a 2 which requires a large number of vectors to fully cover or with the Neural Network a more complex number like a 2 means more neurons firing and therefore more values to compute rather than something like a 1 which is simple and therefore doesn't cause so many neurons to fire and make the computation easier.

This leads into the answer to my question "which Neural Network is better in terms of speed and accuracy?" it turns out that the answer is that, it depends the neural network was better at recognizing the digits than the Stroke Algorithm but at the same time without distortion the Neural Network had an average error rate of 0.4582% and the Stroke Algorithm had an average error rate of 0.6997%. This really isn't that big a difference and taking into account that the stroke based method is 1.33 times faster on average than the Neural Network. But when there is distortion it is a different story, instead of continuing to be faster than the Neural Network the Stroke Algorithm is actually slower, the reason for this is that because distortion gets rid of most of the smooth curves and straight lines, more vectors have to be used to copy the shape and when more vectors are used it means that the pruning and simplifying algorithms have more to go through in their respective loops and as a result there is a massive drop in efficiency. While the Neural Network is not effected as much by having a more complex shape to recognize and as a result it doesn't take nearly as big a hit to efficiency.

These conclusions lead me to believe that while the Neural Network is in general the more accurate Neural Network and therefore has more use in everyday life where things are rarely ordered and in the perfect conditions for the algorithms. But in a situation where you want to transcribe a book the Stroke Algorithm would be perfect for that because that job will not throw strange letters that are not perfect and the conditions will be consistent, which means that it will not be hard for the Stroke Algorithm to recognize all of the letters and numbers and it is also more efficient and will get the job done faster than a Neural Network which is always desirable in any job.

3.1 Experimental Problems and Future Improvements

The experiment that I was conducting had a few problems, they were not big enough to affect my final conclusion too much but they may have resulted in the Neural Network and the Stroke Algorithm being closer with distortion. The main reason is because the Stroke Algorithm was written in java which is inherently less efficient than c++ also I think that the reason for the big jump in inefficiency with distortion was the result of the distortion being applied to the digits, which slowed down the recognition.

In the future I would like to expand the abilities of the Neural Network and the Stroke Algorithm to also include letters from the English alphabet and the ability to recognize whole words, if I were to do this however it would probably be a good idea to write a spell checker into the code which would implant a

bias based on the spelling of words and the likely hood that the word that was written was a word in the dictionary because that would help both algorithms be more accurate.

It might also be interesting to make a more general Neural Network Algorithm and see what the performance difference is between it and the convolutional networks.

✓
H = 2

4.0 Works Cited

O'Neill, Mike. "Neural Network for Recognition of Handwritten Digits." - *CodeProject*. 5 Dec. 2006. Web. 10 Feb. 2015. <<http://www.codeproject.com/Articles/16650/Neural-Network-for-Recognition-of-Handwritten-Digi#Using>>.

LeCun, Yann, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-Based Learning Applied to Document Recognition." *Yann.lecun.com*. 1 Nov. 1988. Web. 10 Feb. 2015. <<http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf>>.

LeCun, Yann, Léon Bottou, Genevieve Orr, and Klaus-Robert Muller. "Efficient BackProp." *Yann.lecun.com*. 1 Jan. 1998. Web. 10 Feb. 2015. <<http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf>>.

Fyfe, Colin. "Artificial Neural Networks and Information Theory." *Www.ece.rice.edu*. 1 Jan. 2000. Web. 10 Feb. 2015. <<http://www.ece.rice.edu/~erzsebet/ANNcourse/handouts502/course-cf-1.pdf>>.

"Structure of the Neuron - Part 1." *Structure of the Neuron - Part 1*. Web. 10 Feb. 2015. <<https://psych.athabascau.ca/html/Psych289/Biotutorials/1/part1.shtml>>.

LeCun, Yann, Corinna Cortes, and Christopher Burges. "THE MNIST DATABASE." *MNIST Handwritten Digit Database, Yann LeCun, Corinna Cortes and Chris Burges*. Web. 10 Feb. 2015. <<http://yann.lecun.com/exdb/mnist/index.html>>.

order
I -
C.P.

5.0 Appendix

Appendix A

```
import java.awt.Color;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import javax.imageio.ImageIO;
/**
 * Handles input of an image and traces the image with vectors.
 *
 * @author Daniel Rosen
 */
public class Input
{
    private BufferedImage img = null;
    private int[][] pixelsRGB;
    private boolean[][] pixels;
    private ShapedVector[] temp = new ShapedVector[1];
    private ShapedVector[] vectors = new ShapedVector[2];
    private ArrayList<ShapedVector> HQVector = new ArrayList<ShapedVector>();
    private int counter = 0;

    /**
     * Creates a new input class with the next image.
     */
    public Input()
    {
        Render(Main.imageName);
        seperateLetter();
        computeImage();
    }

    /**
     * Loads the image into memory so it can be traced.
     *
     * @param picture the image that will be traced
     * @return the memory location of the image
     */
    public BufferedImage Render(String picture){
        try {
            img = ImageIO.read(new File(picture));
            pixelsRGB = new int[img.getWidth()][img.getHeight()];
            pixels = new boolean[img.getWidth()][img.getHeight()];
        }
        catch (IOException e) {}
        return img;
    }

    /**
     * Returns the array that contains the raw pixel data.
     * @return the raw pixel data
     */
    public boolean[][] getPixels(){return pixels;}
    /**
     * Converts in the image to an array of pixels.
     */
    public void seperateLetter(){
        for(int i = 0; i < img.getWidth(); i++){
            for(int x = 0; x < img.getHeight(); x++){pixelsRGB[i][x] = img.getRGB(i, x);}
        }
    }

    /**
     * Turns the array of pixels into data that can be processed.
     * Also removes redundant vectors.
     */
    public void computeImage(){
        Point tempPoint = new Point();
```

```

for(int i = 0; i < img.getWidth(); i++){
    for(int x = 0; x < img.getHeight(); x++){
        if(pixelsRGB[i][x] == Color.black.getRGB()){//((0 << 16) | (0 << 8) | 0)}{pixels[i][x] = true;}
    }
}
for(int i = 0; i < img.getWidth(); i++){
    for(int x = 0; x < img.getHeight(); x++){
        //if there is a black pixel at the current x and y
        if(pixels[i][x]){
            Point temp = new Point(i, x);
            boolean check = true;
            for(int z = 0; z < HQVector.size(); z++){
                {
                    //and if it is not intersecting or passing by an already existing vector
                    if(temp.intersectingOrCloseVector(HQVector.get(z))){check = false;}
                }
            }
            if(check){
                //search for the next closest pixel so a vector can be created
                tempPoint = searchAround(i, x);
            }
        }
    }
}
}
/**
 * Returns all of the existing vectors.
 *
 * @return the existing vectors
 */
public ShapedVector[] getHQVector(){
    ShapedVector[] temp = new ShapedVector[HQVector.size()];
    for(int i = 0; i < HQVector.size(); i++){temp[i] = HQVector.get(i)}
    return temp;
}
/**
 * Returns an arrayList of Vectors.
 *
 * @return an ArrayList of Vectors
 */
public ArrayList<ShapedVector> getArrayVector(){return HQVector}
/**
 * Searches for an adjacent pixel to x and y
 * that is black.
 *
 * @param pixelX the current x location on the image
 * @param pixelY the current y location on the image
 * @return the end point of the Vector
 */
private Point searchAround(int pixelX, int pixelY){
    Point endPoint = new Point();
    //System.out.println(pixelX + " " + pixelY);
    //black above
    if(pixels[pixelX][pixelY-1]){
        HQVector.add(new ShapedVector(pixelX, pixelY));
        HQVector.get(counter).setDirection(ShapedVector.UP);
        endPoint = top(pixelX, pixelY);
    }
    //black top right
    else if(pixels[pixelX+1][pixelY-1]){
        HQVector.add(new ShapedVector(pixelX, pixelY));
        HQVector.get(counter).setDirection(ShapedVector.UP_RIGHT);
        endPoint = topRight(pixelX, pixelY);
    }
    //black right
    else if(pixels[pixelX+1][pixelY]){
        HQVector.add(new ShapedVector(pixelX, pixelY));
        HQVector.get(counter).setDirection(ShapedVector.RIGHT);
        endPoint = right(pixelX, pixelY);
    }
    //black bottom right
    else if(pixels[pixelX+1][pixelY+1]){
        HQVector.add(new ShapedVector(pixelX, pixelY));
        HQVector.get(counter).setDirection(ShapedVector.DOWN_RIGHT);
        endPoint = bottomRight(pixelX, pixelY);
    }
}

```

```

    }
    //black bottom
    else if(pixels[pixelX][pixelY+1]){
        HQVector.add(new ShapedVector(pixelX, pixelY));
        HQVector.get(counter).setDirection(ShapedVector.DOWN);
        endPoint = bottom(pixelX, pixelY);
    }
    //black bottom left
    else if(pixels[pixelX-1][pixelY+1]){
        HQVector.add(new ShapedVector(pixelX, pixelY));
        HQVector.get(counter).setDirection(ShapedVector.DOWN_LEFT);
        endPoint = bottomLeft(pixelX, pixelY);
    }
    //black left
    else if(pixels[pixelX-1][pixelY]){
        HQVector.add(new ShapedVector(pixelX, pixelY));
        HQVector.get(counter).setDirection(ShapedVector.LEFT);
        endPoint = left(pixelX, pixelY);
    }
    // black top left
    else if(pixels[pixelX-1][pixelY-1]){
        HQVector.add(new ShapedVector(pixelX, pixelY));
        HQVector.get(counter).setDirection(ShapedVector.UP_LEFT);
        endPoint = topLeft(pixelX, pixelY);
    }
    else
    {
        // if no black move on because points are ignored
        return null;
    }

    //System.out.println(endPoint);
    HQVector.get(counter).xEnd = endPoint.x;
    HQVector.get(counter).yEnd = endPoint.y;

    counter ++;
    return endPoint;
}

private Point top(int pixelX, int pixelY){
    if(!pixels[pixelX][pixelY-1]){return new Point(pixelX, pixelY);}
    return top(pixelX, pixelY-1);
}

private Point topRight(int pixelX, int pixelY){
    if(!pixels[pixelX+1][pixelY-1]){return new Point(pixelX, pixelY);}
    return topRight(pixelX+1, pixelY-1);
}

private Point right(int pixelX, int pixelY){
    if(!pixels[pixelX+1][pixelY]){return new Point(pixelX, pixelY);}
    return right(pixelX+1, pixelY);
}

private Point bottomRight(int pixelX, int pixelY){
    if(!pixels[pixelX+1][pixelY+1]){return new Point(pixelX, pixelY);}
    return bottomRight(pixelX+1, pixelY+1);
}

private Point bottom(int pixelX, int pixelY){
    try{
        if(!pixels[pixelX][pixelY+1]){return new Point(pixelX, pixelY);}
    }
    catch(ArrayIndexOutOfBoundsException e){return new Point(pixelX, pixelY);}
    return bottom(pixelX, pixelY+1);
}

private Point bottomLeft(int pixelX, int pixelY)
{
    if(!pixels[pixelX-1][pixelY+1]){return new Point(pixelX, pixelY);}
    return bottomLeft(pixelX-1, pixelY+1);
}

private Point left(int pixelX, int pixelY){
    if(!pixels[pixelX-1][pixelY]){return new Point(pixelX, pixelY);}
}

```

```

        return left(pixelX-1, pixelY);
    }

    private Point topLeft(int pixelX, int pixelY)
    {
        if(!pixels[pixelX-1][pixelY-1]){return new Point(pixelX, pixelY);}
        return topLeft(pixelX-1, pixelY-1);
    }
}

```

Appendix B

```
import java.util.ArrayList;
```

```
/**
```

```
* simplifies the pruned Vectors
```

```
*
```

```
* @author Daniel Rosen
```

```
*/
```

```
public class Processing {
```

```
    Input input = new Input();
```

```
    ArrayList<ShapedVector> shape;
```

```
    public Processing(){shape = input.getArrayVector();}
```

```
    /**
```

```
    * Simplifies the vectors down to 8 different ones.
```

```
    */
```

```
    public void process(){
```

```
        while(shape.size() > 8){
```

```
            for(int i = 0; i < shape.size(); i++){
```

```
                for(int x = i+1; x < shape.size()-1; x++){
```

```
                    //if the vector x is immediately after vector i
```

```
                    if(x != i && (shape.get(i).xEnd == shape.get(x).xStart+1 || shape.get(i).xEnd == shape.get(x).xStart-1))
```

```
                    {
```

```
                        if(shape.get(i).yEnd == shape.get(x).yStart+1 || shape.get(i).yEnd == shape.get(x).yStart-1)
```

```
                        {
```

```
                            if(shape.get(i).getLength() <= 2){
```

```
                                //and if the direction that the vectors are facing are similar
```

```
                                if(shape.get(i).getDirection() == shape.get(x).getDirection()+1
```

```
                                || shape.get(i).getDirection() == shape.get(x).getDirection()-1)
```

```
                                {
```

```
                                    //add the vectors together
```

```
                                    shape.get(x).xStart = shape.get(i).xStart;
```

```
                                    shape.get(x).yStart = shape.get(i).yStart;
```

```
                                    shape.remove(i);
```

```
                                }
```

```
                            }
```

```
                        }
```

```
                    }
```

```
                }
```

```
            }
```

```
    public ShapedVector[] getProcessedVectors(){
```

```
        ShapedVector[] temp = new ShapedVector[shape.size()];
```

```
        for(int i = 0; i < shape.size(); i++){temp[i] = shape.get(i);}
```

```
        return temp;
```

```
    }
```

```
}
```