International Baccalaureate Diploma Programme

Extended Essay

Subject: Computer Science HL

THE USE OF NEURAL NETWORKS IN VERIFYING HANDWRITTEN SIGNATURES

Research Question: To what extent can an image recognition neural network verify if a signature is genuine or forged?

Word Count: 3930 + 44 (doubled lines of code) = 3974

Table of Contents

Introduction

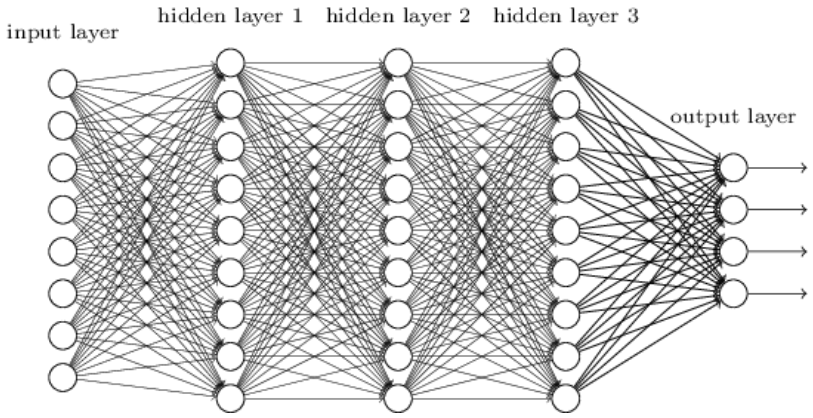
A signature is an important part of our identity, something we use in legal documents to prove that it is truly us who sign a certain document as an authorisation thereof in order to validate a transaction being closed on our behalf as a form of deliberation and informed consent. Therefore, it is no wonder that throughout the history of written documents there have been many people who have tried to impersonate another person by forging the latter's signature for their own gain. Because we cannot check each and every signature, unless we had handwriting experts analysing every document produced, signature forgery, e.g., in the financial industry is still a widespread practice (Johnson). There could be countless instances of forgery elsewhere in the corporate world that have cost millions regardless of elaborate systems of authentication in place, leading to costly litigations oftentimes to no avail. Therefore, the idea should be tested whether signatures could be verified by automated methods, digitally, where neural networks could be trained by examples of legitimate signatures to detect any attempted forgeries. Neural networks, being computer programs which simulate human brains by acting as neurons connected to one another and, most importantly, learning from mistakes, seemed perfect for said purpose. Since even the genuine signatures of one person often deviate to a certain degree, neural networks could be the key to noticing a pattern of writing different from all others which is invisible to the untrained eye. The question is, **to what extent can an image recognition neural network verify a signature?** To answer this question, firstly, an analysis of previous experiments and theoretical background should be done, and, secondly, an artificial neural network should be built, one trying to discover this imagined pattern that directs the system to the genuineness of a set of signatures.

Neural Networks – Theoretical Background

Components of Neural Networks

A neural network consists of perceptrons (another name for neurons in an artificial neural network), each of which has weights, biases and outputs from previous perceptrons assigned to them. A weight is, in essence, the importance of the perceptron's output in relation to other outputs (Nielsen). The weights at first have random numbers assigned because the network has not been trained yet, so that it has a base which can be checked and improved upon. A bias is a value that is also weighted and then applied to every perceptron to react according to the conditions of the network. The neural network is assembled in layers where each perceptron is connected to all perceptrons in the previous layer. The layers are divided into three self-evidently named categories – input, output, and hidden layers. The hidden layer count and sizes are very much dependent on the type of neural network required, and image recognition networks usually need at least 2 hidden layers, which are also called deep neural networks (Nielsen).

Figure 1. Example of a deep neural network's layers and neurons (Nielsen).



The exact number of perceptrons and layers needed cannot be calculated; therefore, many presumptions have to be made.

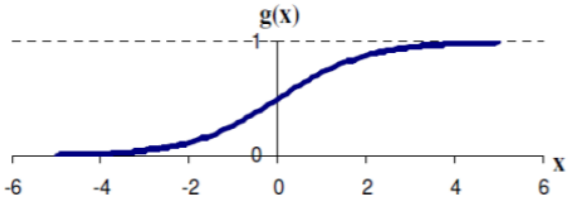
The raw output of a perceptron is calculated by the sum of inputs of perceptrons in the previous layer which are first multiplied by their respective weights plus an added bias, which itself is multiplied by a weight. This process is called forward-propagation. As an equation it would look something like this:

$output = bias + (input0 * weight0) + (input1 * weight1) + \cdots + (inputn * weightn)$ (Nielsen).

Normalisation by an Activation Function

If the range of a perceptron's possible output values were infinite, there would be considerable difficulties in the analysis of the output, and sometimes errors could occur. The solution to this problem is once again an imitation of the human brain in the way synapses fire. In the brain, the electrical impulses in neurons must meet a certain threshold to fire, and the perceptrons in artificial neural networks work in a similar fashion. To simulate the binary system in biology and to normalise, i.e., express in some standard form, the output, an activation function is added to the output. Often the sigmoid function is used because it is one of the simplest and easiest functions to understand and analyse. The sigmoid function (represented mathematically as $\frac{1}{1+e^{-x}}$ and visible in Figure 2) changes the range to numbers between 0 and 1 and utilises a smooth continuous function that normalises the data (Garg and Sharma 237).

Figure 2. Sigmoid function drawn in Cartesian coordinates (Garg and Sharma).



The action of creating a sum from the previous layer's weighted outputs and applying an activation function to it is called forward propagation.

Training the Network

Once the structure is created and the necessary input is acquired, the weights of the perceptrons still have not changed and are still the same random values. For the network to learn, the weights must be changed in such a way that the errors between the expected outputs and the actual outputs would be minimised. In supervised learning, neural networks are not taught from the beginning in the same way that children are taught by teachers and parents, but the acquisition of information is accomplished by learning from mistakes (these mistakes can be made only by the network trying randomly, which is why the weights are random at the start). The mistakes are compiled in what is called a cost function.

The Cost Function

Once the outputs have been calculated with random weights, they can be compared to the values of expected outputs, that is, the so-called target outputs. The cost function (or loss function) is simply a measure of how poor the network's performance is. The bigger the value, the poorer the performance. If the value is close to zero, the cost of the network is almost nothing, which means that the deep network is doing well. Neural networks can greatly vary in the choice of a cost function (because there are many varieties used for different types of neural networks) but most often a mean squared error is used (which usually is an average of many mean squared errors), the mathematical equation for the function being $e(i) = \frac{1}{2}\sum_i (desired_i - actual_i)^2$ (Trappenberg 151).

Backpropagation and Gradient Descent

Backpropagation is an algorithm which provides detailed insights into how changing the weights and biases affects the overall behaviour of the network. Once the cost function is calculated, the main goal of the network is to minimise the value of the function. In a sense, backpropagation and the training of the whole neural network is "just minimising a cost function" (Sanderson). This is achieved by a method called gradient descent.

To code a way for the purpose of getting the local minima of a function (the minimum value of the cost function), the world of computer science once again makes use of calculus. Gradient descent is an algorithm which finds the gradient of a certain function (in this case, a loss function) and then changes variables (in this case, weights) according to the type and steepness of the gradient. To better understand gradient descent, an analogy can be made of an imaginary ball rolling down in a parabola, that is, one of the simplest of functions with a minimum value. As can be seen in Figure 3, if the gradient of the ball's position on the parabola is positive (the ball stands on the right-side branch), the weights should be adjusted with a negative number, i.e., the value of the cost function would decrease if the ball rolled left and, therefore, down. The opposite goes for the situation where the gradient of the ball's position on the parabola is negative.

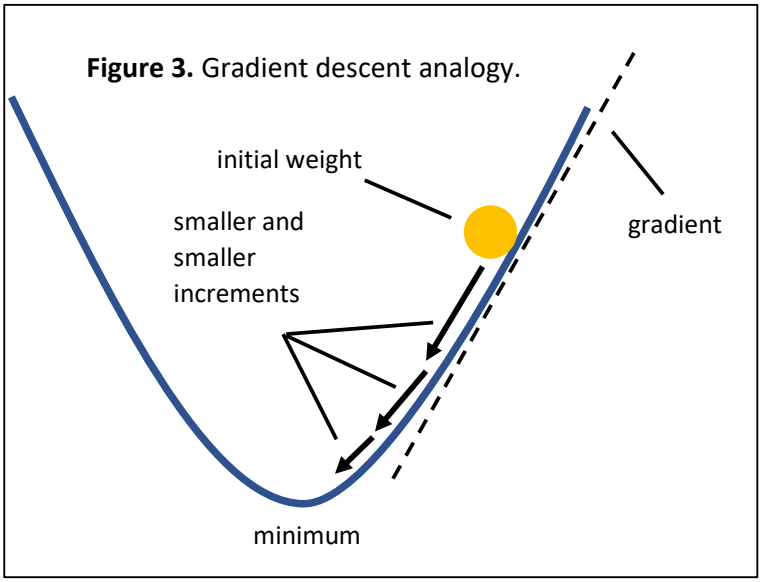


Figure 3. Gradient descent analogy.

Since the descent is multiplied by the gradient, the cost function is minimised not only in the right direction, but also always a little bit closer to the local minimum, somewhat resembling the paradox of the race of Achilles as he always travels half the distance to the finish line. Each iterative step to the local minimum is smaller because the distance is smaller until finally the computer approximates the step's size to be 0 (Nielsen) (Shiffman).

Image Recognition Neural Networks

There are many factors which could influence the results of neural network systems significantly when trying to verify if a signature is authentic or forged. For example, a signature in an image could be shifted in any direction (up, down, left, right), which the neural network considers an entirely new signature with different positions of its pixels instead of just the same signature but shifted. Convolutional neural networks (ConvNets or CNNs) are neural networks which take into account the shiftability of objects in images and recognise the same object but in a different place in an image. Additionally, CNNs work with a special system of numerous filters that can be applied to an image by methods such as pooling to acquire different views of it, e.g., to highlight edges or colour differences (Karpathy). The structure of CNNs is different from simple, fully-connected neural networks because CNNs use local receptive fields to gain feature maps from a hidden layer and pooling to simplify those feature maps.

Instead of every neuron being connected to every neuron in the previous layer like in fully-connected networks, CNNs use convolution by local receptive fields – a small block of neurons is connected to one or a few more of the next layer's neurons, resulting in what are called feature maps. The feature maps are also often called filters because each one looks at a specific feature of the image just as a filter would.

Figure 4. Example of convolution by local receptive field (5x5) and output neuron (Nielsen).

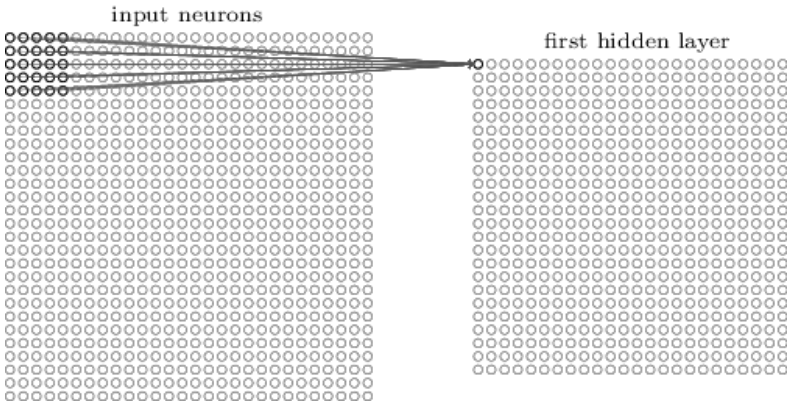
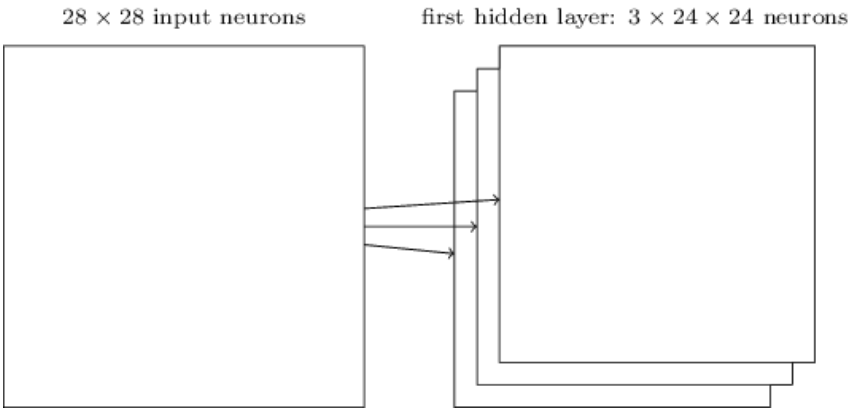
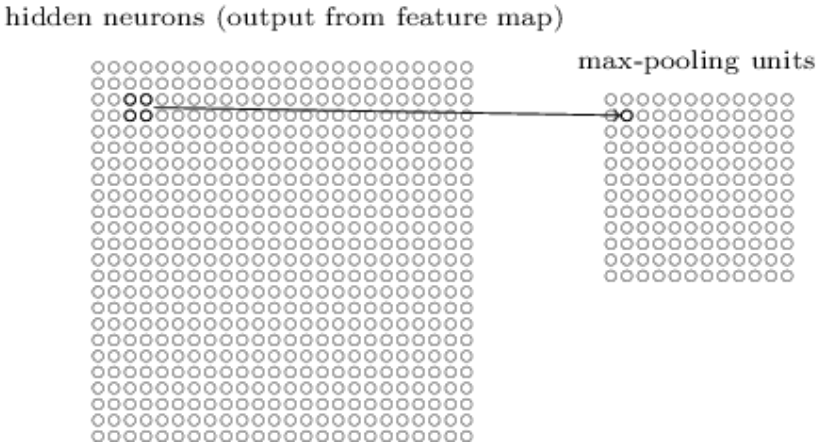


Figure 5. Convolution resulting in 3 feature maps (Nielsen).



Pooling, in contrast to convolution, simplifies the network by taking some measurement of a group of neurons and sending it to just one neuron in the next layer. Visually, it looks similar to creating feature maps, but pooling sends data only to one neuron from each region, like in Figure 6.

Figure 6. Example of max-pooling (Nielsen).



There are many kinds of pooling, but the most frequently used ones are max pooling (taking the maximal value in a region) and L2 pooling (calculating the square root of the squared sums from each neuron in the region) (Nielsen).

Thus, CNNs would be able to analyse different aspects of signatures to learn about them bit by bit. When comparing them to fully-connected networks, for example, a two-stage verifier made by Baltzakis and Papamarkos, their error is significant – 19.19% (Two-stage neural network classifier 102) in comparison to CNNs, for example, a 3.2% error (Nam, Park un Seo 10).

Signature Verification Neural Networks

Neural networks that verify signatures are a specific type of image recognition network where some other variables have been added. Since signatures differ from one another by their size (width and amplitude), slant, pen pressure and maybe outside factors like surface tension and the psychological status of the signatory, it seems impossible that an artificial neural network might take all of these factors into account. For example, to rectify the fact that people sign the same signature in different sizes, which is necessary to correct because

"this size variation could lead to performance degradation of signature verification" (Nam, Park and Seo 4), the authors choose to normalize the size by processing the data through an equation. Although this action allows the neural network to extract data from the signatures, the signatures themselves have still been modified to fit the needs of the network and, therefore, do not represent the true initial signature that was written by the signatory. In another research paper (Baltzakis and Papamarkos 96), the researchers even "skeletonize" the signature, that is, transform the width of ink to only one line, to make it easier to analyse, thereby eliminating an important factor from the equation.

Baltzakis and Papamarkos also highlight how many features a signature can have, e.g., height, width, area, size, baseline shift, slant, edges, cross points, etc, and then they try to keep in mind all of these features and incorporate them into the network, which results in a 19.19% false classification scenario (Two-stage neural network classifier 102). Although it is a "worst-case scenario", the error is still significant and a sign that even neural networks built for signature verification are not always as precise or accurate as intended.

Furthermore, the emotional and physical state of the signatory may vary at different times of writing his or her name, since human handwriting differs depending on whether they are distressed, happy, angry or in any other mood, which influences muscular tension, thereby influencing handwriting (Naftali 532), which consists of many different variables, such as slant, height and width. The context in which the signatory affixed his or her signature is also important, for example, caffeine is shown to influence handwriting by improving the signatories' writing speed and fluency (Tucha, Walitza and Mecklinger); therefore, the handwriting and, therefore, signature might be different if the signatory drinks a caffeine-full drink, such as an energy drink or regular coffee, before signing his or her signature. All of

these examples provide meaningful variables in the context of signature-writing, which can only be analysed by human experts, i.e., graphologists, when deciding if a signature is genuine or forged. When not considering these outside factors (*ceteris paribus*), neural networks (and especially CNNs) are capable of determining the reliability of a signature to some extent.

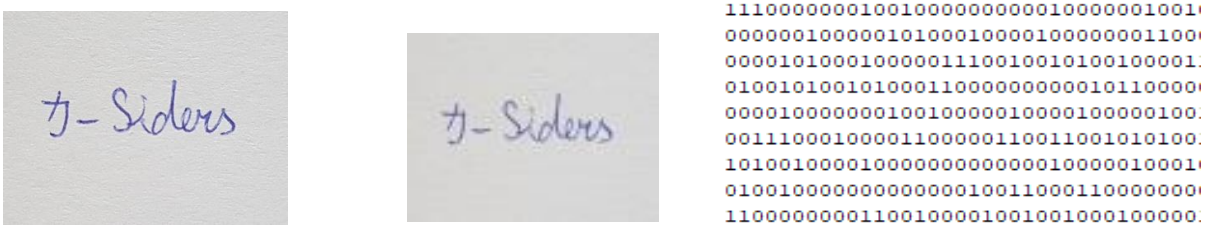
Experiment – Creation of an Artificial Neural Network

This particular deep neural network was built in object-oriented, class-based programming language Java, and a large part of it was built with the code from Finn Eggers' YouTube series "NN – Fully Connected Tutorial" (Eggers).

Input

Firstly, images or scans of the signatures had to be taken in order to generate input. All of the images were recorded in similar lighting conditions on the same paper, and the signatures were written with the same pen to minimise external factors that might influence the results and to emulate as closely as possible the similarity of the background for the signatures, as can be seen in Figure 7 (a).

Figure 7. Samples of: (a) original signature; (b) compressed signature; and (c) converted signature.



(a) *(b)* *(c)*

Secondly, the images had to be converted to input for the network to analyse. This can be accomplished by compressing the images in this specific system to a smaller size (75 * 100

pixels) using an Internet resource[1], and the paths to the files were given as properties of the image class, as can be seen in Figure 7 (b).

A file with a photograph extension (like .png, .jpeg or other) cannot be properly analysed in a fully-connected neural network, so to solve the problem the photograph of the signature is then converted into an array of ones and zeroes (as many as there are pixels, shown in Figure 7 (c) as a small fragment of the whole) so that each pixel is represented in the input. To convert the picture into an array, each pixel's colour is analysed, and if the red, green, and blue levels are below 650, the pixel is associated with the digit 1, if the levels exceed 650, the digit is 0. The code for it is shown in Figure 8.

Figure 8. The code for image conversion into input for the neural network.

```
public void photoFromFile(String fileName) throws FileNotFoundException,
                                                   IOException{
    BufferedImage img = ImageIO.read(new File(fileName));
    for(int i = 0; i < width; i++){
        for(int j = 0; j < height; j++){
            int p = img.getRGB(i, j);
            int r = (p >> 16) & 0xff;
            int g = (p >> 8) & 0xff;
            int b = p & 0xff;
            int digit;
            if(r + g + b < 650) digit = 1;
            else digit = 0;
            photo[i * height + j] = digit;
        }
    }
}
```

One loss from this conversion is that similarly shaded pixels end either as the same digit or the complete opposite, resulting in a very lossy translation from a multi-variable system (combinations of red, green, blue) to a binary system (ones and zeroes), which means that possibly valuable data is discarded. This affects mostly the edges of a signature, but in the

[1] Free Online Image Resizer and Converter, https://www.fixpicture.org/.

ones written on a tablet the edges are sudden and do not grade. Therefore, one further step is taken to emulate passport security signatures.

Construction and Calculation

The proceeding action is the construction of the artificial neural network itself. At first, the number of components have to be agreed upon. In this case, the input layer will consist of 7500 perceptrons, each holding the value of either 1 or 0, depending on the pixel's colour in that specific position. The exact best number of neurons and layers to choose when creating artificial neural networks is impossible to find out mathematically, which is why statistical analysis is usually used (Drouhard, Sabourin and Godbout 421). The count of each in this specific network was also guessed and then analysed, finally resulting in 2 hidden layers, each having 75 perceptrons in them. The output layer consists of only 1 perceptron since the network was built for the purpose of answering a simple "yes" (1) or "no" (0) question regarding the genuineness of the input signatures, the final layer's perceptron's output value being the answer.

The process of forward propagation will be written in Java code, as seen in Figure 9.

Figure 9. Simplified Java code for forward propagation.

```
double sum = bias;
for(int i = 0; i < inputs.length; i++){
    sum += inputs[i] * weights[i];
}
```

The main indicator of how well the network is doing both for the researcher and the network itself is the mean squared error, which can be calculated as shown in Figure 10.

Figure 10. Java code for mean squared error.

```
public double MSE(double[] input, double[] target) {
    if(input.length != inputSize || target.length != outputSize) return 0;
    calculate(input);
    double v = 0;
    for(int i = 0; i < target.length; i++) {
        v += (target[i] - output[networkSize-1][i])
            * (target[i] - output[networkSize-1][i]);
    }
    return v / (2d * target.length);
}
```

Solutions to Internal Problems

Training a deep neural network is not a simple task; therefore, some problems in the training phase of the system had to be solved. For example, pictures of the signatures have to be compressed into smaller sizes to make the input size manageable for the computer in charge of computing the results of the experiment. This compression can cause a loss of detail in the process. However, firstly, that detail would also be missing in a digitally written signature, e.g., for passport security, because of the lack of additional variables like the strength of the pushing of the pen and the length of holding it in this method, and, secondly, computers at the disposal of the government/security agencies would have no need for this compression because of the greater computing power at their disposal. Therefore, the loss of information would be negligible, too. Furthermore, another problem encountered in this experiment was when the last signature was used in the training process, and the output from the testing images tended towards the target of that last signature. This occurs because, as each signature is input in the network for training, it changes all of the weights of the network to ones that tend toward the signature's target (1 if it was genuine or 0 if forged), meaning

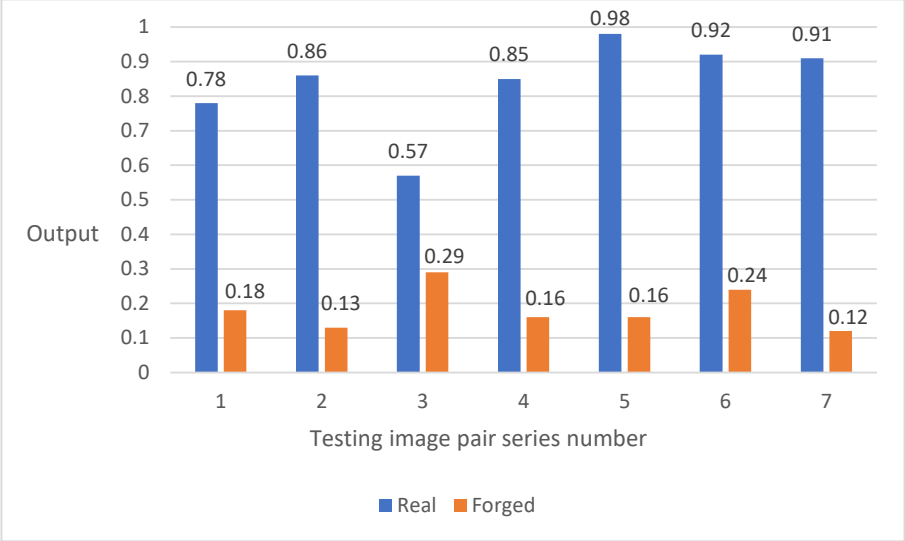
that the last signature's target would be the whole network's target. The solution found was to use the first signatures in the training cycle more and then gradually lessen the impact each consecutive image had on the neurons of the network, which led to an improved reliability of the resulting outputs. The mentioned problems which can occur in the training phase are one of the reasons why neural networks could be considered as ineffective signature verifiers as of yet.

Testing the Network and Results

Once the network was built and ready, and the neurons in it were trained to recognise forged signatures from the genuine ones, it was time to test it and get the results needed. The network was tested by inputting one image of an authentic signature and one image of a forged one without their targets (their desired outputs would otherwise already tell the network the status of the signatures). From each signature the neural network outputs the value of the images using its newly-trained neurons and the final mean squared errors, which

were the largest in the testing phase because they could not be changed by the network as was the case in the training phase.

Figure 11. Output from first set of signatures in testing phase.

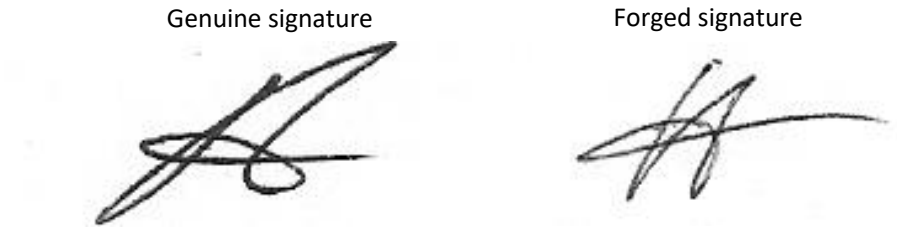


As can be seen in Figure 11, the output from genuine signatures (represented by the blue columns) was, at its maximum, 0.98, when its target was 1. Additionally, the output from forged signatures (represented by the orange columns) was, at its minimum, 0.13, when its target was 0. The third pair of authentic and forged tested images was outside the norm of the other tests either because of the irregularity of the genuine signature's representation of the signatory's usual signature, i.e., genuine signatures can sometimes differ from normal, or because of the above average quality of the forgery.

The experiment was then improved upon by making changes to the acquiring of the photos of signatures, firstly, by scanning them instead of taking a picture and, secondly, by immediately converting them into black and white instead of dealing with the ambiguity of paper colour or the problem of which is the exact edge of the ink of the signature. The second signature was chosen as a different type of signature – with only a few specific strokes (5), creating a sort of stamp or symbol, whereas the first one was a word in the handwriting of

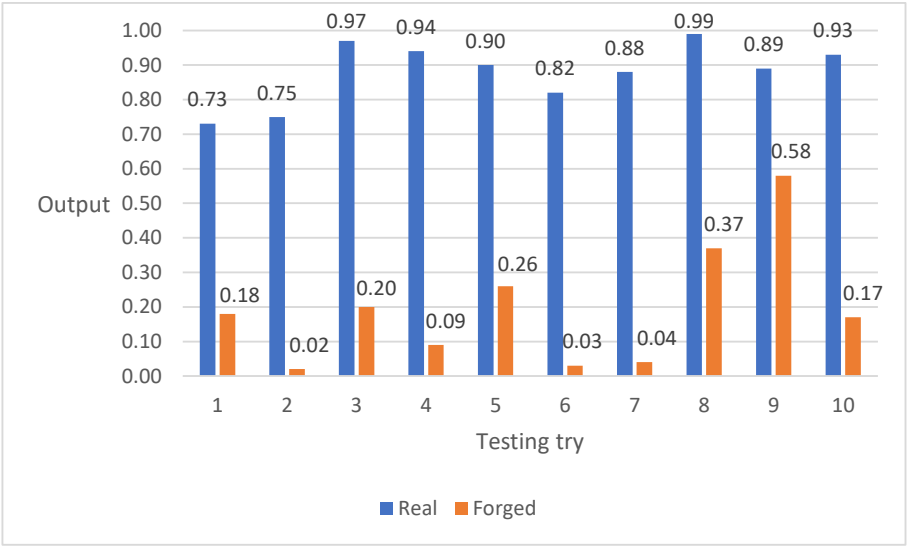
that person. To also contrast the first set of signatures, these were written in a box, such as one that could be found in passport security, to see if this limitation affected the accuracy of results in any way.

Figure 12. Second set of signatures, written in a box, in black and white.



The results of the second set of signatures were then compiled in Figure 13 below.

Figure 13. Output from second set of signatures.



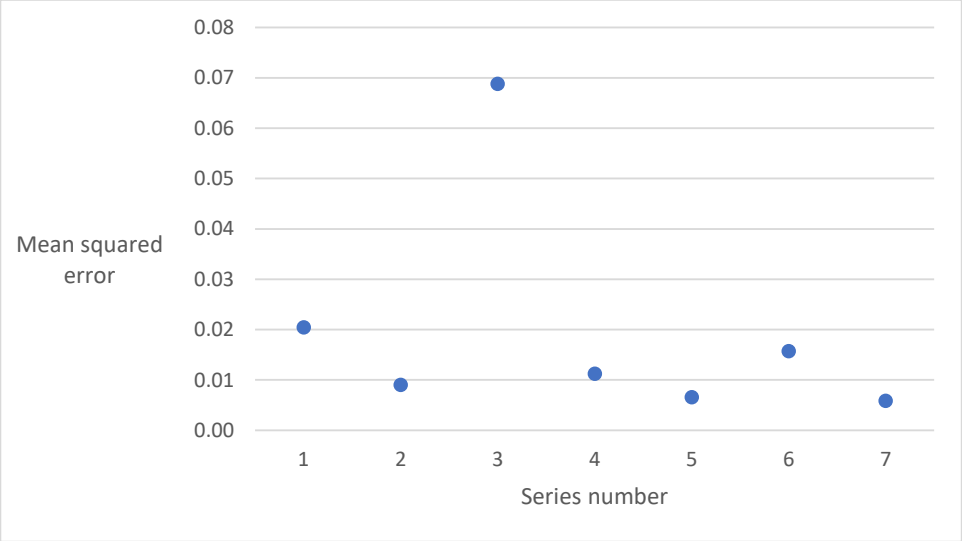
Judging from the results of the second set of signatures, it can be concluded that a neural network can recognise that a genuine signature is indeed a signature, but it fluctuates when it is given a forged one – possibly because the quality of the forgery differs. However, the second set of signatures did not provide any meaningful change in the results of the experiment.

Most importantly, the difference between the outputs of each pair of signatures can be seen, and the trend of each output towards their respective targets is noticeable; therefore, the making of a deep neural network that could distinguish between authentic and forged signatures could be considered a success. However, there still was a meaningful error to consider.

Analysis of the Results and Errors

The error in this experiment was measured by determining the mean squared error, which was calculated both during the training process and in the last phase of the experiment, the testing process.

Figure 14. Average mean squared error in first set of signatures.



As can be seen in Figure 14, the mean squared error ranges from about 0.6% to 6.9%. The error of the experiment varies from considerably small to understandably large values compared to that of errors in other image recognition neural networks, e.g., 3.2% (Nam, Park and Seo 10) and 1.24% (with the rejection rate of 4.506% and reliability factor of 0.987%) (Drouhard, Sabourin and Godbout 421).

Figure 15. Output with the largest average mean squared error.

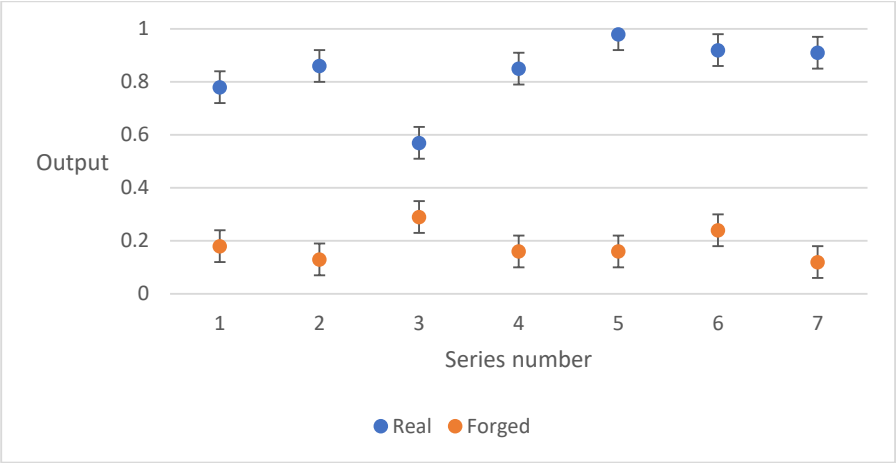


Figure 15 displays the outputs of each pair of signatures with the average mean squared error also shown. The lack of target outputs inside the error bars means that the neural network did not calculate the outputs to the best of its abilities, which can be seen as a possible future improvement to the network.

Conclusion

Convolutional neural networks could be used to verify passport signatures in less important matters, but experts should still handle highly important cases personally because of the current limitations of neural networks when it comes to signatures. The largest limitations are those of human psychology and context, since the neural networks do not yet consider the reasons behind the characteristics of people's handwriting and how it is affected by the personality of the signatory. The detailed aspects of a signature, like the combination of slant, amplitude and size, are determined by the character of the signatory and other

variables unobservable to the current neural networks. Additionally, the context is also very important when determining whether the signatory was overwhelmed by any particular emotion or under influence of any specific physical state or signals received by the system of perception when affixing his or her signature.

However, the experimental neural network did distinguish authentic signatures from forged ones and it did so with a much smaller error than anticipated. The network correctly labelled each tested signature and was always right in the value tending towards a 1 or a 0 (in the first set of signatures) even if it was close to the 0.5 edge (with one outlier in the second set of signatures). Therefore, even simple image recognition neural networks can be used to verify signatures to some extent.

Works Cited

Baltzakis, H. and N. Papamarkos. “A new signature verification technique based on a two-stage neural network classifier.” *Engineering Applications of Artificial Intelligence* (2001): 95-103. 6 March 2019. <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.231.5087&rep=rep1&type=pdf>.

Drouhard, J.-P., R. Sabourin and M. Godbout. “A neural network approach to off-line signature verification using directional PDF.” *Pattern Recognition* 29 (1996): 415-424. 6 March 2019. <http://www.etsmtl.ca/ETS/media/ImagesETS/Labo/LIVIA/Publications/1996/Drouhard_PR_1996.pdf>.

Eggers, Finn. *NN - Fully Connected Tutorial*. 31 January 2018. Video playlist. 15 August 2018. <https://www.youtube.com/playlist?list=PLgomWLYGNl1dL1Qsmgumhcg4HOcWZMd3k>.

Garg, Gourav and Poonam Sharma. “An Analysis of Contrast Enhancement using Activation Functions.” *International Journal of Hybrid Information Technology* 7 (2014): 235-244. 6 March 2019. <https://pdfs.semanticscholar.org/fe35/f6b8cd0c83c0e51242ec89df577c2b2ee721.pdf>.

Johnson, Erica. *Document forgery in financial industry more common than you'd think, past employees say*. CBC, 31 May 2017. 6 March 2019. <https://www.cbc.ca/news/business/financial-industry-employees-forge-documents-more-often-than-you-d-think-1.4138212>.

Karpathy, Andrej. *CS231n Convolutional Neural Networks for Visual Recognition*. Spring 2018. Stanford study notes. 6 March 2019. <http://cs231n.github.io/convolutional-networks/>.

Naftali, A. “Behavior Factors in Handwriting Identification.” *Journal of Criminal Law and Criminology* 56.4 (1965): 528-539. 6 March 2019. <https://scholarlycommons.law.northwestern.edu/cgi/viewcontent.cgi?article=5342&context=jclc>.

Nam, Seungsoo, et al. “Forged Signature Distinction Using Convolutional Neural Network for Feature Extraction.” *Applied Sciences* (2018): 1-14. 6 March 2019.

Nielsen, Michael A. *Neural Networks and Deep Learning*. 2015. 6 March 2019. <http://neuralnetworksanddeeplearning.com>.

Sanderson, Grant. *Gradient descent, how neural networks learn | Deep learning, chapter 2*. 16 October 2017. Video. 15 August 2018. <https://www.youtube.com/watch?v=IHZwWFHWa-w>.

Shiffman, Daniel. *The Nature of Code*. 12 December 2012. Book. 12 November 2018. <https://natureofcode.com/book/>.

Trappenberg, Thomas P. *Fundamentals of Computational Neuroscience*. Oxford: Oxford University Press, 2010.

Tucha, Oliver, et al. “The effect of caffeine on handwriting movements in skilled writers.” *Human Movement Science* 25.4-5 (2006): 523-535. 12 November 2018. <https://www.sciencedirect.com/science/article/abs/pii/S0167945706000522>.

Appendix

Results table of first set of signatures

Output		Mean Squared Error			
Real	Forged	Real	Forged	Worst	Average
0,78	0,18	0,0239	0,0170	0,0239	0,0205
0,86	0,13	0,0100	0,0080	0,0100	0,0090
0,57	0,29	0,0935	0,0441	0,0935	0,0688
0,85	0,16	0,0101	0,0123	0,0123	0,0112
0,98	0,16	0,0002	0,0129	0,0129	0,0066
0,92	0,24	0,0028	0,0286	0,0286	0,0157
0,91	0,12	0,0044	0,0073	0,0073	0,0059

Results table of second set of signatures

Output		Mean Square Error			
Real	Forged	Real	Forged	Worst	Average
0.73	0.18	0.0378	0.0167	0.0378	0.02725
0.75	0.02	0.0313	0.0003	0.0313	0.01580
0.97	0.20	0.0006	0.0209	0.0209	0.01075
0.94	0.09	0.0020	0.0043	0.0043	0.00315
0.90	0.26	0.0054	0.0337	0.0337	0.01955
0.82	0.03	0.0170	0.0004	0.017	0.00870
0.88	0.04	0.0070	0.0009	0.007	0.00395
0.99	0.37	0.0001	0.0692	0.0692	0.03465
0.89	0.58	0.0062	0.1654	0.1654	0.08580
0.93	0.17	0.0024	0.0149	0.0149	0.00865

Signature class file

```
package eesignatures;


import java.awt.image.BufferedImage;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import javax.imageio.ImageIO;
```

```
public class EESignatures {
   int width = 100;
   int height = 75;
   int[] photo = new int[7500];
   private String loc;
   public EESignatures(String location){
      loc = location;
   }
   String getLoc(){
      return loc;
   }

   public void photoFromFile(String fileName) throws FileNotFoundException, IOException{
      BufferedImage img = ImageIO.read(new File(fileName));
      for(int i = 0; i < width; i++){
         for(int j = 0; j < height; j++){
            int p = img.getRGB(i, j);
            int r = (p >> 16) & 0xff;
            int g = (p >> 8) & 0xff;
            int b = p & 0xff;
            int digit;
            if(r+g+b < 650) digit = 1;
            else digit = 0;
            photo[i * height + j] = digit;
         }
      }
   }
}
```