DEPARTMENT OF MECHANICAL AND AEROSPACE ENGINEERING

PROJECT REPORT for

AI Handwriting Recognition Using Artificial Neural Networks

by

Harry Lufturim Pjetri, Naimur Lamim Ruhid, Daanish Khan, Yousif Amin, Henry Kwok and Aaron Blanco

Student ID: *1910238, 1948928, 1913394, 1914609, 1915196, 1911261*

Supervisor: *Dr QingPing Yang*

Submitted in part fulfilment of the requirements for

the degree of Bachelor of Engineering

in

Mechanical Engineering

APRIL 2022

Contents

Nomenclature

ANN	Artificial Neural Network
CNN	Convolutional Neural Network
HMM	Hidden Markov Model
OCR	Optical Character Recognition
RNN	Recurrent Neural Network
SVM	Support Vector Machine
ICR	Intelligent Character recognition
MDLSTM	Multi-dimensional long short-term memory
API	Application Programming Interface
HOG	Histogram of Oriented Gradients
GLCM	Grey Level Co-occurrence Matrix
BCHWTR	Bank Cheque Handwritten Text Recognition
RELU	Rectified Linear Unit
ELU	Exponential Linear Unit

Abstract (AMIN)

This report researched, designed, and tested a handwriting recognition neural network as part of a larger deep learning subject. The report focused on using lowercase handwritten characters to train a neural network. Using a simple layer structure (shown in Figure 15) with pre-calculated learnables which reduced the image matrix from 30x30 to 2x2. Once training was completed, the neural network achieved a validation accuracy of 88.7%. Using the padded word “bench” shown in Figure 23, as a test for the model, the network managed to accurately predict each character and output these letters in a sequential order to a separate text file. The conclusion is that the neural network has been designed to be reliable for lowercase handwriting recognition however there are areas for improvement for a more complex network that allows for word segmentation in sentences.

Introduction (KHAN)

As technology is progressing the need for pen and paper is slowly diminishing and the move towards electronically submitted documents is rising exponentially. As such there are calls for the need in handwriting to text recognition systems. These have applications such as converting students into a document that can be saved as a pdf or the use in the banking industry where peoples handwriting is analysed to protect against fraud. No matter the use, there is a key underlying feature of all recognition systems and that is the algorithm used by the computer to train itself for recognition and save that information as a way of processing future inputs. The purpose of this study is to develop a handwriting recognition model that can accurately read images containing words and output them correctly in a quick efficient manner, this will be accomplished using preliminary trials and experiments so that a baseline for accuracy can established and improved upon. To ensure that computational power is not wasted it is also the aim to keep the amount of time the model takes to read the input low.

Literature Review

Background and Theory (RUHID)

Brief History

In reference (1) it is stated that deep learning is a term associated with machine learning, which uses algorithms and layers to process data to mimic a thinking process. Each layer is a simple algorithm with one end function, whether to activate the neuron (2). This activation function determines if the neuron’s input is important or not. This information from one layer is passed onto the next layer, which will undergo a different algorithm to process a different part of the information.

As written in (1) Kunihiko Fukushima designed an artificial neural network called Neocognitron in 1979. This neural network was deemed to be the first CNN, which included multiple pooling and convolutional layers. The Neocognitron was designed to learn how to recognize visual patterns as well as be able to adjust the design to focus on the more important features. Building on this, in 1989, Yann LeCun combined the use of CNN with back propagation. Back propagation is a method of using errors to

train models. Essentially, as the program passes through each layer in a network, the back propagation goes back through the previous layer to adjust parameters (3).

Dataset

Reference (4) defines a machine learning dataset to be “the collection of data that is needed to train the model and make predictions”. The data within a dataset depends on the need for the model, such as images, texts, audio etc. Sydorenko (5) stated that “a dataset contains a lot of separate pieces of data but can be used to train an algorithm with a goal of finding predictable patterns”. As of this moment, there are a wide range of datasets available for handwriting recognition. Each dataset differs in the type of data stored, the amount of data stored, and the type of pre-processing needed.

A dataset is usually split into three separate parts: training, validation and testing as shown in Figure 1. The reasoning behind the splitting of data is to test the accuracy of the model with untampered data. The training dataset is usually the largest of the split group and is used to train and update the model. The validation dataset is crucial in avoiding biased training on the model. It ensures that any increase in accuracy of the model is legit by testing the model with unused data. The testing dataset is used for the final code, to test the accuracy of the model on “new” data to prevent overfitting (4).

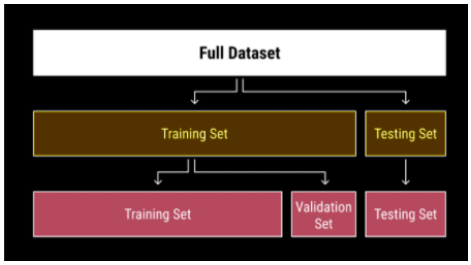

Figure 1, Splitting of a dataset into training, validation, and testing datasets (5)

Online and Offline

Tappert, Charles et al., (6) stated that online handwriting recognition works on the principle of tracking pen movements as the user writes. Tappert discovered that the recognition software can be slower than the writing speed due to different algorithms and processor power. To enable online recognition, the system “requires a transducer that captures the writing as it is written” such as a “tablet or digitizer” (6). On the other hand, offline handwriting recognition utilizes pre-written images to be processed into a bit pattern. It is a form of OCR, which is a technology designed to recognize printed or handwritten characters on images of documents.

Tappert et al., (6) explained that the online process of capturing dynamic information is an advantage because transducers can track the pen stroke and store the movement as a set of coordinate points. By contrast, the offline method requires extra processing to convert the data, which reduces the model accuracy. Another advantage expressed is the adaptation of the user’s writing in real time if the recognition software is unable to accurately determine the characters. The information can then be reused to correct future chances of error. The advantage for offline over online is the lack of need for specialized equipment.

Reference (7) and (8) both evidently show that online recognition rates are higher than offline. (7) recorded the recognition accuracy for offline unconstrained handwriting to be 78% for 1000 words. For online, it was discovered that the recognition rate was 80% for 21,000 words in pure cursive. To back this information, the work produced by reference (8) displays that the unconstrained word recognition accuracy for online is 79.7% and offline is 74.1%. While the accuracy values are different per case, they both indicate towards online being more accurate at word recognition.

Constrained and Unconstrained

Constrained handwriting are texts which are separated by lines akin to lines found in a A4 notebook. The basis behind the theory is to use the boundaries to help separate sentences, words, and characters when using OCR. On the other hand, unconstrained handwriting are texts with no boundaries. These can be road signs, text on a document, or writing on a blank piece of paper. Reference (9) defines unconstrained to be “no fixed lexicon and words have unknown length”. The problem of unconstrained images can be broken down into two stages: Detection and Recognition. In the detection phase, the system identifies each word and creates a box around the word to separate them. In the recognition stage, the system reads each box individually to identify the text within (9).

Recognition Models (PJETRI)

The recognition of handwriting starts with making a model and training it to distinguish characters, words and eventually sentences. This is done using models such as RNN and CNN which have different benefits such as ability of using previous context and memory to make predictions or decisions. Following below is a breakdown of the different models used.

Recurrent Neural Networks

Initially created in the 1980’s RNN’s are based on a deep learning network structure that harnesses past information to improve the performance of the network on current and future inputs. As shown in Figure 2 below, there is a loop within the structure that along with the hidden layer allows the RNN to store short term data and work back through it to make predictions. The data begins in the input layer where it is then processed and if relevant it is stored in the hidden layer which future data can be cross referenced against before being outputted (10) (11).

The clear advantage of RNNs is the ability to use the short-term memory to provide context to data by accessing the past and future, the usefulness of this can be seen in Figure 3 when the ‘n’ is removed it is no longer recognisable but with the context of the other letters it can be distinguished. Additionally, RNNs apply a weight matrix to the previous inputs as well as tweaking the inputs using gradient descent and backpropagation through time and thus RNNs can be seen as a sequence of neural networks. Exploding gradients and vanishing gradients are both problems that are exhibited with RNNs. Exploding gradients being when unreasonably high importance to the weights is assigned by the algorithm resulting in diverging accuracy instead of improvements and vanishing gradients are when the gradients become too small, and the model no longer learns or takes too long. The latter can be solved easily by

truncating or squashing the gradients whereas LSTM was invented to solve the vanishing gradient problem (10) (8).

LSTM replaces the simple computation node within the RNN unit with a computation block, within this block the LSTM can read, write, and delete information from its stored memory much like that of a computer. This memory is gated meaning that the cell decides what is stored and deleted, there are three gates: the input, forget and output gate. Using an RNN model with LSTM A. Graves et al were able to obtain a word accuracy of 79.7% (14.7% larger then HMM) for online data writing recognition and a word accuracy of 74.1% (9.6% larger than HMM) for offline data recognition owing to the improved training through backpropagation and the extended memory for the surrounding characters (8).

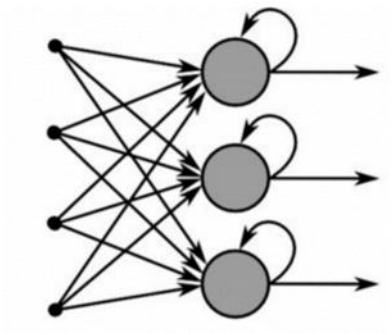

Figure 2, Schematic of RNN showing input into hidden layer and loop back feature (10)

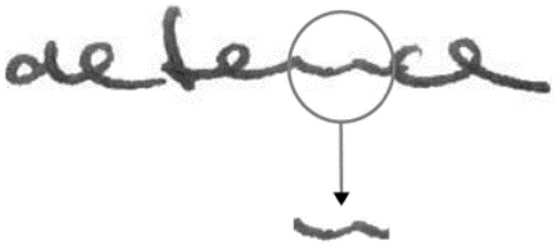

Figure 3, The word 'defence' written clearly except when isolated the 'n' is ambiguous (8)

Convolutional Neural Networks

CNNs are a form of ANN that use a system consisting of three layers: the input layer, output layer and the hidden layer which retains many more convolutional layers including pooling layers, fully connected layers, and pooling layers. They are used for operations such as video and image recognition making them very useful for handwriting recognition amongst other things (12).

The job of the CNN is to reduce images into data that is easier to process whilst maintaining accuracy and important features. It does this by first extracting high level features such as edges, colours and gradient orientation and then more and more data is extracted as the layers progress. Adding many layers has the benefit of achieving higher levels of accuracy due to the extra data extracted however comes at the cost of an increase in computational power (13). As seen in Figure 4 below the raw data is processed in this image in the form multiple kernels that convolve the image to generate a feature map as an output (14). Once the data has passed through the

hidden layers it is then processed through a feed forward network like those used in other models such as RNN and HMM.

CNNs are used in many different algorithms by companies such as Google (googleNet) due to its high precision and ability to self-learn without constant input from an engineer (12). However, the model is often very complex, and it is often hard to interpret and explain the model as well as requiring a lot of data to train it. Baldominos et al. reviewed an experiment using a CNN based model that recognised handwritten numbers using training from the MNIST database and it was able to recognise numbers with a 0.37% error rate and those numbers that failed classification also would've been hard to distinguish for humans (14). This demonstrates the strong capabilities of the CNN model structure and further justifies the use within image recognition.

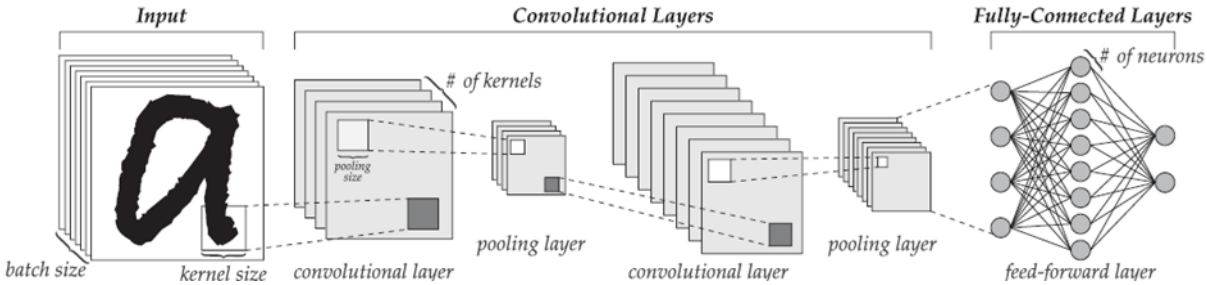

Figure 4, Typical structure of a CNN (14)

Existing Recognition Systems (KWOK)

Current Handwriting recognition systems have been put in place for a variety of industries and areas such as insurance companies, banks, and the healthcare industries (15). The use of OCR, ICR and MDLSTM RNN's have enabled these industries to reduce human labour costs and work hours through the digitalisation of their business models. Much of the development towards these handwriting recognition systems go towards offline text recognition, fraud detection, signature verification and essay marking (15). Current programmable interfaces are being used and sold in other online services such as Google Cloud Vision API and Microsoft Azure Read API which can extract handwritten texts of different languages and symbols from images. Other online recognition uses may consist of real-time pen to tablet drawing in which drawn text can be analysed and translated into written text.

BCHWTR is a system that is built for cheques in Indian banks, used to recognize handwritten characters presented in the `payee name` box (16). These techniques used involve the processing of handwritten cheque images. It states the use of SVM based classification process which uses a self-generated dataset of bank cheque images whereby 100 different people provided handwritten texts in a separate bank cheque each. Figure 5 shows a simple visualisation of how the system functions flows from input to output.

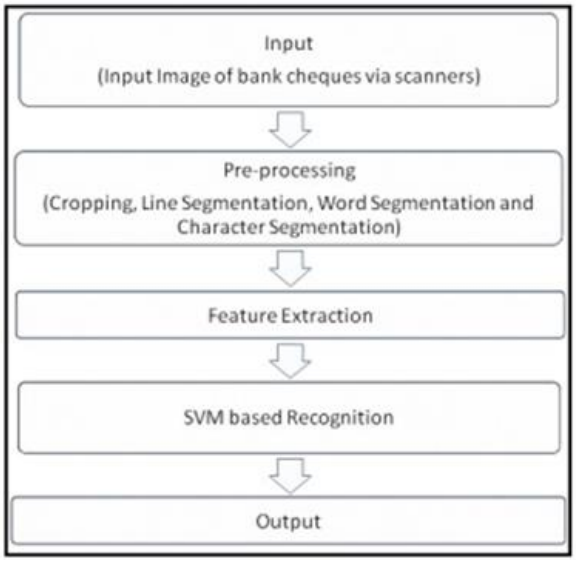

Figure 5, Basic framework for proposed BCHWTR (16)

It was noted that the proposed system utilises HOG and GLCM to differentiate between classes of inputted patterns. Post segmentation of the text has yielded an accuracy of 91.83% with the added use of noise removal technique. (16)

As mentioned before, signature verification is an important feature in increasing the detection of fraudulent activity and forgeries. CycleGAN is a based scanning artefact removal deep network designed to remove noises and blur from documents (17). This is used in conjunction with a signature representation which utilises CNN's and is noted to perform well using ResNet-50 or VGG-16 (17). The needed dataset used to train the signature representations are from collected order documents and signature declaration documents. It may be noted that each "signature declaration document" contains 3 of the same signatures which are then used as reference. The representation learning dataset consists of 109 different signatures which through augmentation brings it to a total of 9000 signatures (17). For verification, two sets of 178 different signatures are used of which are either unstamped or stamped. Finally, a Tobacco-800 dataset consisting of a selected 4200 signatures for training (17). The models are then implemented into TensorFlow which utilises Python as the coding language and set with an initial learning rate of 0.001. The error rates are shown below.

Test Setups	EER_{global}					
	VGG-16	VGG-16$_{cleaned}$	VGG-16$_{inverse}$	ResNet-50	ResNet-50$_{cleaned}$	ResNet-50$_{inverse}$
Reference Signature - Unstamped Target Signature	0.18	**0.16**	0.18	0.20	0.20	0.20
Cleaned Reference Signature - Cleaned Unstamped Target Signature	0.18	**0.17**	0.18	0.19	0.19	0.20
Reference Signature - Stamped Target Signature	0.33	**0.31**	0.32	0.34	0.34	0.34
Reference Signature - Cleaned Stamped Target Signature	0.23	0.23	**0.22**	0.26	0.26	0.26
Cleaned Reference Signature - Cleaned Stamped Target Signature	**0.22**	**0.22**	0.23	0.26	0.26	0.23
Tobacco-800	0.24	**0.17**	-	-	-	-

Figure 6, Signature verification results (EER Equal error rate) (17)

A further subjective evaluation consisting of 18 volunteers where each is shown 60 pairs of signatures to verify the validity of it. Figure 7 shows the accuracy of the verification methods compared to a human interpretation which although is higher, requires human intervention and possibly greater processing time of which a coded network could reduce greatly especially with a larger volume of data.

Evaluation Method	Accuracy (%)		
	Human	$VGG\text{-}16_{cleaned}$	$ResNet\text{-}50_{cleaned}$
Majority Voting	91.66	76.38	75.00
Individual	89.25		

Figure 7, Results of the subjective evaluation (17)

Recent achievements in off-line handwriting recognition systems (AMIN)

A study carried out by B. Verma, M. Blumenstein and S. Kulkarni on offline handwriting recognition systems suggested the use of two approaches to improve the existing handwriting recognition system. The global approach uses identifying features to recognise words, whereas a segmentation approach divides the words into letters to be recognised individually and uses them to check against specific words (18). Developing an offline handwriting recognition system is a very complex task because of its nature and industrial importance as it can be utilised for different applications such as bank cheque recognition, reading machines for the blind, postal address recognition and postal address recognition (19).

The table below in Figure 8 shows the results of the off-line handwriting recognition research studies accomplished around the world to obtain the highest accuracy, and this is divided into three categories, numeral handwriting recognition, character and cursive word recognitions, and the results heavily depend on databases selection.

Scope for Improvement

	Authors	Recognition Rate [%]
Numeral	Denker et al. [28]	86
	Bottou et al. [27]	91.9
	Srihari [19]	89-93
	Lee [24]	99.5
Character	Koutsougeras and Jameel [38]	65-98
	Liou and Yang [36]	88-95
	Srihari [19]	85-93
	Shustorovich [37]	89.40-96.44
Cursive Word	Edelman et al [33]	50
	Lecolinet et al. [2]	53
	Leroux et al. [2]	62
	Chen et al. [34]	64.9-72.3
	Bozinovic et al. [12]	54-72
	Senior and Fallside [35]	78.6
	Simon [2]	86
	Guillevic and Suen [6]	76-98.5
	Bunke et al. [26]	98.27

Figure 8, Summary of recognition rates (18)

Techniques based on ANNs dictionary approach gives 100% better handwriting recognition words rates. The ANN system is designed to have 700 inputs character matrix size of 25x28, a learning rate and momentum of 0.2,25 hidden units and 26 outputs.

***Real-time on-line unconstrained handwriting recognition using statistical methods* (20). (KHAN)**

The investigators aim was to develop a system that can read written text presented in different variations of style from clear cursive handwriting to discrete unclear handwriting. To accomplish this the general recognition framework of the system was composed of HMM as these systems main feature is to simultaneously carry out recognition and segmentation of handwriting in an integrated procedure which in turn enables the system to recognise a combination of both cursive and mixed handwriting fulfilling the system function. The investigators collected a data pool of 100,000 characters from 100 writers translating to 1000 characters per word, by doing this the HMM system was given a broad range of sample information where the system would be able to build text models of different variations of the same character. The models code used the data pool of sample characters and a Gaussian model mixture which was used determined the probability of each character and output a transcript.

Experiment and Results

To test the model's capacity for error the investigators put the system through an examination looking at the systems capability to recognise small, medium, and long ranged vocabulary in a list of 21,000 characters. The error rates of the experiment are shown below in Table 1, boasting a 9% error for small vocabulary on the detailed match in comparison to the almost 15% error recorded for the fast match. The second test carried out looked at the model's recognition speed and tested to see if it differed for the vocabulary size. The results in Table 1 below show that the speed at which the writing could be processed showed little variation with an average speed on 0.45s for the short, medium, and long vocabulary tests.

Table 1, Error Rates and Computation Time for word Recognition

		Small Vocabulary (3,000 words)	**Medium Vocabulary (12,000 words)**	**Large Vocabulary (21,000 words)**
Fast Match	Independent word error rates	14.9%	25.1%	27.9%
	Recognition Time (s)	0.29	0.33	0.36
Detailed Match	Independent word error rates	9%	14.9%	18.9%
	Recognition Time (s)	0.40	0.45	0.48

Methodology

Dataset (BLANCO)

There are various datasets that are considered such as IAM, NIST, Stanford, OCR and Devangri which are well known to be some of the best handwriting datasets for machine learning (21). The IAM Handwriting database especially stood out compared to NIST because the NIST database only consists of character images whereas IAM consists of word images. Word images are more appreciated as the function of the program that will be made is to detect words rather than characters alone, therefore the IAM database would be ideal for the project.

Choosing a dataset depends on the hardware of the user's PC. The larger the data, the larger the computational operations are in terms of memory. To compute this data efficiently, a GPU is an optimum choice. Since they are optimised for training deep learning models and can process computations simultaneously. Lastly, a GPU has a larger bandwidth, more cores and contains VRAM. VRAM is video RAM memory which allows the GPU to take in a lot of data and process it whereas a CPU has a significantly smaller amount of memory and cores compared to a GPU (22).

The IAM dataset contains 115,320 isolated and labelled words. The size of this database is too large for the computers that are being used to train the neural network as it will take too long. So, with further research, a dataset created by Dhruvil Dave is being used which was posted to Kaggle as an open dataset. This data set consists of 3410 images with the dimensions of 1200x900 of handwritten English characters (lowercase and uppercase) and numbers. Despite finding a suitable dataset, the number of images that are being used for training the neural network are 1430 as only the lowercase images are being used.

Pretrained Models (KHAN)

Pretrained models are networks that have been created by other users to solve similar issues, for the purpose of this study the pretrained networks intend to be used have been engineered to recognise images of text near perfectly the benefit of this is that it means that the networks will function well with the model and enable quick and efficient results to be received. The two pretrained networks intended to use for this study are Darknet-19 and Squeeze-net (acquired from the MATLAB designer application). The initial plan is to import the current dataset into these individual networks to test the accuracy and performance duration of these networks, by analysing the results received it will highlight the suitable techniques that the networks use to function progressively and allow the features to be incorporated into the studies model giving higher accuracies and more robust recognition.

Figure 9 shows the structure of layers in the darknet-19 network, the primary reason behind investigating this pretrained network is to give a better understanding of filter parameters and the effects they will have. The network starts with an RGB image of the size 256,256,3 in the input layer. Following the input layer, you will see that the layers follow a repetitive pattern of convolution layer then batch normalisation layer followed by the leaky relu layer and ending with the pooling layer, the network follows this pattern for a total of 64 layers. The first convolution layer is 3x3 with 32 filters with the filter sizes proceeding afterwards following a non-linear pattern ranging between

filters of 128 to 1024. The network has a high weighting with a total of 20.8 million learnables which has the benefit of bringing higher accuracies but also the drawback of longer training periods due to its size.

Figure 9 Darknet-19 Structure

Figure 10 shows the structure of layers in the squeeze-net network, the main reason this pretrained network is being tested is because it will help to give an insight into concatenation layers which are used to bring separated layers together. Here the network starts with a an RGB image of the size 227,227,3 in the input layer. The total amount of layers within the network structure is 68 and follows a layer pattern of the convolution layer, relu layer, the concatenation layer, and the pooling layer. The first convolution layer is 3x3 with 64 filters with the proceeding convolution layers afterwards varying between 1x1 to 3x3 to 5x5. The network has a total weighting of 1.2 million learnable. The size of this weighting implies that it will deliver less accurate results but at a faster rate.

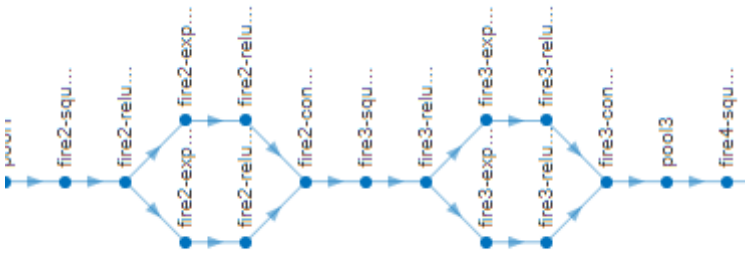

Figure 10 Squeeze-net Structure

Network Architecture (PJETRI)

Using the information gathered in the literature review, the most appropriate method of recognising handwritten text from images in a non-cursive writing style is with the use of convolutional layers to form a CNN. If the writing was written cursively then an RNN would be more suited with the inclusion of an LSTM layer. This would allow back propagation to predict the output of the character along with the standard feedforward image classification. Furthermore, the study of the pretrained networks suggested that the convolutional layers were more appropriate for distinguishing shapes and figures from images as well as being more efficient as they reduce the number of nodes within the network. For these reasons the network investigated within this report will be based around the use of convolutional layers and finding the optimum weights and biases to operate the network efficiently and accurately.

Activation layers are also under investigation as there is a benefit of using each of them. Relu layers are widely used in many CNN networks as they inhibit the occurrence of negative integers in the computation of the matrix, if the number is positive then it is output directly into the next layer however if it is negative then it is set to zero. This aids in reducing the computation required within the modal as it stops the calculations from spiralling exponentially out of control. Sigmoid functions serve a similar purpose to Relu layers however it was found that they did not perform as well. Leaky Relu layers can be seen in the structure for darknet layers, and they have similar performance and computation time to Relu layers. Finally, is the possibility of using an Elu activation layer, the main difference of this layer is that it allows the values to be negative as well as producing a smoother output as opposed to the sharp output generated by Relu. As such it is within the interest of this experiment to investigate the use of Relu, Leaky Relu and Elu layers to find which best suits the image recognition model and provides the quickest most accurate results.

To reduce the sample size of the network after the convolutional layer a pooling layer is often used. The most used within CNN architectures are max pooling layers, the largest value of the filter applied by the pooling layer is taken and therefore reduces the sample. Max pooling layers are the most used within CNNs and can be seen in both the squeeze net and darknet 19 networks as well as having many reports available outlining the high efficiency of this layer at reducing computational cost whilst also ensuing good accuracy. As such a max pooling layer will be used within the architecture for the handwriting recognition network.

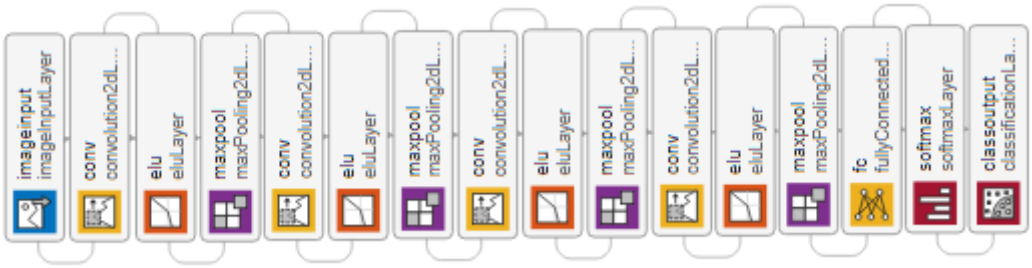

Figure 11, Schematic of initial Network Architecture

Figure 11 above displays the schematic of the initial CNN created using the Neural Network Designer in MATLAB. The network begins with the image input layer with image size of [227 227 1], the matrix is then convoluted and pooled sequentially until the fully connected layer and classification output. Elu layers are used here to test their effectiveness at transmitting the output of the convoluted layers, furthermore four max pooling layers have been added to simplify the computation as the initial input size is large and subsequently so are the convoluted matrices. Finally, within the output layers a SoftMax function is used to activate the classification function as there are 26 possible class outputs.

Segmentation Methods (PJETRI)

In terms of handwriting recognition there are many different approaches that can be taken and all of them depend on the form of the inputted data as well as the intended output. As the aim is to achieve a model that can recognise the image input that contains text and to be able to output that into an alternate digital format it means that segmentation is necessary.

The first method is the separation of blocks of texts such as combined sentences and paragraphs which need to be split into singular lines. The way to approach this is by means of horizontal projection that uses the binarized image matrix to locate the gaps between the lines. Once the image has been input a matrix of (x by y) is produced where x corresponds to the number of horizontal pixels and y corresponds to the number of vertical pixels. The matrix is then binarized to produce a new matrix where the values are ‘1’ for black pixels and ‘0’ for white pixels as opposed to the original grey scale image that has a scale of 255. A vertical sweep of the matrix is then performed from top to bottom where each row is summated, and that value corresponds to the matching y position. Shown in Figure 12 below the areas with a lack of black pixels produce a low value for the sum of the row and therefore this can be used to locate the gaps between the line of the text. By configuring an ‘if’ statement with calculated boundaries the code should be able to split the image where the lowest number of pixels are present.

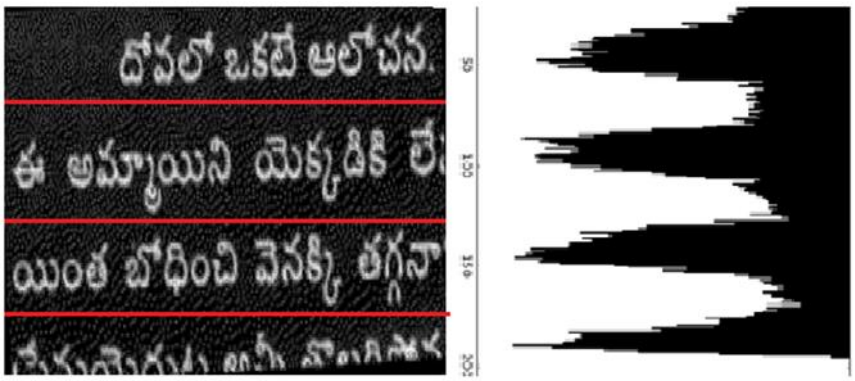

Figure 12, Written lines of text segmented using Horizontal Projection (23)

For the segmentation of words, a similar method can be used, instead of horizontal this time the method is vertical. The same steps are involved with the binarized matrix however the image matrix is now swept in the horizontal direction. The value in the columns is summated and assigned to the corresponding x value and this can produce another histogram for the vertical projection. As shown in Figure 13 below the words have segmentation between them, it is also visible that the histogram plot shows the gaps between the letters and therefore this method can also be used for letter segmentation. By instating boundaries within the ‘if’ statement in the code it is possible to only segment between the words where the larger gaps are present.

Figure 13, Written words that are segmented using Vertical Projection (23)

Finally, is the means of segmenting the letters. This can be done using a built-in function within MATLAB called ‘regionprop’, this function takes the input image that

needs to be inverted to white text on a black background where the white pixels are ‘0’ and the black are ‘1’. The code then finds the first ‘0’ in the matrix and calculates the distance until the last ‘0’ that is placed consecutively. A similar procedure then occurs in the vertical direction where it finds the lowest point where a ‘0’ is present and then calculates the distance in pixels until the last consecutive ‘0’. Using the initial points that it finds and the distances the code is then able to generate a parameter that called a ‘bounding box’ which contains an image of only the segmented letter.

Investigating these methods of image segmentation will allow for simpler images to be input into the network and should increase the recognition accuracy of the images. All of them however will require some image pre-processing first that reduces the noise in the images and then converts them to the correct colour format (i.e., black on white or white on black).

Output Possibilities (KWOK)

Given that this is a handwritten recognition model, there are several scenarios of which different outputs can be utilised in. The system will successfully recognise and identify an image of a handwritten word which is then outputted as a column of letters in text format.

The column of letters is transposed and presented in a row of text. This can be further implemented into a text to speech function with the identified text output as audio whereby designed to aid the visually impaired. Alternatively, the system can be further developed to include word segmentation whereby the chosen input is an image which contains a block of text instead of a singular handwritten word. It will still carry out the letter segmentation only after the word segmentation function has been carried out and after identifying the image and the words contained within it, should it then output the words in text format in a separate pdf or word file format.

This advanced version of the handwritten recognition system can be used to help in the identification of handwritten texts and literatures by students in school for the purpose of marking exam papers in situations where perhaps the legibility of text has become an issue. Or perhaps it may be used in the aid in identifying plagiarism between 2 handwritten literatures. Given that there is a digital input such as someone writing on a tablet, the system will then take the image and perform the necessary segmentation, then processing it into an exported word text file.

Code Design and Implementation (PJETRI, RUHID)

Load Initial Parameters

```
trainingSetup = load("E:\Layer Info\final_code_parrams.mat");
```

Import Data

```
imdsTrain = imageDatastore("E:\Datasets\Binary
Large","IncludeSubfolders",true,"LabelSource","foldernames");
[imdsTrain, imdsValidation] = splitEachLabel(imdsTrain,0.85,"randomized");

augimdsTrain = augmentedImageDatastore([30 30 1],imdsTrain);
augimdsValidation = augmentedImageDatastore([30 30 1],imdsValidation);
```

Set Training Options

```
opts = trainingOptions("sgdm",...
    "ExecutionEnvironment","auto",...
    "InitialLearnRate",0.01,...
    "MaxEpochs",16,...
    "MiniBatchSize",64,...
    "Shuffle","every-epoch",...
    "ValidationFrequency",30,...
    "Plots","training-progress",...
    "ValidationData",augimdsValidation);
```

Create Array of Layers

```
layers = [
    imageInputLayer([30 30 1],"Name","imageinput")
    convolution2dLayer([3 3],27,"Name","conv_1","Padding","same")
    reluLayer("Name","relu_1")
    maxPooling2dLayer([3 3],"Name","maxpool_1","Padding","same")
    convolution2dLayer([3 3],6,"Name","conv_2","Padding","same")
    reluLayer("Name","relu_2")
    maxPooling2dLayer([3 3],"Name","maxpool_2","Padding","same")
    dropoutLayer(0.5,"Name","dropout")
    fullyConnectedLayer(26,"Name","fc")
    softmaxLayer("Name","softmax")
    classificationLayer("Name","classoutput")];
```

Train Network

```
[net, traininfo] = trainNetwork(augimdsTrain,layers,opts);
```

Letter Segmenting

```
delete('Functions\PicOutput\*.png')
str = 'Functions\bench.png';
[A,M] = imread(str,"BackgroundColor",1);
I = ind2gray(A,M);
BW = im2bw(I, 0.8);
BW = ~BW;


stats = regionprops(BW);
    for index=1:length(stats)
        if stats(index).Area > 200 &&
stats(index).BoundingBox(3)*stats(index).BoundingBox(4) < 120000
            x = ceil(stats(index).BoundingBox(1));
            y= ceil(stats(index).BoundingBox(2));
            widthX = floor(stats(index).BoundingBox(3)-1);
            widthY = floor(stats(index).BoundingBox(4)-1);
            subimage(index) = {BW(y:y+widthY,x:x+widthX,:)};
```

```
            FILENAME = string(strcat('Functions\PicOutput\',
sprintf('%d.png',index)));
            imwrite(subimage{index},FILENAME)
        end
    end
```

Network Test

```
delete('Functions\TestLetters\*.png')
subimages = imageDatastore("Functions\PicOutput\*.png");
value = length(subimages.Files);
for num =1:value
    img = imread(subimages.Files{num});
    inv = imcomplement(img);
    pad = padarray(inv,[100 100],1,'both');
    figure, imshow(pad)
    IMNAME = string(strcat('Functions\TestLetters\',
sprintf('%d.png',num)));
imwrite(pad,IMNAME)
end

imageSize = [30 30 1]

PredImStore = imageDatastore("Functions\TestLetters\");
augimdsPred = augmentedImageDatastore(imageSize,PredImStore);

pred = classify(net,augimdsPred)

numImages = length(augimdsPred.Files)

ibx = randperm(numel(augimdsPred.Files),numImages);
figure
tiledlayout("flow")
for w = 1:numImages
    nexttile
    imshow(augimdsPred.Files{ibx(w)});
    title("Predicted Label: " + string(pred(ibx(w))))
end
```

Network Validation

```
YPred = classify(net,augimdsValidation);
YValidation = imdsValidation.Labels;
accuracy = sum(YPred == YValidation)/numel(YValidation)
plotconfusion(YValidation,YPred)
```

Output

```
T = transpose(pred);
disp(T)
```

```
fileID = fopen('Functions\OutputText.txt','w');
fprintf(fileID,'%c',T);
pause(20);
open("Functions\OutputText.txt")
```

Results and Discussion

Network Architecture (RUHID)

To gain inspiration for possible layer structures, the model Squeeze-net was trained using the dataset acquired for this report. This Squeeze-net adaption led to a validation accuracy of 56% which took 50 minutes to complete. For a first attempt, this result was acceptable as a starting guideline, however due to the mid-level accuracy and length of computation, the exact structure was not suitable for the classification of the dataset. An identical method was applied to the model, Darknet-19. This resulted in an accuracy of 90% taking 66 minutes to finish the training. While this model generated a more acceptable accuracy, the computational time was too high and would be concluded as an ineffective neural network. Both models utilise the convolutional, ReLu, and max-pooling layers as their main structure with either batch-normalization or concatenation layer as an alteration. Both pre-trained models as well as published articles include a fully connected layer followed by SoftMax layer for probabilities and classification layer since the task is classifying English characters.

Building on from the pre-trained networks, the first created neural network is shown in Figure 14. This architecture follows a similar pattern to the Darknet-19 model where a group of layers are repeated but, in this case, only a few times resulting in 16 layers and 85.7 million learnables. A key difference between the Darknet-19 and Figure 14, Layer Structure for Initial Neural Network is the use of ELU activation function. By conducting premature trials, the ELU layer was found to give a higher accuracy than when using ReLu or leaky ReLu for the layer configurations in Figure 14. The parameters for the convolution layers were identical except that the number of filters were increasing from 16 to 128 in multiples of 2. Otherwise, all convolutional layers were set to a filter size of 3 by 3 and a stride of 2 by 2. The max-pooling values were kept constant throughout the network with parameters of 3 by 3 with a stride of 2 by 2. This neural network achieved an accuracy of 76.5% within 4 minutes of running time. This is the correct trajectory for a reliable neural network however achieving a higher accuracy within the same time frame is desirable.

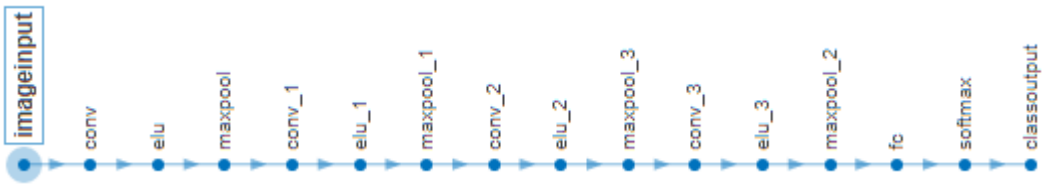

Figure 14, Layer Structure for Initial Neural Network

Improving on the initial neural network, the final neural network created can be seen in Figure 15. For the final model it was decided that to increase the accuracy, the complexity of the layers would need to be condensed. In other terms, the repetition of

grouped layers needed to be reduced. This model used a similar framework to the initial model, except the number of layers were decreased to 11 and it used ReLu activation function. This resulted to 142.3 thousand learnables. To ensure high accuracy in the validation, correct parameters for weights and biases were used for the layers.

One of the variations from the initial model was to include an image input size of 30 by 30 (900 pixels) compared to the original 227 by 227 (51,529 pixels). This was implemented to reduce the training time as well as to reduce the number of layers necessary. MATLAB prefers to compute smaller images as it requires less computational power. Another reason for choosing a 30x30 image size was to ensure the exact mapping of the filters from corresponding layers such as max pooling.

The next layer used is the convolutional, set to a filter size of 3x3, a stride of 1x1 and 27 filters. This layer essentially means that it condenses the image size by placing the filters along the image matrix. This leads to a convolution layer image output size of 27x27 pixels. This information is fed into the ReLu activation layer which replaces all negative pixel values to 0 and keeps all the positive values. The importance of ReLu is that MATLAB favours binarized data for images because those image matrices are easier to compute. It is also important to note that the ReLu layer does not affect the matrix size of the image.

The corresponding layer shown in Figure 15 is the max-pooling which was set to a filter size of 3x3 and a stride of 1x1. The max-pooling layer is efficient at reducing the size of the image as it essentially divides the current image size by the pooling filter size, i.e., the 27x27 image matrix retrieved from the first ReLu is divided or “pooled” together by the filter size 3x3. The first max-pooling layer then outputs the classification information of a 9x9 image to the second convolution layer.

The following convolution layer has identical filter size and stride as the previous convolution but with only 6 filters. After the image has been convoluted, the remaining image size is a 6x6 image matrix which is fed into the ReLu. From the ReLu, the information is passed to the max pooling layer which has the same parameters as its counterpart. By applying a 3x3 pooling to a 6x6 image matrix, the final working image size is a 2x2 pixel image. This smaller image size indicates that the neural network has classified all possibilities for the prediction of the image which should lead to and accurate and efficient image classification. The final layer used is the dropout layer set to the default (0.5) dropout rate. The purpose of this layer is to ensure there is no overfitting in the training, achieving the best possible version of a handwritten neural network.

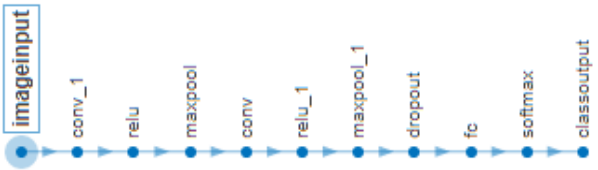

Figure 15: Final Layer Structure for Handwriting Recognition

Training and Validation (PJETRI)

For the initial network architecture shown in Figure 14 the training was run using the ‘adam’ solver method as opposed to the more popular ‘sgdm’. Additionally, the number of max epochs was set to 15 as this was when the model stopped learning and beyond this computational power was wasted. Some of the learning parameters can be seen in Figure 18, part (A) shows that model was able to run 2145 iterations within 4 minutes and 20 seconds. However as shown by the graph in Figure 16 the initial model has a lot of fluctuation within training as well as demonstrating overfitting. Due to this, the model was amended, and the aspects of the training were changed.

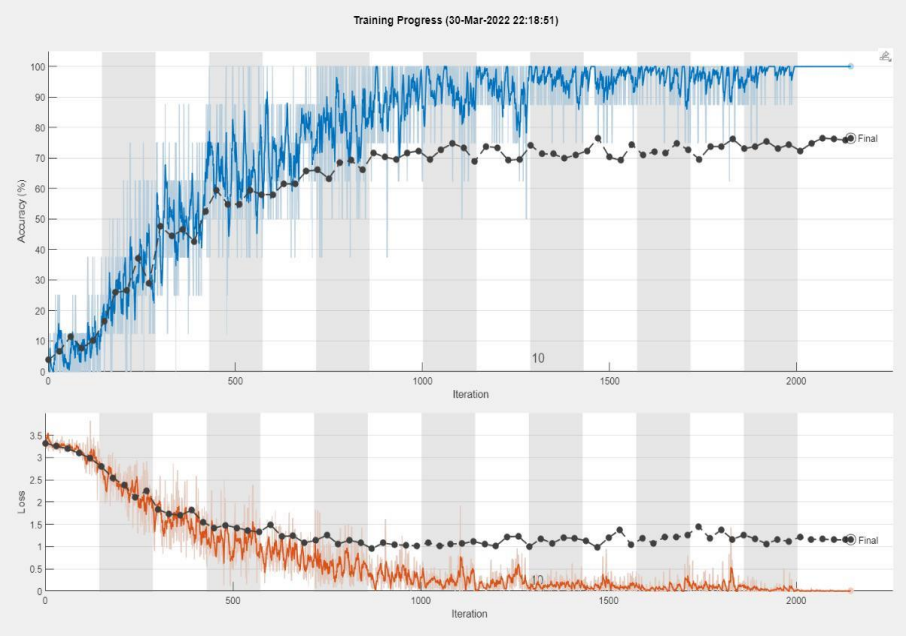

Figure 16, Training Model for Initial Network Design

Adaptions to the initial network architecture were made as outlined in the section above. Additionally, there were further steps taken to reduce the problem of overfitting as well as adaptions to the dataset. The original dataset sourced from Kaggle had a series of augmentation methods applied to it. There was a random rotation between -15 and 15 degrees as well as random rescaling between 0.7 and 1.3 and translation in the x and y direction by 30 pixels each way. As well as this, there were changes in the parameters used for the training, the biggest one being the use of the ‘sgdm’ algorithm for computation. This helped to decrease the fluctuation in the model as well as improving the computation time over using ‘adam’. As shown in Figure 17, the final network generated had a much smoother learning curve as well as a reduction in the validation loss. Shown in Figure 18 (B) are the training results for the final model and overall, it achieved a validation accuracy of 88.7% and this was an increase of over 12% on the initial trained network. Also, it was seen to have almost eliminated the problem of overfitting.

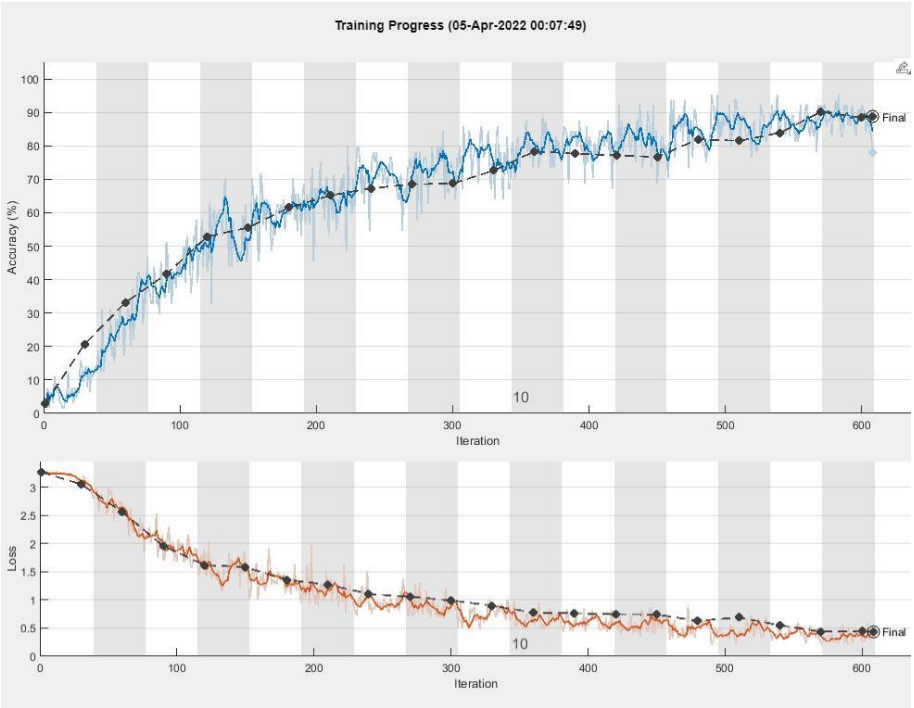

Figure 17,Training Model for Final Network Design

A

Results	
Validation accuracy:	76.57%
Training finished:	Max epochs completed
Training Time	
Start time:	30-Mar-2022 22:18:51
Elapsed time:	4 min 20 sec
Training Cycle	
Epoch:	15 of 15
Iteration:	2145 of 2145
Iterations per epoch:	143
Maximum iterations:	2145
Validation	
Frequency:	30 iterations
Other Information	
Hardware resource:	Single CPU
Learning rate schedule:	Constant
Learning rate:	0.001

B

Results	
Validation accuracy:	88.70%
Training finished:	Max epochs completed
Training Time	
Start time:	05-Apr-2022 00:07:49
Elapsed time:	4 min 10 sec
Training Cycle	
Epoch:	16 of 16
Iteration:	608 of 608
Iterations per epoch:	38
Maximum iterations:	608
Validation	
Frequency:	30 iterations
Other Information	
Hardware resource:	Single CPU
Learning rate schedule:	Constant
Learning rate:	0.01

Figure 18, Training Details for Initial Network (A) and Final Network (B)

Shown in Figure 19 below is the confusion matrix for the final network model This demonstrates the classes that the model was successful in recognising as well as the occasions where there was mis-prediction. In general, it was able to identify most characters correctly however there were two occasions in which it identified a ‘p’ and a ‘u’ as a ‘b’. These mistakes are believed to have occurred because of the similarities in shape between the letters. As a ‘p’ has a round ball type top and a thin line protruding from it, essentially it can be categorised as a ‘b’ reflected in the x axis. The ‘u’ however is less obvious as to why the mis-prediction occurs, the most fitting theory is that it is due to the curved nature at the bottom of the letter. Furthermore, the image that was recognised incorrectly could have portrayed qualities of a ‘b’ such as being a

closed ‘u’ rather than being opened at the top. The character that was recognised most incorrectly was ‘c’ and this was mistaken for an ‘e’ twice as well as an ‘r’. These are understandable as sometimes even by human eye, if there is no context to the letters, they are easily mistaken in the same way.

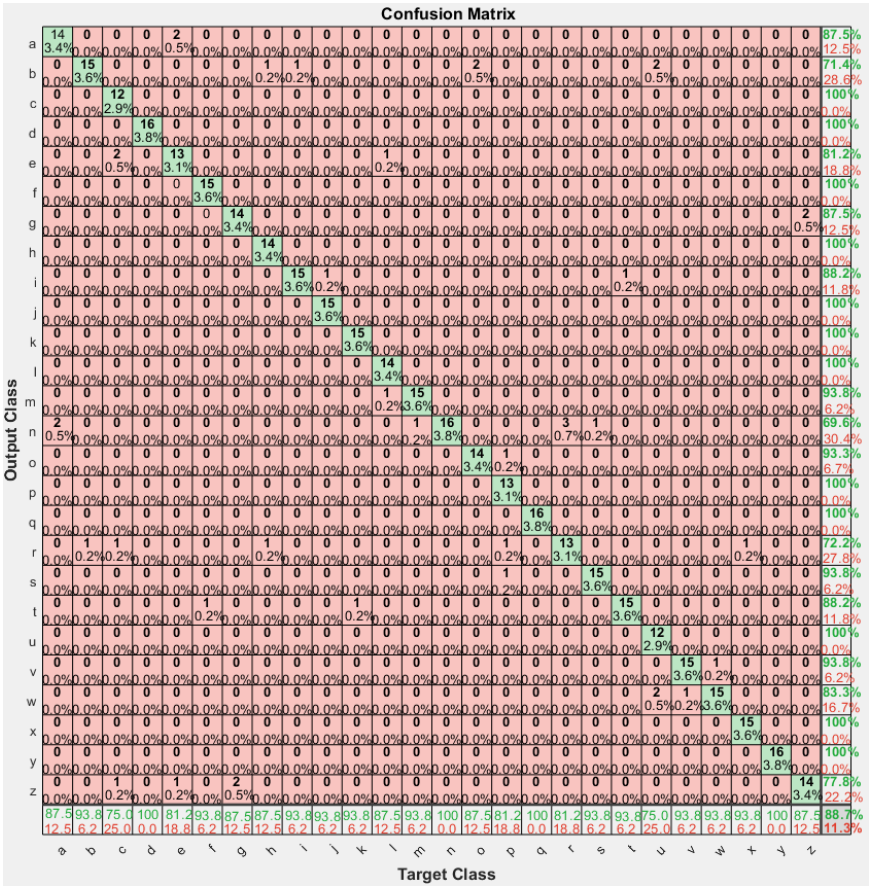

Figure 19, Final Network Confusion Matrix

Segmentation and Output (AMIN)

Figure 20: Test Word 'Spring' with Black Background

Figure 21: The Segmented Word 'Spring'

Figure 20 shows the handwritten word 'spring' used for the segmentation process. This word has a similar gap between the characters which will assist in the character segmentation. The method proposed is not suitable for cursive writing where each letter has no discernible pixel gap.

As shown in the letter segmenting code, the parameters are set so that if the area of the word is more than 200 pixels and less than 120000 pixels, the code will detect the word for segmentation. These parameters are necessary because of the potential large sizes of the inputted image.

The utilised parameters helped identify a wider length variation and different writing styles. Figure 21 shows the segmented character results for the word 'spring', emphasising that the approach method has achieved its purpose of splitting the letters. Furthermore, as the dataset used to train the network was black text on white background the outputted letters were reverted to this format to increase the recognition accuracy. The process was completed successfully, and all the grey and colour pixels were converted to black and white. Then, the images were saved into the 'PicOutput' folder to be ready for testing.

The segmented characters were saved as numerical labels to ensure that the new images were stored correctly when the words were tested each time. Thus, the labels for the segmented characters are saved as the current index of the 'for loop'. Moreover, a delete line was used in letter segmentation which removes the contents of the folder 'PicOutput' before the segmentation code is executed. After testing the characters using the output code, the prediction results are shown in Figure 22. This figure demonstrates that the network managed to classify 66.6% of the characters accurately. It is possible that due to the augmentation of the training dataset, some characters are being classified incorrectly. Another possible reason is the lack of padding on the segmented characters.

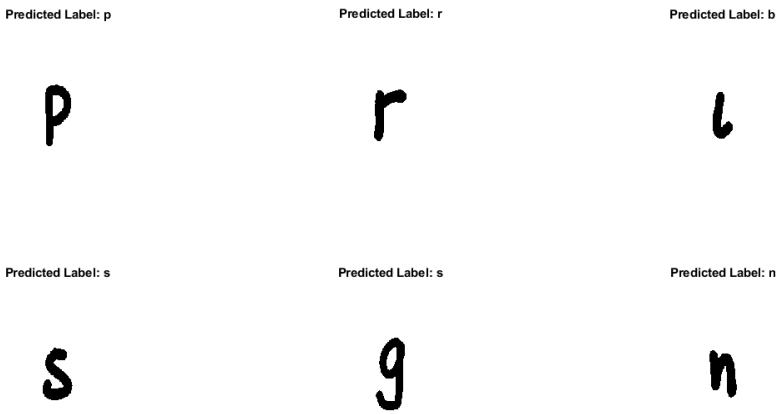

Figure 22: Image Classification for the word 'spring'

To increase the reliability of the classification part, the testing code was altered to include padding to the segmented characters. Previously the code separated the characters with the first and last pixel in both x and y intercepts of the perimeter of the 'boundingbox'. With the addition of the padding code, the word 'bench' shown in Figure 23 was used for the testing. The segmented output is shown in Figure 24. Figure 25 depicts the prediction of the word which shows that the neural network managed to achieve a 100%-word accuracy. It is important to note that not all words will achieve the same accuracy since some characters have been mis-identified as shown by the confusion plot. The output code above illustrates how the process compares each scanned letter pixel by pixel, and then decides on the closest match. Afterwards, the code was used to print out the prediction letters in a text file, and the word was successfully displayed. This concludes that the segmentation and output processes was executed accurately and achieved its purpose of separating the characters and predicting accurate handwriting recognition.

Figure 23, 'bench' with inverted colours

Figure 24, 'bench' segmented into individual letters with padding

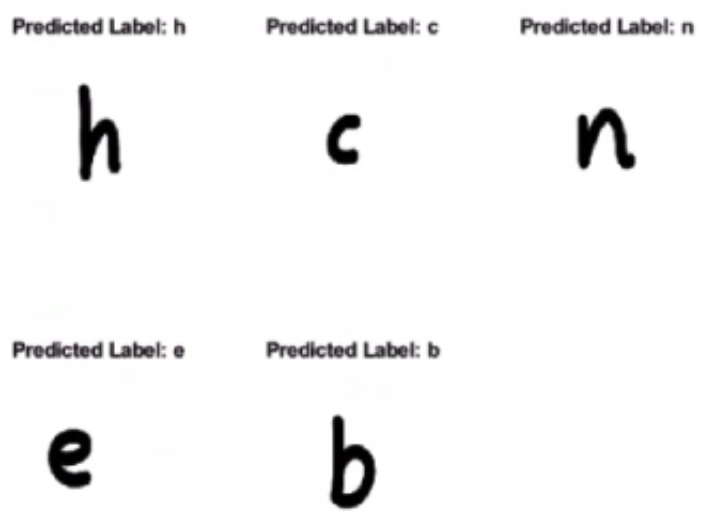

Figure 25, prediction of the word bench with padding

Conclusion (KWOK)

The final iteration of the handwritten recognition model was shown to be well fit whereby the training accuracy is observed to be averaging similar values to the validation accuracy. A test was performed with the inputted image containing the handwritten word “bench” and was able to obtain a 100% accuracy in identifying it after padding was applied to the sub images. Overall, the system achieved an 88.7% validation accuracy in the identification of individual characters. Through carrying these processes successfully, the code written for this recognition system was able to output the text contained within a given image in a text-file format automatically.

Future development into this system may include padding to the character segmentation to achieve a greater validation accuracy and more reliable recognition system. Other segmentation methods can also be utilised to further advance and complexify the network. Such may include the use of vertical projection or the adjustment of the “regionprops” values which allows segmentation of words in sentences; horizontal projection may also be implemented to segment sentences within paragraphs from handwritten literatures. Functions may also be put in place to output the text differently such as a text to speech function which directly converts the identified word image into an audio file to aid people who are visually impaired.

References

1. **Foote, Keith D.** A Brief History of Deep Learning. *Dataversity.* [Online] 07 February 2017. [Cited: 04 March 2022.] https://www.dataversity.net/brief-history-deep-learning/.

2. **Baheti, Pragati.** 12 Types of Neural Network Activation Functions: How to Choose? *V7 Labs.* [Online] 10 January 2022. [Cited: 04 March 2022.] https://www.v7labs.com/blog/neural-networks-activation-functions.

3. **Kostadinov, Simeon.** Understanding Backpropagation Algorithm. *Medium.* [Online] 08 August 2019. [Cited: 04 March 2022.] https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd.

4. **Pedamkar, Priya.** Machine Learning Datasets. *Educba.* [Online] 2020. [Cited: 04 March 2022.] https://www.educba.com/machine-learning-datasets/.

5. **Sydorenko, Iryna.** What Is a Dataset in Machine Learning: Sources, Features, Analysis. *Label Your Data.* [Online] 05 April 2021. [Cited: 04 March 2022.] https://labelyourdata.com/articles/what-is-dataset-in-machine-learning#:~:text=A%20dataset%20in%20machine%20learning%20is%2C%20quite%20simply%2C%20a%20collection,same%20way%20as%20humans%20do..

6. *The State of the Art in On-Line Handwriting Recognition.* **Tappert, Charles C., Suen, Ching Y. and Wakahara, Toru.** 1990, Vol. 12. 10.1109/34.57669.

7. *On-Line and Off-Line Handwriting Recognition: A Comprehensive Survey.* **Plamondon, Rejean and Srihari, Sargur N.** s.l. : IEEE, 2000, Vol. 22. 10.1109/34.824821.

8. *A Novel Connectionist System for Unconstrained Handwriting Recognition.* **Graves, Alex, et al.** s.l. : IEEE, 2009, Vol. 31. 10.1109/TPAMI.2008.137.

9. *Deep Structured Output Learning for Unconstrained Text Recognition.* **Jaderberg, Max, et al.** s.l. : Cornell University, 2014. arXiv:1412.5903 .

10. **Niklas, Donges.** A Guide to RNN: Understanding Recurrent Neural Networks and LSTM Networks. *BuiltIn.com.* [Online] Built In, 17 08 2021. [Cited: 04 03 2022.] https://builtin.com/data-science/recurrent-neural-networks-and-lstm..

11. **Yang, Dr QingPing.** *Recurrent Neural Networks (RNNs).* London : Brunel University, 2022.

12. **Saha, Sumit.** A Comprehensive Guide to Convulutional Neural Networks. *Towards Data Science.* [Online] Towards Data Science, 15 12 2018. [Cited: 08 03 2022.] https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53..

13. **Yang, Dr QingPing.** *CNNs.* London : Brunel University, 2022.

14. **Baldominos, Alejandro, Saez, Yago and Isasi, Pedro.** *Evolutionary Convoutional Nueral Networks: An application to handwriting recognition.* Madrid : Elsevier, 2017.

15. **Matcha, Anil Chandra Naidu.** How to easily do Handwriting recognition Using Deep Learning. *NanoNets.* [Online] N/A N/A N/A. https://nanonets.com/blog/handwritten-character-recognition/.

16. **Rajib Ghosh, Chinmaya Panda, Prabhat Kumar.** *Handwritten Text Recognition in Bank Cheques.* Patna : CICT, 2018.

17. **Deniz Engin, Alperen Kantarci, Sec¸il Arslan, Hazım Kemal Ekenel.** *Offline Signature Verification on Real-World Documents.*

18. **Verma, B., Blumenstein, M. and Kulkarni, S. .** *Recent achievements in off-line handwriting recognition. .* Queensland : Academia, 1998.

19. *Offline Arabic Handwriting Recognition: A Survey.* **Lorigo, L. M. and Govindaraju, V.** s.l. : IEEE Transactions on pattern analysis and machine intelligence, 2006, Vol. 28. 1939-3539.

20. **K. S. Nathan, H. S. M. Beigi, Jayashree Subrahmonia, G. J. Clary and H. Maruyama.** *Real-time on-line unconstrained handwriting recognition using statistical methods,.* Detroit, MI, USA : IEEE, 1995. 1520-6149.

21. **Ambalina, Limarc.** 15 Best OCR & Handwriting Datasets for Machine Learning. *LinkedIn.* [Online] 2022. https://www.linkedin.com/pulse/15-best-ocr-handwriting-datasets-machine-learning-limarc-ambalina/.

22. **Dsouza, Jason.** What is a GPU and do you need one in Deep Learning? *Towards Data Science.* [Online] 2022. https://towardsdatascience.com/what-is-a-gpu-and-do-you-need-one-in-deep-learning-718b9597aa0d.

23. **Reddy, Susmith.** Segmentation in OCR. *Towards Data Science.* [Online] Towards Data Science, 25 03 2019. [Cited: 05 04 2022.] https://towardsdatascience.com/segmentation-in-ocr-10de176cf373.