**HANDWRITTEN CHARACTER RECOGNITION**

**Ng Chung Hou**

**26436442**

**Turki Baghlaf**

**26226960**

**FIT3081 Image Processing**

**Implementation Project**

**Semester 1, Year 2019**

**School of IT**

**Monash University Malaysia**

**ABSTRACT**

**Give a brief description about your project, the results you have obtained and concluding remarks.**

**Table of Contents**

[**1.0 Introduction**](#_1q8tpo2dy7g2) **4**

[1.1 Background](#_sqt8fjvo9k5y) 4

[1.2 Aims & Motivation](#_anggtfrdtpn6) 5

[**2.0 Literature Review**](#_acw4zz2rwowx) **6**

[2.1 Online HCR](#_kad59vghuc7a) 7

[2.2 Offline HCR](#_kb3gz5tlo8v3) 9

[**3.0 Design and Methodology**](#_4xiua7ogbom) **12**

[3.1 Preprocessing](#_bzf4whg3vnv) 12

[3.1.1 Initial input](#_gupa0k22fyry) 12

[3.1.2 Thickening and Closing](#_urhzelfdh5o) 13

[3.1.3 Rotation and reorientation](#_6lku8ddpeqnf) 14

[3.1.4 Skew Correction](#_laaitcl61gv) 15

[3.2 Segmentation](#_110up6its8ux) 16

[3.3 Training and Recognition](#_10k9hbd9vjtr) 17

[**4.0 Implementation Details**](#_25oi5hj8wnxd) **21**

[4.1 preparing the dataset](#_byains5az0dg) 22

[4.2 Training the network](#_8bxkbbpzy8fm) 23

[4.3 CharacterRecognitionTest](#_l8t8bsaislbm) 25

[**5.0 Results and Discussion**](#_8kj10ti3qaxr) **30**

[**6.0 Limitations**](#_ph78it4bulq) **33**

[**7.0 Conclusion**](#_xwhz6h4jlrug) **33**

[**8.0 References**](#_t4orp3jfz8h) **33**

# 1.0 Introduction

Handwriting recognition has been a problem since we are able to feed in clear images into a computer. The methodologies used for handwriting recognition has evolved due to the increase in computational power available, and the development with machine learning, that spurred the demand for a effective and accurate algorithm for handwriting recognition.

## 1.1 Background

Back in the 1940’s the first kind of character recognizer utilised template matching technology which includes a library of characters to compare it from. The technology used back then were simple image matching, effective but it only recognise clear and distinct handwriting, or machine printed characters. This is the first form of off- line recognition technology, where the image is given and the characters are recognised after processing.

In the 1950’s the first electronic tablet came out, along with the first commercial character recogniser. This particular technology uses on-line recognition, meaning it will record the x-y coordinates of the user’s input instantly and make predictions using that. That is the first form of on-line recognition, where the recognition attempt is made while input is continuously feeding into the algorithm.

During the 1980’s and 1990’s more powerful computers and the wide availability of it created a rapid growth for character recognition technology. Now the research is focused on shape recognition that does not rely on a large library of images. Having the availability of higher resolution images also increased the requirement for more powerful computers as those images contained more data for the computer to crunch. Furthermore the shape recognition technique was not sufficient for practical applications as well.

In modern times, with more extensive and complex recognition algorithms that utilised image processing techniques as well as Artificial Intelligence methodologies have dramatically improved the efficiency and accuracy of technology. The advancement on neural network as well has opened a whole new opportunity, utilising machine learning to teach the recognition technology to identify all sorts of characters, from weird fonts to unclear images. Utilising Image Processing techniques to clear up the image and segment the characters, then using machine learning to identify the character is what this project is going to do.

## 1.2 Aims & Motivation

The use of handwritten characters recognition technology would streamline a lot of menial tasks and advance all offices and businesses to the digital age. Imagine if there is a reliable handwritten character recognition technology that is reliable and affordable to all. Offices that contain physical logs can be digitised into databases in record time. Being able to immediately digitise a handwritten character could also help with identification technologies, Google Lens is the best example of this technology where it could scan a live video and identify words to scan for google results or quick translations to/from another language. With the improvement of handwritten character technology also brings about normal character recognition, so car plates can be more reliably scanned as well, or printed documents, or even speeding up the rate of marking exam papers by digitising it and passing it through an automated marker.

We intend to create a fully functional offline handwriting recognition technology, by simply passing an image with words in it, and will output the text that is in the image. We will be utilising Matlab to create a script that will have the functionality mentioned, Matlab is used because it contains valuable tool boxes such as the image processing toolbox and deep learning tool boxes that we will use for the script. The script will first enhance and reorient the image to segment the words clearly into characters, and then pass it to a neural network to identify the characters, then piece it back together output the word contained in the image.

# 2.0 Literature Review

Character Recognition is a broad area of research for algorithms and techniques that enables a computer program to detect and recognise printed or written characters that were converted into a digital form such as an image. Under this umbrella, there are the two branches, Optical Character Recognition (OCR) and Handwritten Character Recognition (HCR), which share a common goal of converting detected characters in an input image to a digital symbol in a computer, but slightly differ in the process to accommodate posed challenges.

In this report, HCR will be our focus as there are many factors that could affect the process of recognising handwritten characters compared to computer printed standardised characters. Few challenges that are faced when dealing with handwritten characters is the variety of ways which a character can be written as each individual has their own writing style and writing speed. Furthermore, their physical and mental condition could affect their writing style and character size (Poovizhi, 2014). The major steps in recognition of characters that has to be implemented to any character recognition system include preprocessing, segmentation, feature extraction and classification. HCR can be further classified to two types based on the data acquisition process. Below, the two branches of HCR will be discussed addressing the difference between them and recent contributions for optimization.

## 2.1 Online HCR

The data acquisition in an online HCR system involves a stylus which is used by the writer to write characters on an electronic surface such as a digitizer embedded to a tablet screen to obtain spatio-temporal information which contains the positions of traces of the stylus tip on the screen at each point in time,the total number of strokes and velocity of writing within each stroke. Instead of the traditionally preprocessing the obtained data, a simpler method was presented by researchers from the world academy of science to replace the preprocessing and later stages with more efficient set of processes as seen in figure below.

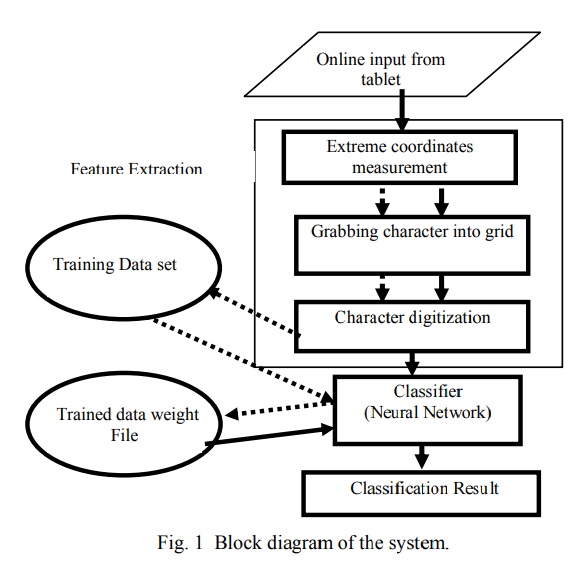


Figure 2.1.1

As illustrated in the block diagram of the proposed system, the next step after data acquisition is feature extraction, which involves finding the extreme coordinates in order to detect the characters by scanning pixels from each border towards the border parallel to it and assigning the first high pixel as the new border which detects where the character starts in each direction (top, bottom, left and right).

The next step is to scan the binary image for row gaps by iterating from top to bottom and stopping at each empty pixel to scan from left to right for that particular row. In case no trace of the character ( high pixel), the entire row is eliminated as it is considered a gap row. Then the system converts the character image into a grid with the appropriate size based on the dimensions of the character having certain amount of pixels in each grid as seen in the figure below and marks each grid found containing a high pixel (containing a part of the character pixels) as high and leaves other grids as low in a process known as character digitization.

The grid is then passed to a counter propagation neural network after being flattened to a 1 by n dimension where n is the number of grids in the extracted feature matrix. The system was able to correctly classify 94 percent of test written characters after being trained on 66 samples of each character with the threshold parameter of the network turned off (Zafar, Mohamad & Othman, 2007).

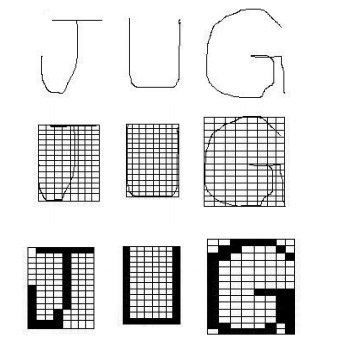


Figure 2.1.2

## 

## 2.2 Offline HCR

In an offline HCR, data acquisition involves transforming a physical document, that has characters already written to them, to a digital form via an optical instrument such as a scanner or a camera. The process is severely prone to noise due to interference of surrounding lights and the settings of the optical device, which could increase the variance in the character images especially when multiple optical devices were used for conversion or conversion had taken place in multiple locations with mismatching lighting temperature and intensity. Furthermore, offline handwritten character recognition operates on spacio-luminance data of images and lack the temporal features available in its online counterpart. Hence, the complexity of offline handwritten character recognition is more than its online counterpart due to being more prone to noise and the lack of data to perform stroke level recognition leading to a poorer accuracy of recognition (Sá, Alexandre, Duch & Mandic, 2007).

Recent work on offline character recognition mostly focus on the application of machine learning and deep learning techniques on the classification and recognition step. A recent experiment by (Paul et al., 2019) involved using Long Short Term Memory (LTSM) network to be trained and used to recognise handwritten characters. LTSM is an enhanced modification of the Recurrent Neural Network (RNN) architecture to overcome the vanishing gradient problem that occurs when RNN is used on a task with long-term dependencies (Bengio et al., 1994). The team used chars74k dataset of captured character images via phone cameras as seen in figure below, and proceeded to preprocess the images by converting it to grayscale, then used a specified threshold value of pixel value found in the histogram to binarize the images. Then, Morphological operations such as opening and closing were used to cure any disjoint characters by reconnecting the edges and to eliminate any noise, followed by rotation correction using the row histogram value.

For segmentation, a hierarchical segmentation method was followed, as the segmentation was performed on line, word and character, then zone wise modifier isolation technique was used to partition the character into zones (lower, mid and upper). Finally, the features of each character was extracted in binary code as illustrated in figure 2.2.1 the text sample figure, and passed to the LTSM network to train in figure 2.2.2.

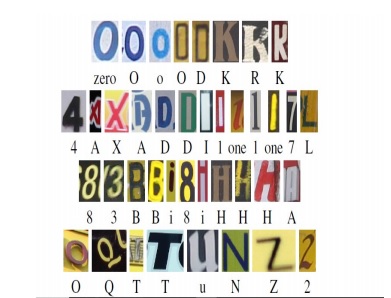


Figure 2.2.1



Figure 2.2.2

Using the abovementioned architecture, the team was able to achieve an accuracy of 88.32% by testing the system with handwritten character photos taken with phone cameras as shown in figure 2.2.3 and 2.2.4.

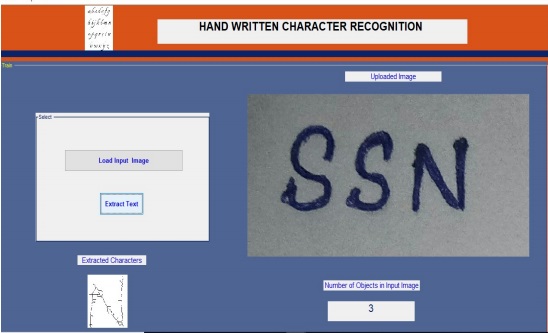


Figure 2.2.3

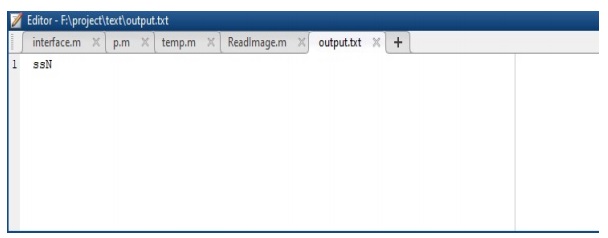


Figure 2.2.4

# 3.0 Design and Methodology

The aim of this project is to produce a MATLAB script that is a fully functional offline handwritten character recognition system, that will reliably convert an image with words into a digital text. Accuracy is of the utmost importance and we will aim to have an accuracy of above 80%. The section below we will detail the workflow of our system in figure 3.0.1:



Figure 3.0.1

## 3.1 Preprocessing

When the initial input image is passed in, there are multiple steps to ensure that the character is as clear and as legible for the network to recognise. The image might contain noise, blobs or even at the wrong rotation. So it is important to preprocess the image to its clearest form before segmentation to ensure proper segmentation of characters as well.

### 3.1.1 Initial input

When the image is read by the script, it first gets resized to ensure the consistency in the size of the image, as there might be different sized image taken by the camera. After the resizing the image will get turned into a grayscale image to perform adaptive thresholding. This will remove some of the noise in the image as well as enhance the contrast of the background and the words. Finally for the first phase of preprocessing, the image is binarised and the colors are flipped, this is because since we want the data represented as words to be a 1 instead of a 0 in an binarized image, we simply invert the image to have the words white and the background black. The transformation from a raw image after applying these steps can be seen in Figure 3.1.1.1:

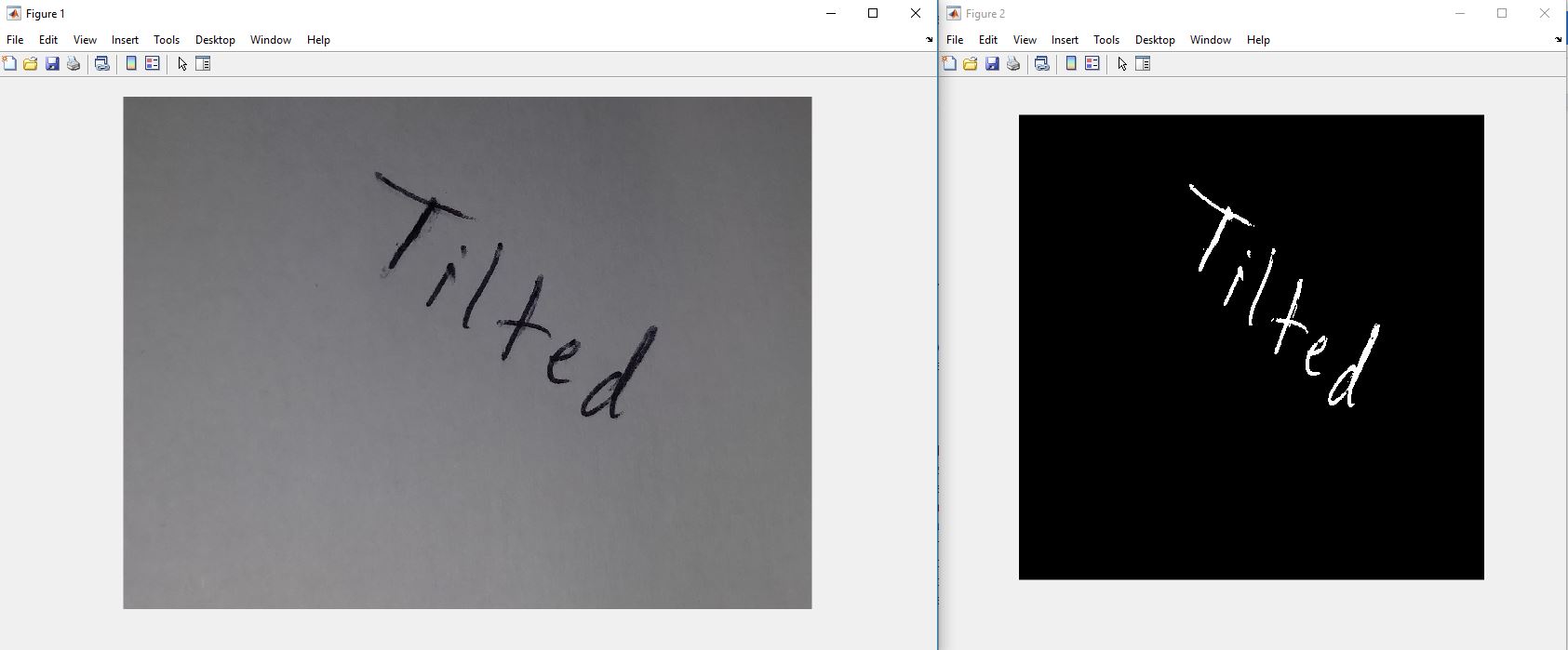


Figure 3.1.1.1

We can see that the noise are non existent in the output image, and the character do not have the smudges shown in the “T” of the first image. The image is also resized into a more manageable size as the initial image was fairly large (2068 x 1540 compared to 500 x 500).

### 3.1.2 Thickening and Closing

In certain scenarios, especially if the handwriting is less than optimal, there exists gaps even between the characters themselves. Since we are using regionprops to identify the characters, the characters in the image needs to be connected, so we need to morph the characters to ensure that the characters are connected. As seen in figure 3.1.2.1 below, the “y” and “k” is disconnected from each other



Figure 3.1.2.1

Now we would put the image and thicken it by using bwmorph function. To ensure that the gaps are now much closer as shown in the middle window of Figure 3.1.2.2.

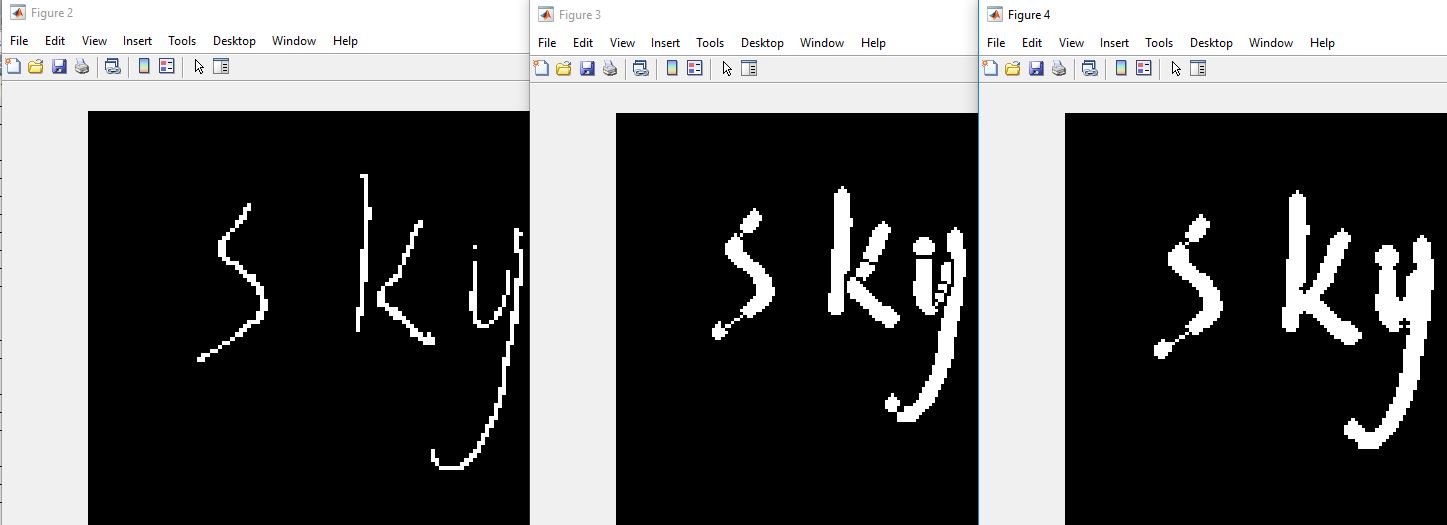


Figure 3.1.2.2

However after thickening, the gap still isn’t closed because of how thicken morph function in matlab prevents it from merging with other object, so utilising imclose to fill in the gaps where needed will ensure the characters are now considered as one connected blob

### 3.1.3 Rotation and reorientation

Images may also come with the words oriented deviating from the horizon, this would obviously cause problems for segmentation and recognition. So reorienting the image to ensure that it is readable is essential.

First the script has to determine the angle to which the words are off the horizon, this can be done by using regionprops Orientation extension, which can determine the angle of a particular image region. However since the word needs to be a single connected entity for it to determine the entire orientation of the word, simply closing the image with a large disk object will yield an “eclipse” that can easily identify the angle

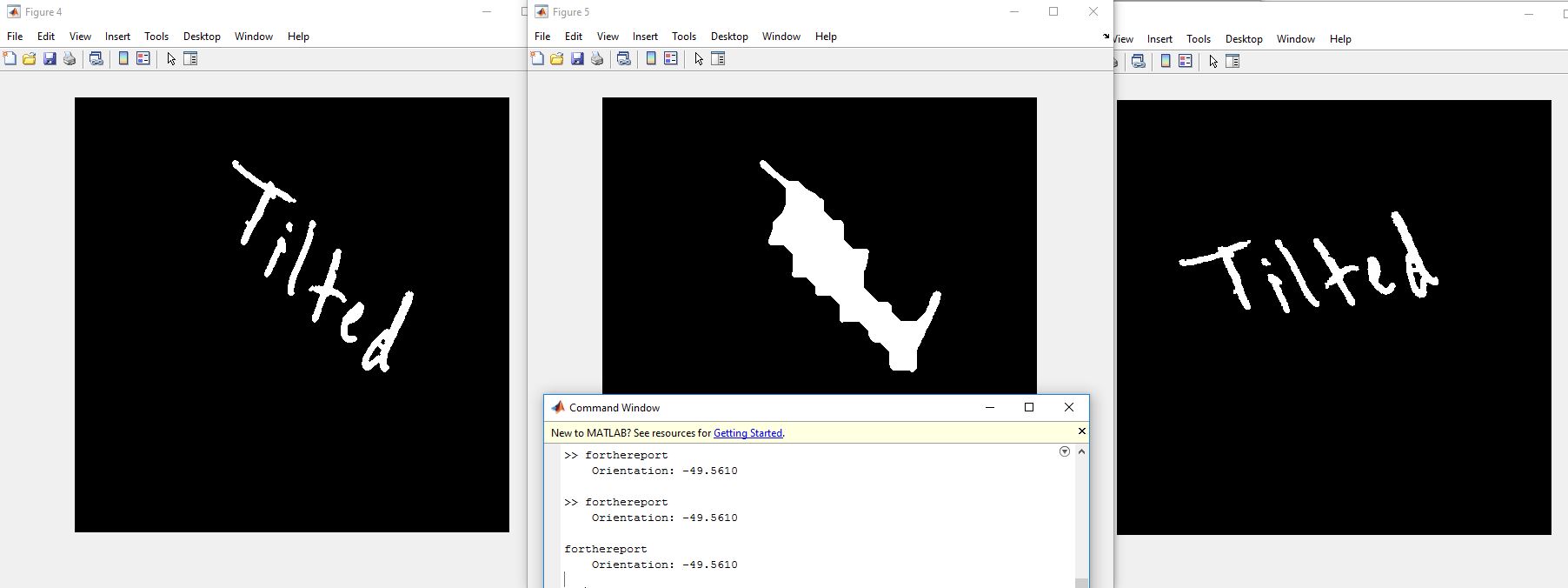


Figure 3.1.3.1

As you can see in the figure above, after converting the word into a connected blob for the regionprop Orientation function, it returns the angle and determines the word in Figure 3.1.3.1 to be -49 degrees off the horizon, simply rotating the negative of that value would yield an image that is orientated.

### 3.1.4 Skew Correction

When the image is rotated, or when the letters are written in cursive, the character can appear slanted, which can affect the efficacy of the recognition, therefore correction is needed to correct this. One thing to note is that this part is done while the image is being segmented to characters, because getting the angle of the word would give the orientation of the word, not the slantness of the individual character.

Using hough transform for the individual character and determining the hough peak would give the angle to which the character is off the vertical axis. Once the angle is determined, using shear transformation to correct the slantness of the character. Utilising trial and error we have figured out that 1 point in the shear transformation is equivalent to 2.2 degrees.

On final issue with the skew correction, is that since the image is cropped using a square boundary box, the skewed characters would overstep into the next characters bounding box, resulting in each cropped character image containing a distorted partial image of the next character. This is solved by simply removing any "blob" of pixels less than a certain value with the bwareaopen function, which can be considered as a stray stroke. The result of the slant correction can be seen below in figure 3.1.4.1

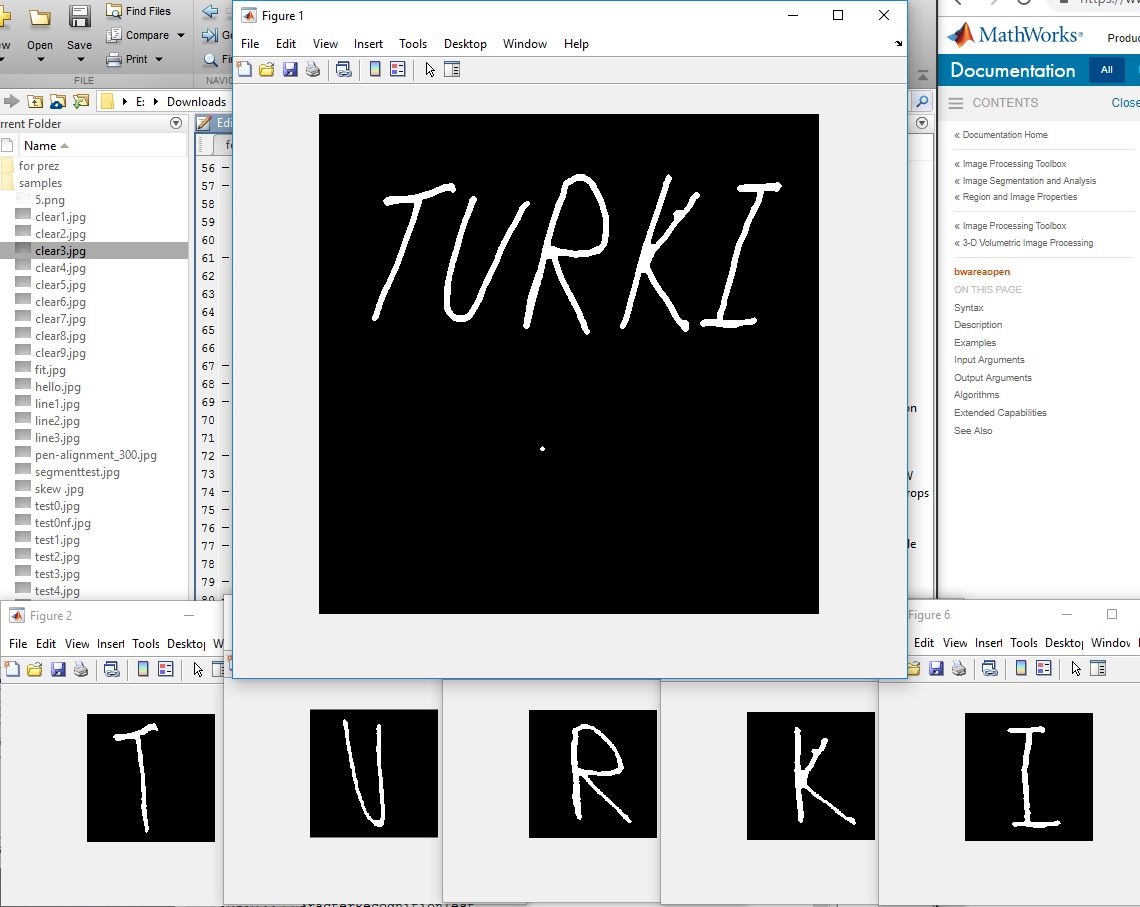


Figure 3.1.4.1

## 3.2 Segmentation

Once the image has been reorientated and preprocessed, segmentation can begin. For this script we utilised the in built regionprop function to determine the characters. Regionprops return all the regions of an image, that means in a binarized image it returns all the “blobs” of white pixel that exists within the image

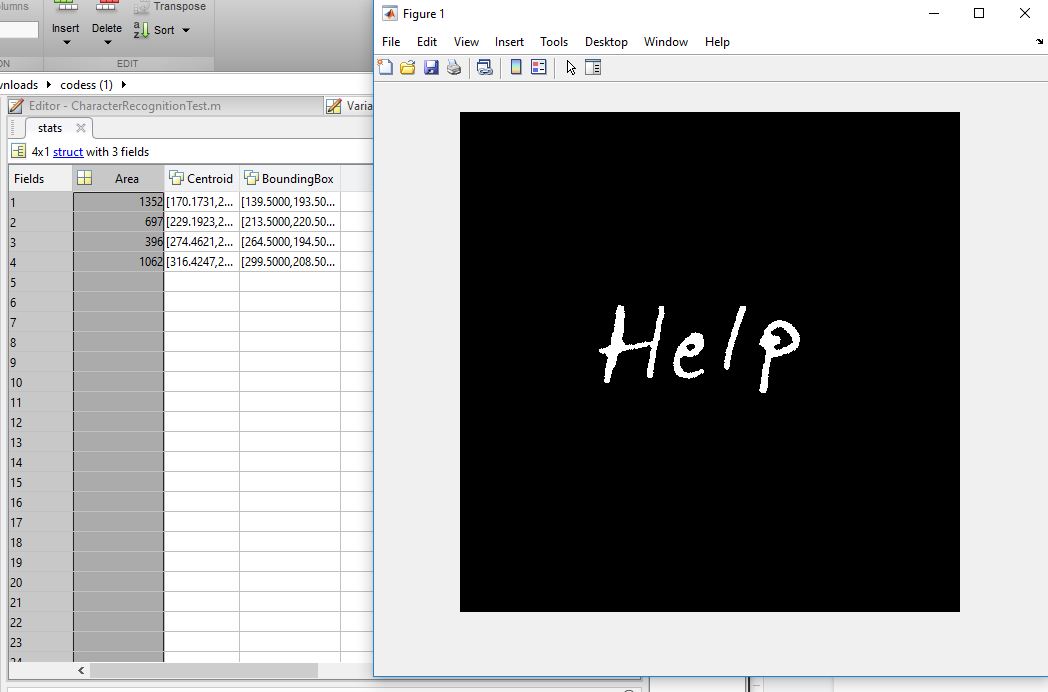


Figure 3.2.1

As you can see in figure 3.2.1, in the image only exist four characters, and regionprops can only identify four different “blobs” within the image. So in the left table within figure 3.2.1, there are four rows containing information of those blobs, the area of the blob, the centroid of the blob, and the bounding box of the blob.

With that information in mind, first thing to do is filtering out noise, noise tends to have a really small area so with a simple if-else statement, we are able to filter out all the noises and blobs that are too small to be considered a letter.

For the remaining blobs that are within the right size, utilizing the bounding box property we can get the coordinates of the particular character and crop it out

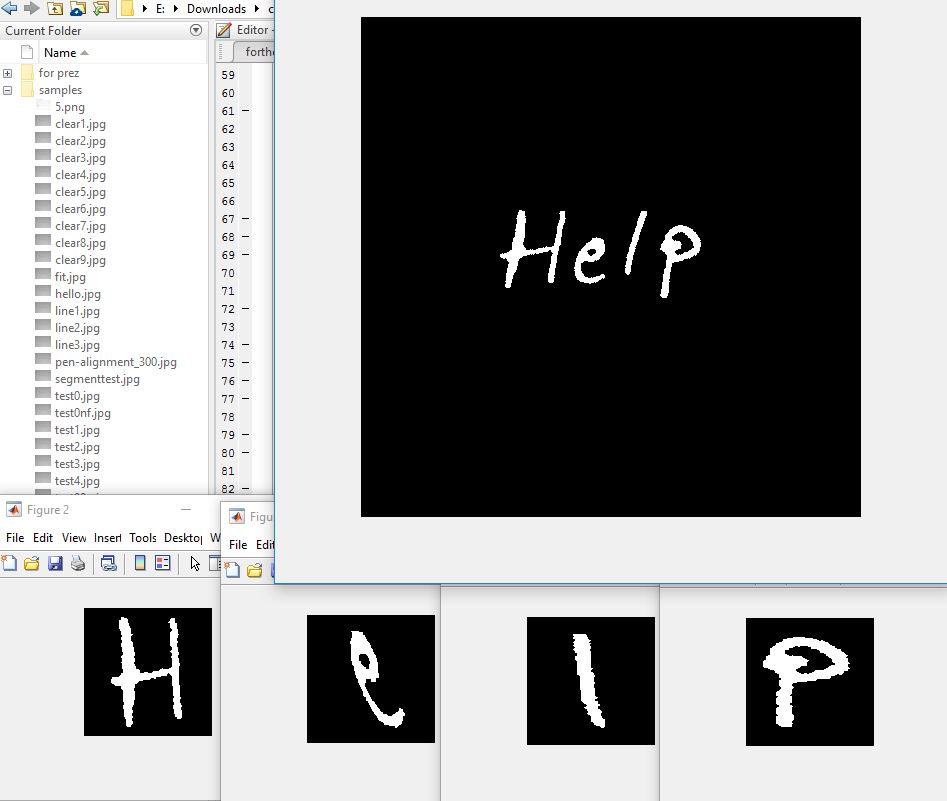


Figure 3.2.2

## 3.3 Training and Recognition

The dataset used for training is the NIST handprinted forms and characters obtained from Data.gov following this [link](https://catalog.data.gov/dataset/nist-handprinted-forms-and-characters-nist-special-database-19/resource/be00b652-093c-4ab6-8e0d-0edc8920913c). After extraction, the dataset consists of 62 (a-z, A-Z and 0-9) folders each containing samples of the same character as seen in Figure 3.3.1 below. The images size is standardized to (128x128x3) having the character stroke in black and padded with a white background.



Figure 3.3.1 raw NIST dataset images

Since the samples can get up to three thousand samples for some character and lesser for others, A code was written to form a new dataset by extracting the first thousand samples of each class with minor pre-processing such as grey scaling as both 3 channels had the same values which is redundant and complementing to turn the stroke traces to white and background to black was performed as seen in the figure 3.3.2 below.

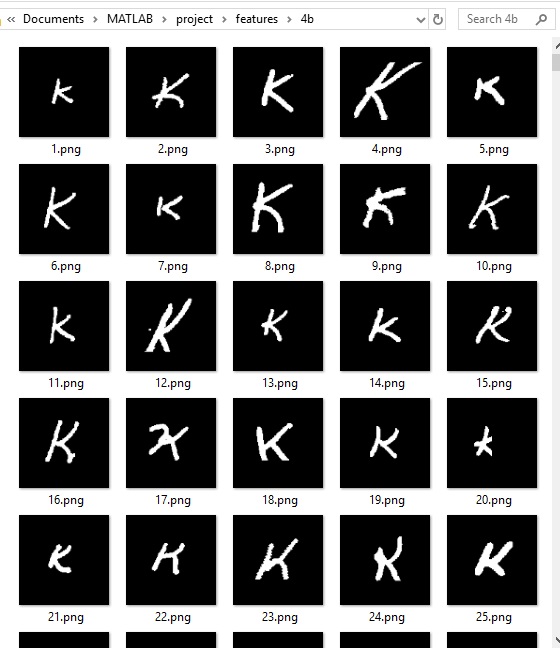


Figure 3.3.2 Processed NIST dataset images

After dataset preparation, the new dataset is loaded to be partitioned before passing to the neural network. The loaded dataset is divided into three partitions, seventy percent of the dataset is allocated to the training set, twenty percent to the validation set and ten percent to testing.Then the training and validation datasets are fed to the network to train. The network used for training and recognition is a Convolutional network as it was proven to be robust for visual object recognition and handwritten character recognition applications as it contains local receptive fields, weight sharing and spatial pooling features over other feed-forward neural networks.

The architecture design was chosen based the setup that yielded the best result during experimentation which will be discussed a later section, and it consists of the following layers. The input layer which is the first layer in the network, which role is to fit each pixel of the image to a static neuron to pass it to the hidden layers of the network in a process called feedforward during training. Another type found in CNN is a convolution layer, which has n number of filters of a specific size in which, are used to extract patterns found in the image by passing each filter on the image with a specific stride value to generate feature maps. Figure 3.3.3 below is an illustration of a filter of the size 5 passing on the image and matching its pattern with the image to extract each value forming a feature map.

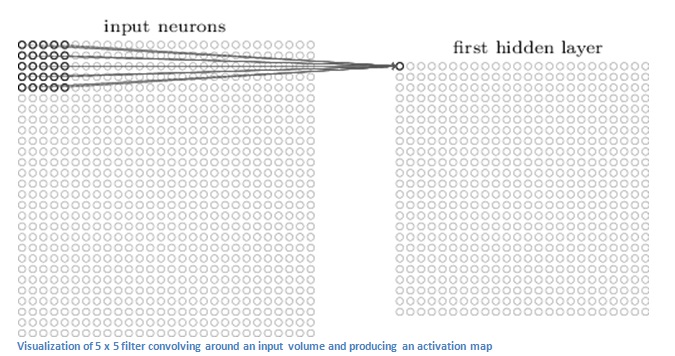


Figure 3.3.3 A Beginner’s Guide To Understand Convolutional Neural Networks, by Adit Deshpande, retrieved from<https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/>

At the final layers of a CNN, a fully connected layer (FC) with the size of the number of classifications is attached to the network to derive the probability for each class from the previous layer weights. Therefore, it is often followed by a Softmax layer (for multi-classification), to convert the weights of the last FC layer to a probability which the highest neuron in the FC layer having the highest probability and all probabilities summing to 1. The output layer is a classification layer that uses the index of the highest probability generated by softmax to reference the name of the class to be outputted.

During training, the network is highly prone to overfitting, which means that it will perform with high accuracy on the training data as it memorizes the patterns but suffers from low accuracy when testing it on data that it had not been trained on. This problem can be solved by reducing the parameters of the network and introducing layers such as batch normalization, which ensures to normalize the scaling in the hidden layers and optimize overall performance, pooling, which helps to make the representation become approximately invariant to small translations of the input and reduce parameters, and activation functions such as Rectified Linear Units, which normalizes the weights generated to increase efficiency.

The setup of our CNN consists of the following which can also be seen in Figure 3.3.4

* Input layer for binary images of size 128x128x1
* Convolution layer using 16 filters of size 20x20
  + Batch normalization layer
  + Rectified Linear Units (ReLU) activation function
* Maxpooling layer of size 2x2 with a stride of 2
* Convolution layer using 32 filters of size 11x11
  + Batch normalization layer
  + ReLU function
* Maxpooling layer of size 2x2 with a stride of 2
* Convolution layer using 64 filters of size 5x5
  + Batch normalization layer
  + ReLU function
* Fully connected layer of the size 62 (26+26+10)
* SoftMax layer
* Classification layer

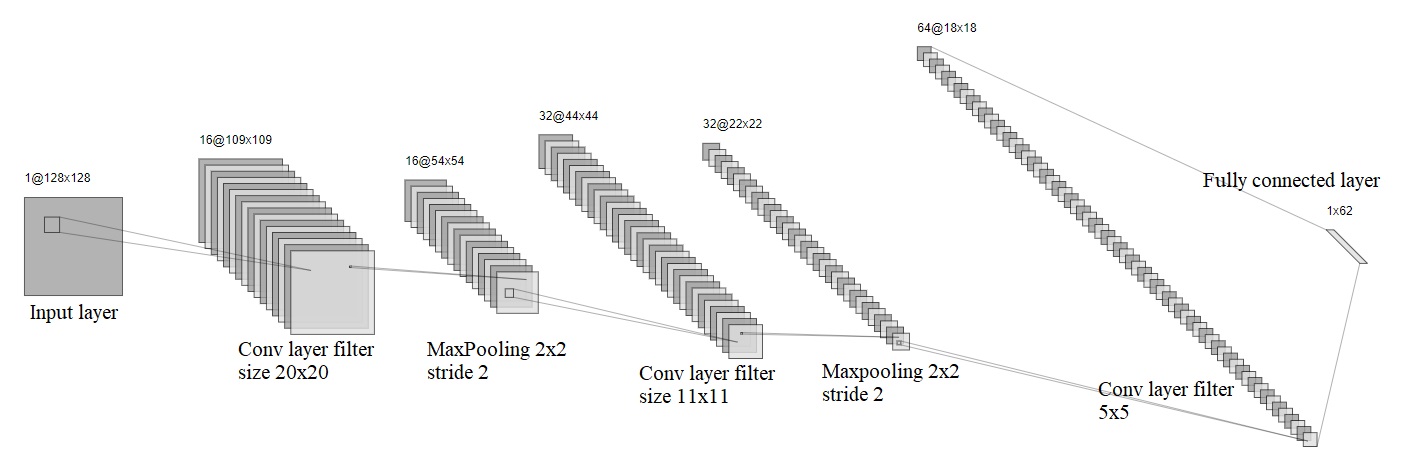


Figure 3.3.4 CNN network architecture

After training the network, the testing dataset is used to test whether the accuracy of the network is close to its validation accuracy, then the weights of the network are saved and imported during the recognition process to pass a grayscale image into the network in order to predict the character in the image. The filters of the first and second convolution layer of the trained network can be visualized as seen in figure 3.3.5 and 3.3.6 respectively.

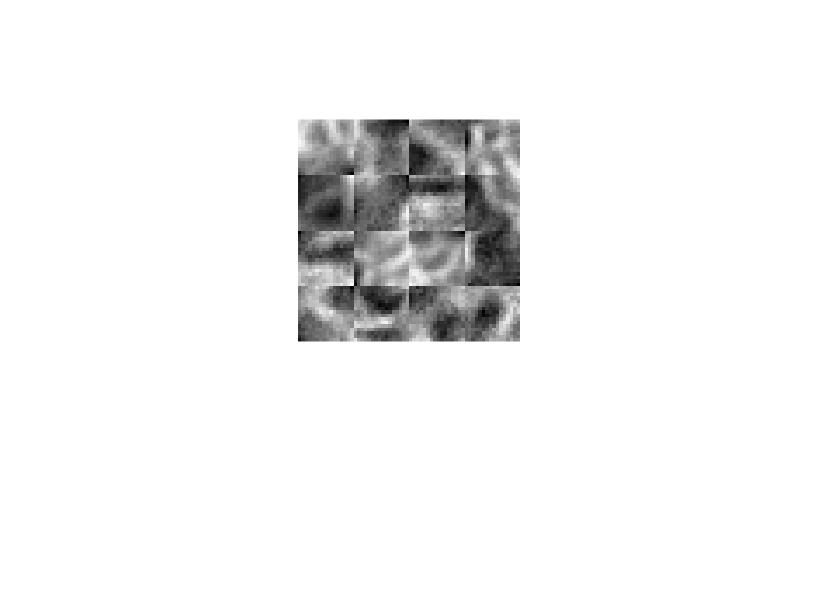


Figure 3.3.5: Visualization of first Conv layer filters



Figure 3.3.6 Visualization of secon Conv filters

# 4.0 Implementation Details

The implementation of functionalities of this project are divided in multiple scripts below is a brief summary while in detailed description of the process will follow.

* **datasetPrep**: Its purpose it to reform a dataset with n samples per character
  + Executed as datasetPrep(‘by\_class’,1000);
  + Converts images to grayscale and complement them before storing
  + Can be dependent on **preprocess** script (deprecated)
* **preprocess**:was used in an experiment to preprocess the training dataset
  + Must be called from **datasetPrep** example: img= datasetPrep(img,[64 64]);
  + Removes white noise from the image by cropping the character and resizing it
  + Dependant on **crop**
* **CharacterRecognitionTest**: takes a JPEG image of handwritten character and return the prediction in text form.
  + Performs rotation and slant correction
  + Extracts individual characters
  + Preprocess the extracted characters
  + Pass to the CNN to do prediction
  + Dependent on **normalize** and cnn weights
* **Trainnn**: takes the directory of dataset formed by **datasetPrep** and trains a CNN and saves it in the root folder automatically.
* **crop and normalize**: its purpose is to turn a rectangle image into square without distorting the character
  + Crop eliminates unnecessary whitespace
  + Normalize adds whitespace

## 4.1 preparing the dataset

The original NSIT dataset had many sample which takes more than three hours to complete one training session. Furthermore, during experimentation, alterations needed to be done on the dataset, hence datasetPrep script was created. First the matlab function *dir(addr)* was used where addr is the dataset folder name to get the names of all the folders inside it.

Once we have the folder names ( ascii characters represented in hex), the folder ‘features’ is created at root to contain the new dataset using the command *mkdir()*. Then for each folder in the training set (each character folder), a subfolder is created inside our features folder having the name of the character in hex representation and s number of samples are taken into memory, processed and stored at the destination using the functions *imread()*, *~rgb2gray()* to make the character strokes white and background black of the greyscale image and *imwrite()* to write to the new destination. Figure 4.1.1 below shows the end result of character folders inside the new dataset folder (features) generated by datasetPrep script.

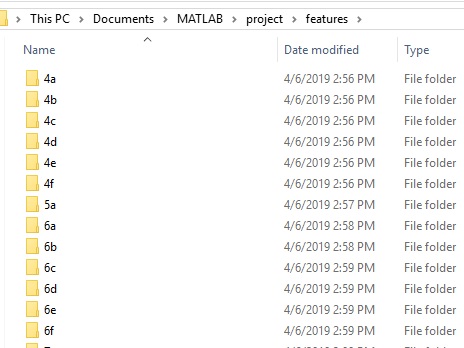


Figure 4.1.1 Features directory

## 4.2 Training the network

Training a Convolutional Neural Network (CNN) is governed by the script trainnn.m. First, the names of subfolders (characters in hex representation) are obtained in a similar fashion as when creating the new dataset. Then an image datastore is created using the command *imageDatastore(fullfile(dataset\_folder,subFolders),'LabelSource','foldernames','FileExtensions','.png');* the parameters are explained below

* Fullfile constructs a directory string of the main folder and all the character folders
* ‘LabelSource’ set to take the folder names as the label
* ‘FileExtension’ set to take all the files with .png extension

The reason imageDatastore was used is to optimize memory efficiency as it does not load all the images to memory but rather keeps their location in a table. Afterwards, the dataset is partitioned to training, validation and testing using the command *splitEachLabel(imds,0.7,0.1, 0.2, 'randomize');* which splits imds which is the original image Datastore to three image datastores having seventy, ten and twenty percent of the original and the parameter ‘*randomize’* ensures to randomly pick images when splitting.

During experementation, an imageDataAugmenter was used to add random rotation, shear and translation to the training set in an attempt to increase accuracy and solve overfitting, Figure 4.2.1 below shows the use of imageDataAugmenter on the training set.

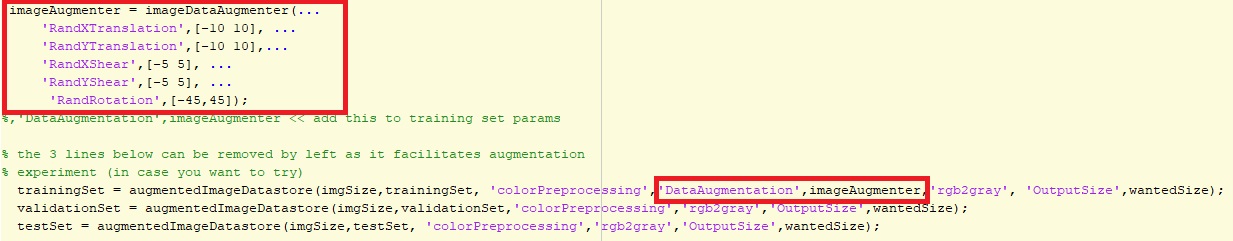


Figure 4.2.1 ImageDataaugmenter usage

Layers of the CNN are defined in an array with their respective functions as shown in the figure 4.2.2 below .

* imageInputLayer: takes [width,height,color channels] as a parameter
* convolution2dLayer: takes (window size, number of filters) as a parameter
* batchNormalizationLayer: to normalize the scale variance and optimize performance
* reluLayer: activation function to keep calculations sane by setting any negative value to zero]
* fullyConnectedLayer: takes the number of classes as input
* Softmax: generates the probability for each class
* Classification layer: outputs the name of the character in hex

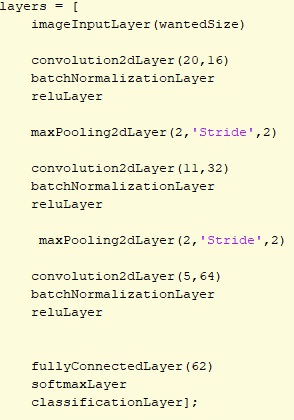


Figure 4.2.2 CNN layers setup

For the training options, the settings below were used which can also be found in the code snippet in figure 4.2.3 that shows how to declare them

* ‘Sgdm’ for stochastic gradient descent with momentum (good to avoid stucking at local minimum when reducing error)
* ValidationData has the validation image datastore for validation after training batch
* InitialLearningRate is the percentage of consideration of the values obtained from the back propagation process (when the network is wrong and need to adjust weights)
* Shuffle set to every-epoc to shuffle the dataset after each epoch to prevent overfitting
* MaxEpochs is the maximum rounds to train
* MiniBatchSize is the number of images in one batch to be used for training

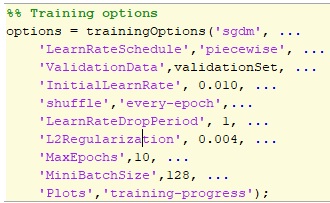
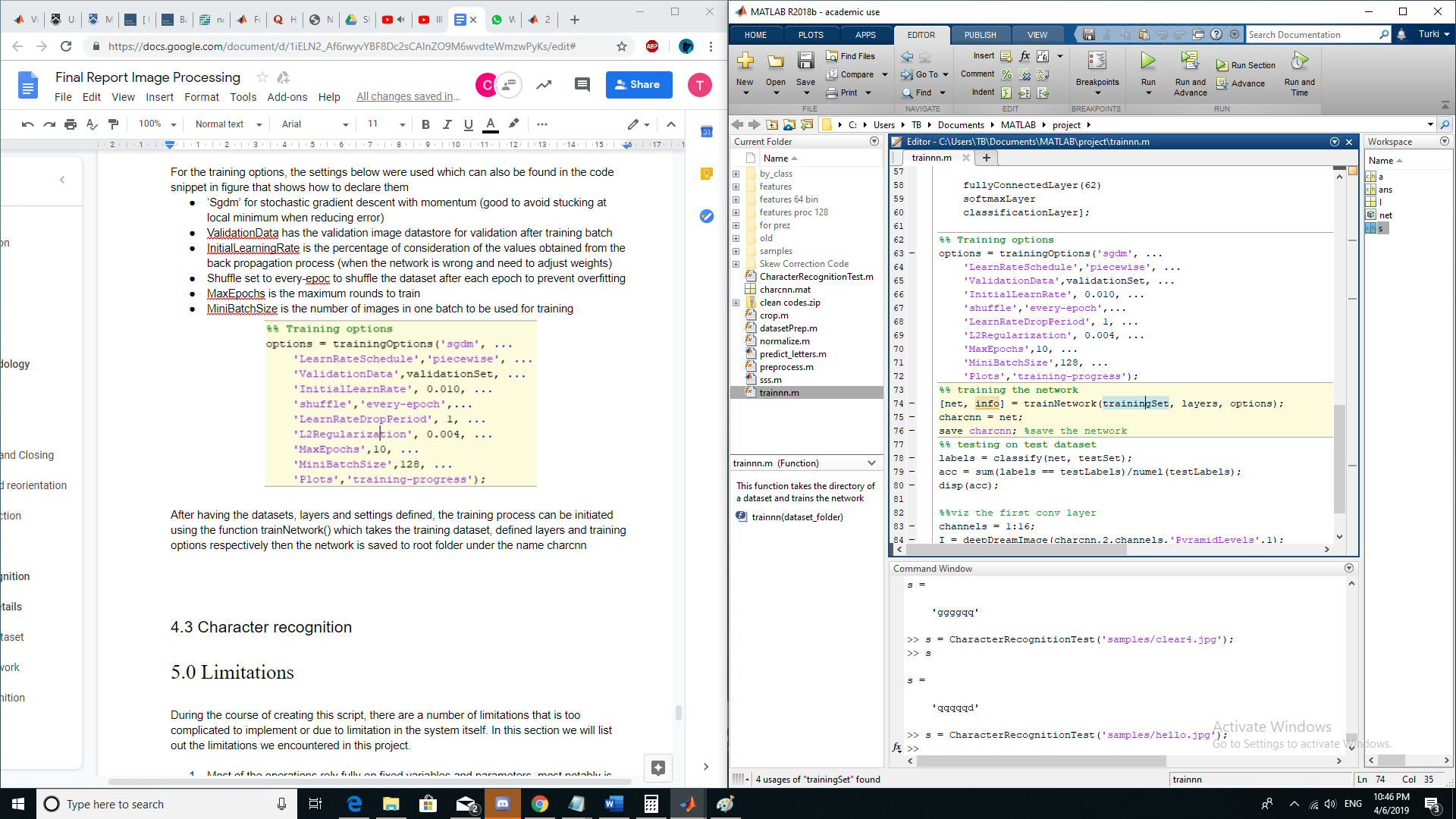
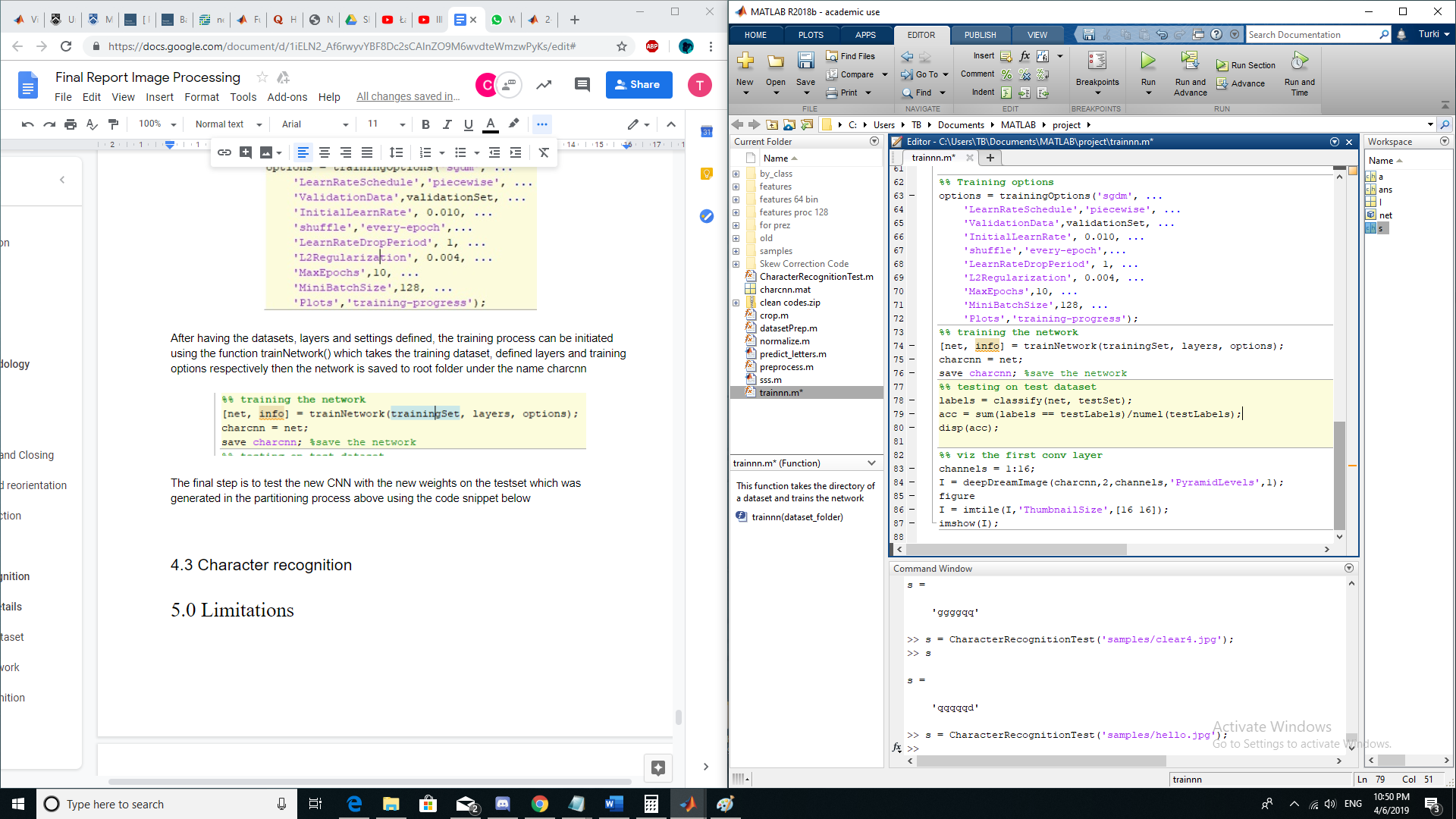


Figure 4.2.3: CNN training options setting

After having the datasets, layers and settings defined, the training process can be initiated using the function trainNetwork() which takes the training dataset, defined layers and training options respectively then the network is saved to root folder under the name charcnn



The final step is to test the new CNN with the new weights on the testset which was generated in the partitioning process above using the code snippet below



## 4.3 CharacterRecognitionTest

The CharacterRecognitionTest.m is the driver class for the entire system. It starts with preprocessing and segmenting the image, and then finally calling the neural network to identify the code.

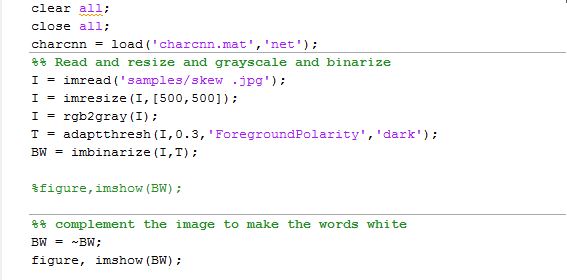


Figure 4.3.1

In Figure 4.3.1, the function we see are:

* Imread, Simply to read the image into a variable in Matlab
* Imresize, This resizes an image to a specific size, it uses bicubic interpolation by default.
* Rgb2gray, Which converts a rgb image into a grayscale image. Having 3 matrices that represent each primary color of the image to 1 matrice that represent the image in 1 color tone.
* Adaptthresh, Adaptive image thresholding which computes the average illumination of the background, for the next function to use, ‘ForegroundPolarity’ and ‘Dark’ tells the function that the foreground of the image is darker than the background
* Imbinarize, takes the computed threshold value of the previous function, and binarize the image accordingly
* ~I, This reverses the image, turning black pixels to white and white pixels to black, This is because we assume that the image text was written on a bright paper with a dark pen.

As listed above, the code section shown in Figure 4.3.1 are mostly to deal with reading and binarising the image.

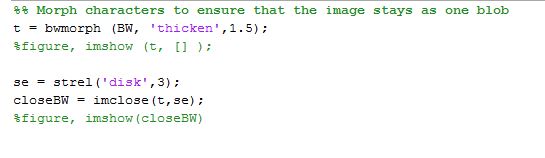


Figure 4.3.2

In this section shown in Figure 4.3.2, the morphological operations happens here. Only two specific functions are used in this section

* bwmorph(‘thicken’), This operation thickens the image by adding pixels along the perimeter of the words, resulting in a much more substantial and prominent character
* Imclose, utilises a structure element created by the strel function, to close and join any possible disconnects between pixels, treating the structure element as a brush to fill in the gap

As explained in the previous section, the morphological operations are used to ensure that the characters are one single blob instead of a disconnected entity.

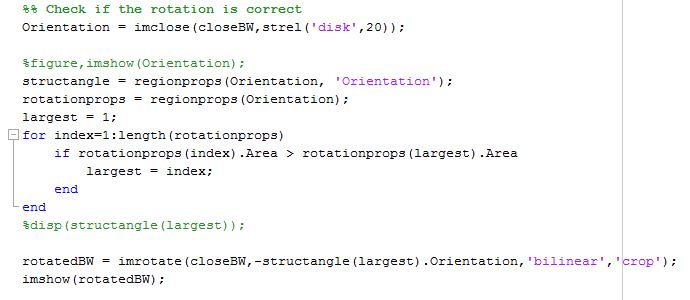


Figure 4.3.3

The section shown in Figure 4.3.3 deals with reorienting the image if the rotation is off the horizon. The entire code structure relies on two main functions:

* regionprops(Orientation), this takes a “blob” and treats it as an eclipse or an oval and takes the major axis of the eclipse and measure it against the x-axis, determining the angle it is off the horizon. Figure 4.3.4 visualises the function

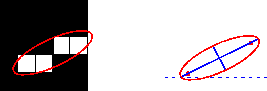


Figure 4.3.4

* Imrotate, quite simply rotates the image based on the angle given

So in the section, we see a thickening function that converts the entire word to a giant blob temporarily, this is so that the regionprops(Orientation) would find the orientation of the entire word instead of a single character

However, considering that regionprops still considers all the “blobs” in the image, we have to filter out and only take one significant blob and rotate the entire image based on the chosen “blob”. This is a temporary fix to a limitation on our script which finds difficulty in identifying multiple lines of texts, especially if they are all in different orientation.

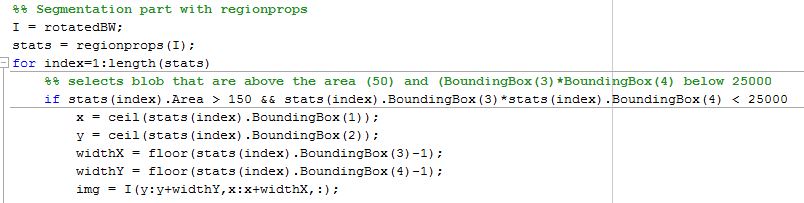


Figure 4.3.5

In this section, we see the same functions that have been used before. The loop simply goes through each potential “blobs” of the appropriate size that is returned by region prop and saves the coordinates of the “blob”

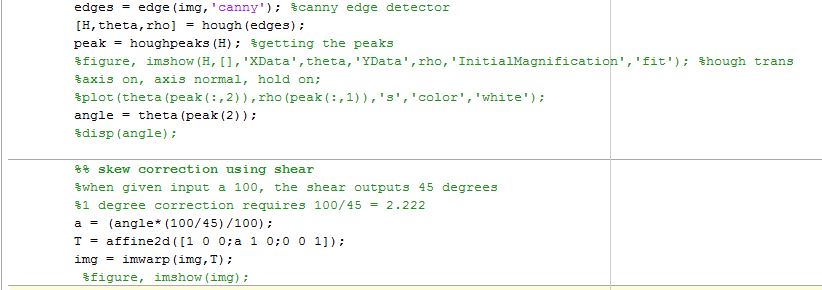


Figure 4.3.6

This section shown in Figure 4.3.6 deals with skew corrections while each character is being segmented. We can see two features in this section, which is

* Hough Peaks, This detects the general direction of the image, by detecting the edges by using a canny edge detector, then identifying the Hough peaks in the image. The result is the angle of which the general direction the image is pointing to. If the image is slanted, then the general direction of the image would not point straight anymore.
* Imwarp, This function warps the image based on the transformation matrix given to it. In variable T we can see a matrix used to shear an image, which can be used to correct a slanted image. By getting the angle using hough peak, and a little trial and error which found that changing 1 degree requires 2.222 increase in the variable a of the transformation matrix, we can correct the slanted characters accurately.

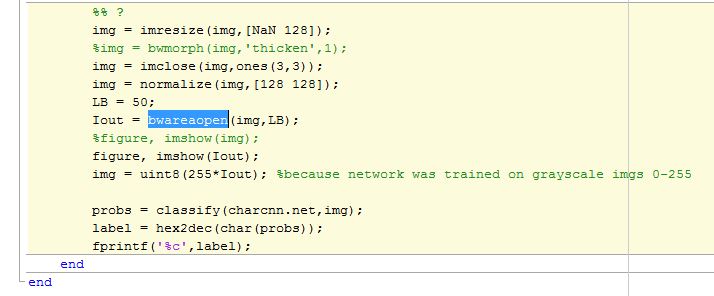


Figure 4.3.7

In the final section of the code, we remove any stray strokes that can be caused by the skew correction in the previous section by using bwareopen. This function simply removes any strokes that has a pixel area of less than a certain value.

After that, the remainder of the script prepares the segmented character image for the neural network, resizing it to 128x128 size, and calling the function to identify the letters. The final line of the code prints the predicted text to the command window

# 5.0 Results and Discussion

At earlier stages, the network was extremely overfitting as the validation accuracy converges around sixty percent while training accuracy reaches up to one hundred percent. This is due to the training set being swapped with validation set which taught as a lesson that the lack of data could cause overfitting.

After the correction, the network seems to converge below seventy percent as seen in figure 5.0.1 below, however, training accuracy is a bit higher which made us run an experiment with augmenting the training set before passing it to the CNN.

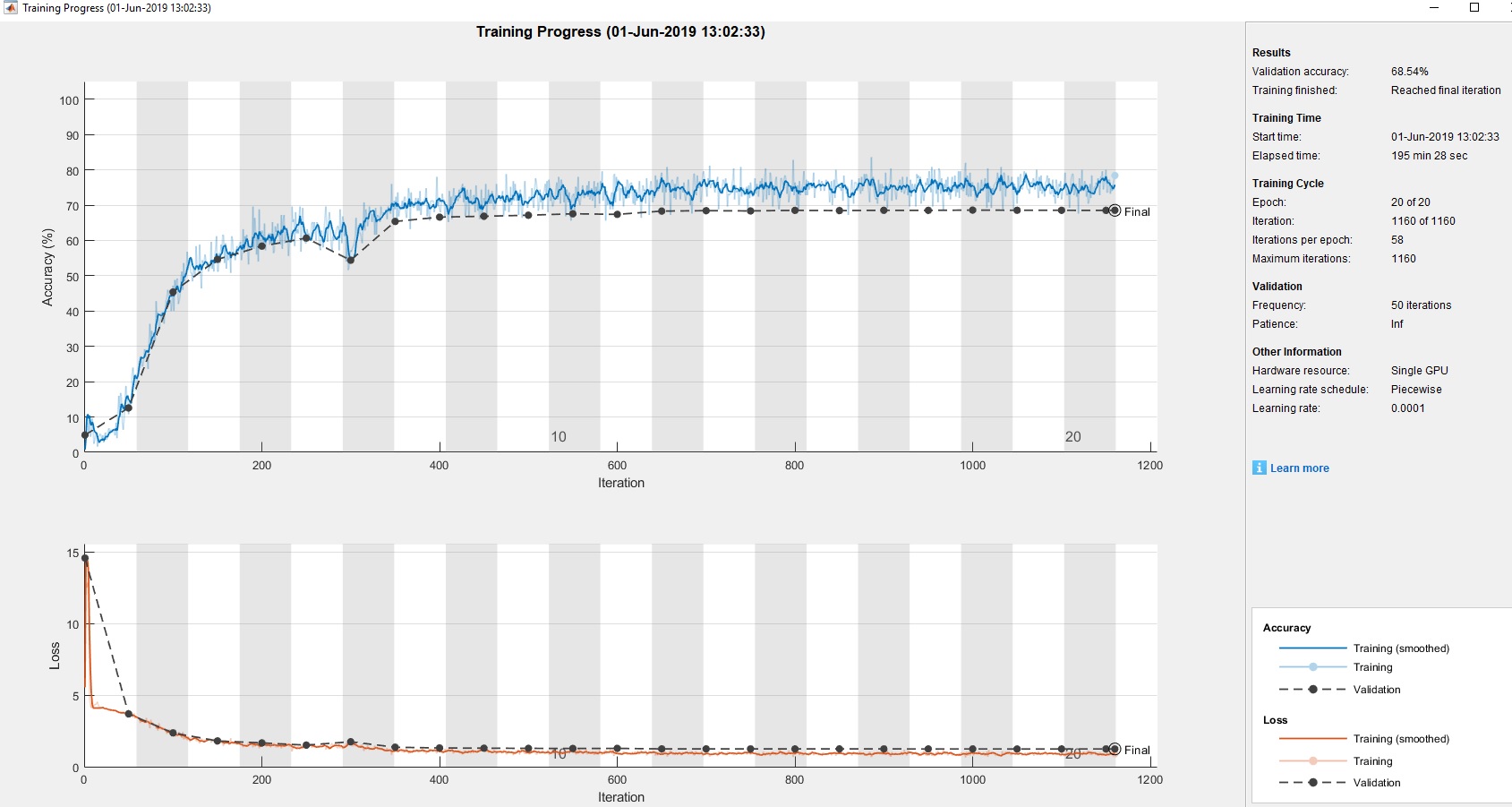


Figure 5.0.1 Overfitting while training

After augmenting the data by adding random rotation, translation and shear to the training set, we have noticed that the validation accuracy is always higher than training accuracy as seen in the figure 5.0.2 below, which is a hint that the network is generalizing well and not memorizing the training set.

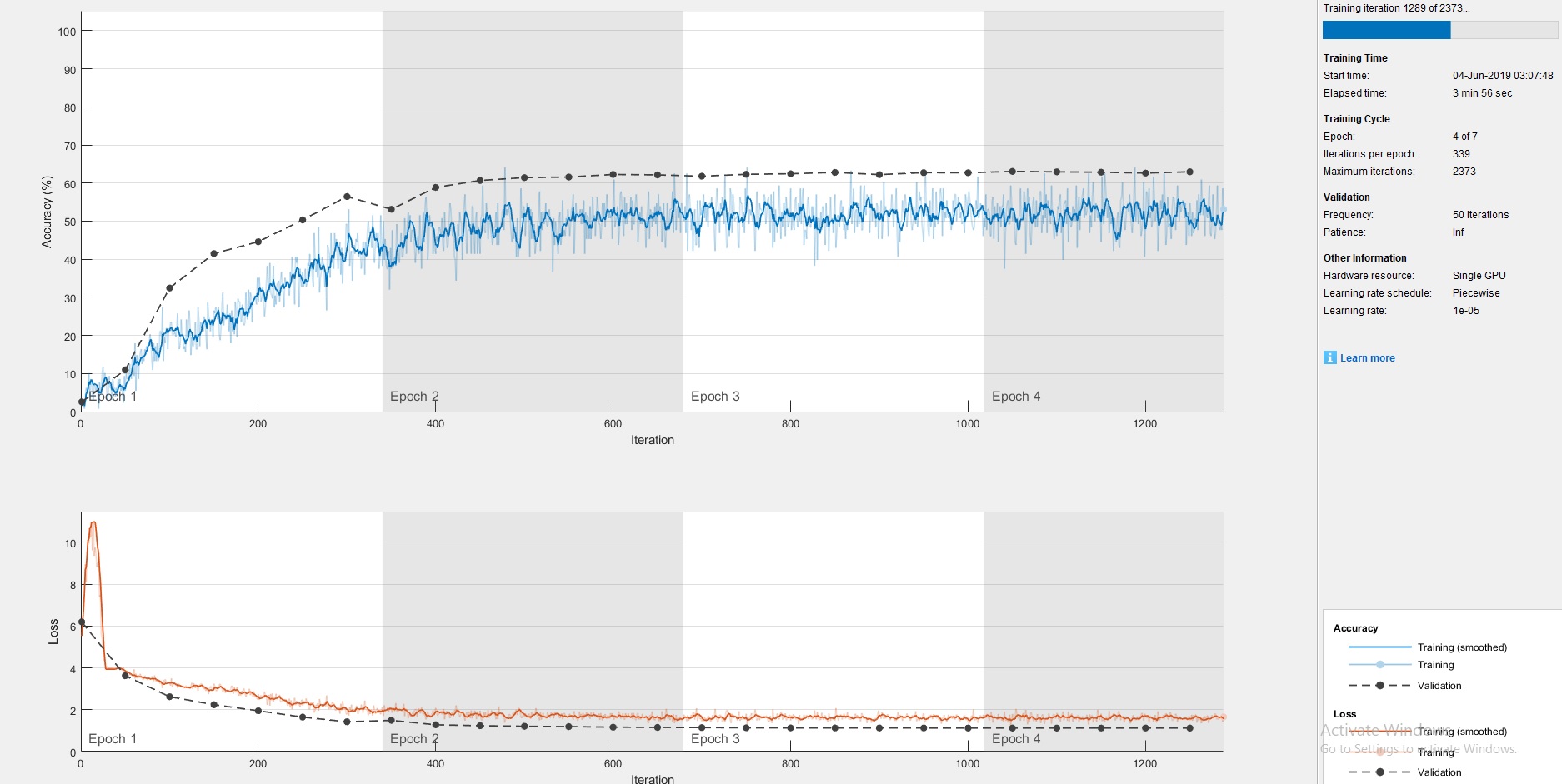


Figure 5.0.2 Training with augmentation

However, in the previous two experiments, the input size was 64x64x1 images with unoptimized weights. After running many experiments to optimize the accuracy, the architecture configuration in the methodology section is selected as it achieved above eighty percent accuracy for both testing, validation and training and the training progress can be seen in figure 5.0.3 below.

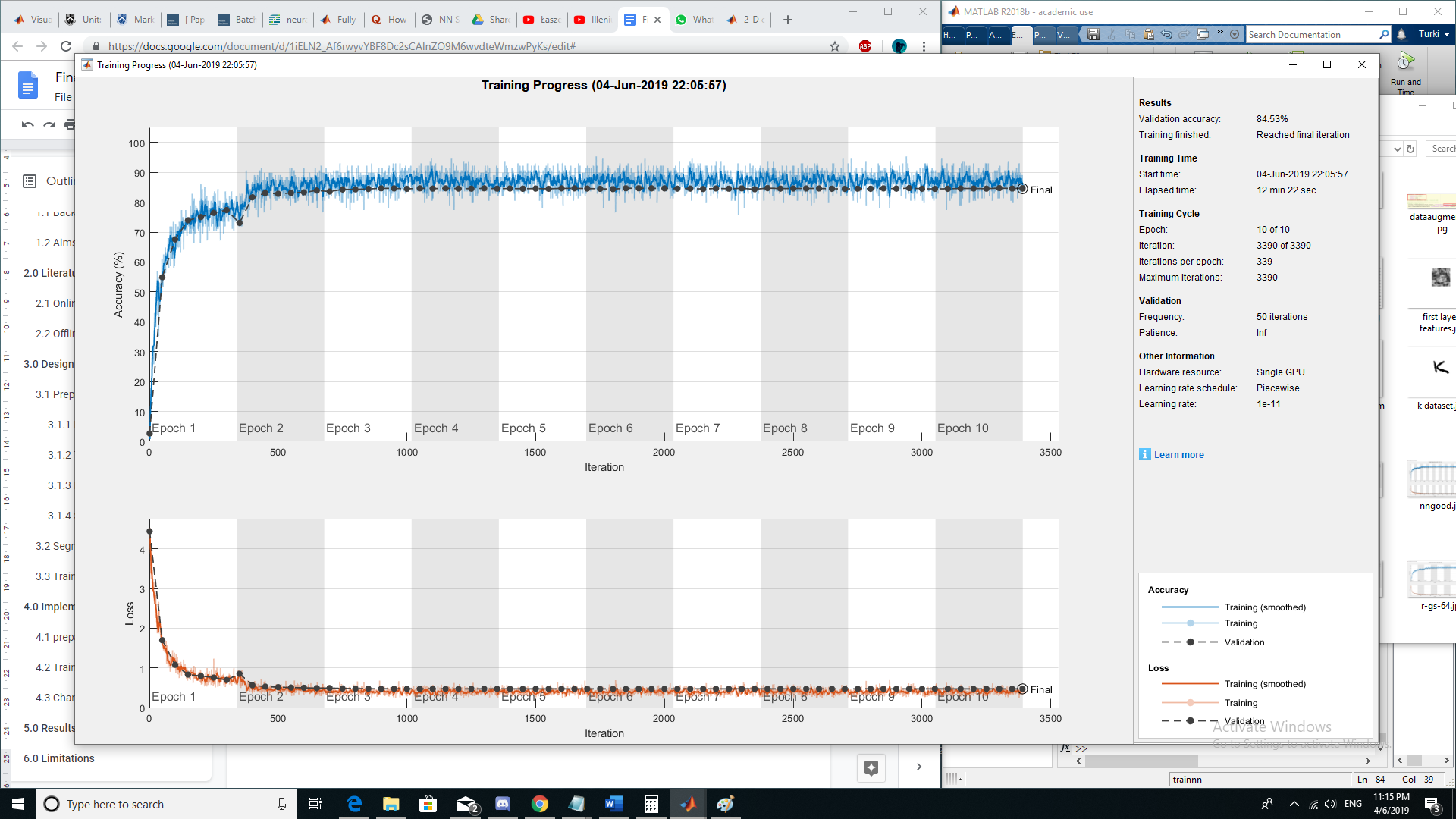


Figure 5.0.3 Best result

However, one major problem is using the network to classify characters as it no longer able to recognise any character at all compared to the mid stages (during presentation week) as seen from the figure 5.0.4 below. We have spent majority of time in the late stages retraining and optimizing the CNN and looking into as many aspects regarding the network as we could but no improvement when it comes to using it for actual prediction.

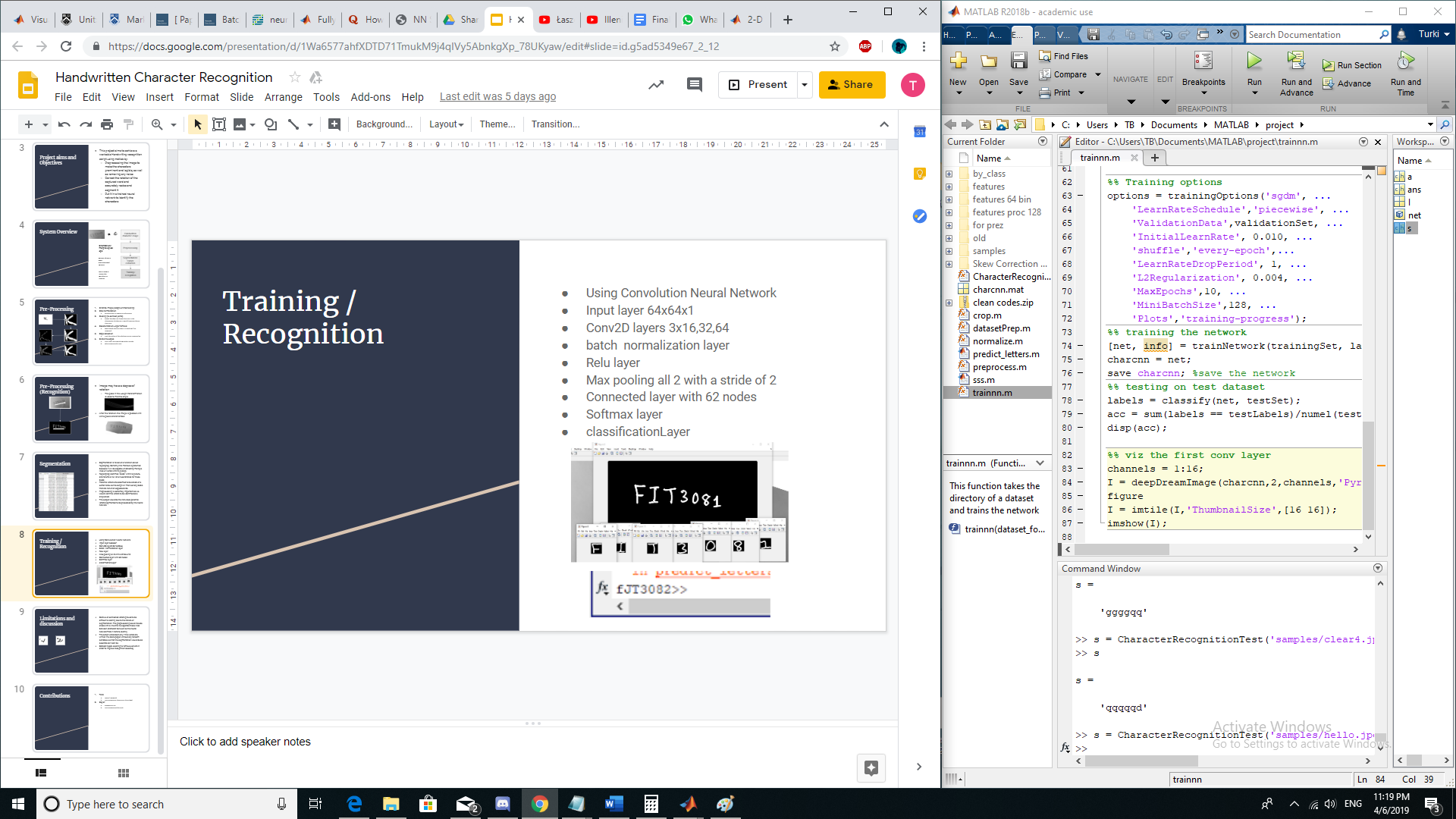


Figure 5.0.4 Results of prediction

# 6.0 Limitations

During the course of creating this script, there are a number of limitations that is too complicated to implement or due to limitation in the system itself. In this section we will list out the limitations we encountered in this project.

1. Most of the operations rely fully on fixed variables and parameters, most notably is the segmentation where if the area of an image region is below a fixed value, it will treat it as noise. Other functions such as the intensity of the morphing functions or the objects used to morph, are all using fixed function. Future improvements for this project was to determine a more dynamic approach for the preprocessing step.
2. Cursive words will not be segmented with our current approach, this is because the current method of determining individual characters was utilising regionprops, which require each character to be disconnected with each other. An improvement to this would be to either erode the links between the characters, or utilise an entirely different method of segmentation.
3. The dataset augmentation function is not robust in matlab as it was not able to perform rotation, shear and translation on images with a single color channel such as a binary image or greyscale. This could have been a major increase in accuracy for us.

# 7.0 Conclusion

Although the system was not able to correctly classify all the handwritten characters written on an image, it manages to successfully rectify tilted and slanted handwritings and manages to achieve validation and testing accuracy of above eighty percent on the dataset.

# 8.0 References

1. Arica, N., & Yarman-Vural, F. T. (2001). An overview of character recognition focused on off-line handwriting. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 31(2), 216-233.
2. Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. IEEE transactions on neural networks, 5(2), 157-166.
3. Dabov, K., Foi, A., Katkovnik, V., & Egiazarian, K. (2007). Image denoising by sparse 3-D transform-domain collaborative filtering. Image Processing, IEEE Transactions on 16 (8), pp. 2080-2095.
4. Eikvil, L. (1993). Optical character recognition. citeseer. ist. psu. edu/142042. html.
5. F. Cesarini, M. Gori, S. Mariani, and G. Soda, “Structured document segmentation and representation by the modified X–Y tree,” in Proc.5th Int. Conf. Document Anal. Recognit., Bangalore, India, 1999, pp.563–566.
6. J. Ha, R. Haralick, and I. Philips, “Document page decomposition by bounding-box projection technique,” in Proc. 3rd Int. Conf. Document Anal. Recognit., Montreal, QC, Canada, 1995, pp. 119–122.
7. K. Kise, O. Yanagida, and S. Takamatsu, “Page segmentation based on thinning of background,” in Proc 13th Int. Conf. Pattern Recognit., Vienna, Austria, 1996, pp. 788–792.
8. Maheshwari, M., Namdev, D., & Maheshwari, S. (2018). A Systematic Review of Automation in Handwritten Character Recognition. International Journal of Applied Engineering Research, 13(10), 8090-8099.
9. MATLAB and Statistics Toolbox Release 2018b, The MathWorks, Inc., Natick, Massachusetts, United States.
10. O. Akindele and A. Belaid, “Page segmentation by segment tracing,”in Proc. 2nd Int. Conf. Document Anal. Recognit., Japan, 1993, pp.314–344.
11. Otsu, N. (1979). A threshold selection method from gray-level histograms. IEEE transactions on systems, man, and cybernetics, 9(1), 62-66.
12. Paul, I. J. L., Sasirekha, S., Vishnu, D. R., & Surya, K. (2019, April). Recognition of handwritten text using long short term memory (LSTM) recurrent neural network (RNN). In AIP Conference Proceedings (Vol. 2095, No. 1, p. 030011). AIP Publishing.
13. Poovizhi (2014). A Study on Preprocessing Techniques for the Character Recognition.
14. R Patel, Shivangkumar & Jha, Jasmine. (2015). Handwritten character recognition using machine learning approach - A survey. 1-5. 10.1109/EESCO.2015.7253978.
15. R. G. Casey and E. Lecolinet, “A survey of methods and strategies in character segmentation,” IEEE Trans. Pattern Anal. Machine Intell., vol.18, pp. 690–706, July 1996
16. Sá, J., Alexandre, L., Duch, W., & Mandic, D. (2007). Artificial Neural Networks ICANN 2007 part2 (pp. 230 - 239). Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg.
17. Verma, R., & Ali, J. (2013). A comparative study of various types of image noise and efficient noise removal techniques. International Journal of advanced research in computer science and software engineering, 3(10).Verma, R., & Ali, J. (2013). A comparative study of various types of image noise and efficient noise removal techniques. International Journal of advanced research in computer science and software engineering, 3(10).
18. Wang, T., Wu, D. J., Coates, A., & Ng, A. Y. (2012, November). End-to-end text recognition with convolutional neural networks. In Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012) (pp. 3304-3308). IEEE.
19. Zafar, M., Mohamad, D. and Othman, R. (2007). On-line Handwritten Character Recognition: An Implementation of Counter propagation Neural Net. 1st ed. (ebook) Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.922.4879&rep=rep1&type=pdf (Accessed 14 Apr. 2019).

# 