1.
2.

Optical Character Recognition

Nithesh Koneswaran
Nk00374
COM2028

1. Introduction
Optical character recognition is a widely researched topic and has a broad range of usage some of which include converting printed documents into a digital copy, banks using OCR to handle cheques, translating text extracted from an image to another language. With the increased usage and demand of OCR it is important to research areas and develop algorithms in order to minimise the errors involved since the impact of having a poor OCR can be significant.

My objectives for this project are as follows:

• Creating an engine that will recognize and output text from images of printed text.
• Creating an engine that will recognize and output text from my own handwritten notes.
• Creating an engine that will recognize people's handwritten signatures and output their names.
• Creating an engine that will recognize the author or a handwritten piece of writing.

For this project I will be creating an array of engines that will translate images of handwritten pieces of texts/signatures/printed documents to text using methods such as CNN, KNN and SVM. What we are essentially building is a far less complex optical character recognition that will take an input through a camera and extract the text from the image. The complexity of such an OCR will be discussed in the 'Analysis' section of this poster. In this project I will discuss and identify the advantages and disadvantages of the techniques I used to implement my objectives and discuss the results I have gotten and making comparisons between different techniques. I will also be discussing about the following processing stages: data preparation, feature extraction and classification and the overall performance.

2. Analysis and approach to the problems
When it comes to text recognition, there are numerous factors to consider. The primary being how exactly we are going to extract the images from the images. This is an essential step for text recognition, we could manually crop and process the images and split them into equal sized letters, though this job becomes tedious especially when an image contains many potential text letters. The solution was to [1] segment the image with image labeling to extract the letters from the images, this extraction tool was provided to us.

In order to further reduce the complexity of our optical character recognition to allow the extract tool to perform optimally, we will only be trying to extract letters from a blank white, a4 sized paper, this is far more optimal since trying to recognize text from objects can introduce a lot more noise requiring a much thorough preprocessing of images. The structure of the text is quite important too and can affect the extraction quite a lot, it is important that each contiguous letter is separated by a wide enough gap and that the size of the text should also be appropriately sized for the extraction tool to pick up. When extracting the letters, each corresponding letter snippet should be assigned with a label. It's quite common for the extraction tool to make errors causing it to pick up few little or more characters than it should, requiring external assistance to fix these errors such that the alignment with the snippets and labels are correct. These assistances include applying a threshold to the input image or manually removing marks using a text editor such as gimp or adding code to correctly align the snippets with their corresponding labels. When creating our training and testing samples it was very important that they follow a correct standard such as: making sure lines of text aren't slanted, letters aren't rotated too much, using an ink pen/fine liner that will draw the letters clearly. If these steps were broken, the accuracy of our model will significantly be impacted especially if each snippet were labeled incorrectly. Therefore, taking cation in the first processing step can be pivotal.

An issue that arose during Task A was trying to assign the labels (from the truth file) to the images. I found that the extraction tool did not extract some letters from the images preventing my test images not to be labeled and hence causing me to calculate the accuracy manually. The extraction tool also considered punctuation, a solution to prevent punctuation from being picked up is to minimize the size of the bounding box. Another technical issue that arose was the classifier's poor ability to classify lower case letters and capital letters that looked the same, these letters include c, x, v, w, m. It also had a low ability to differentiate between some letters and numerics that look similar, these include 1,l, 7,Z. This is quite comparable to humans since we also tend to make mistakes when differentiating between some letters. However the aim for OCR is to minimize the amount of errors as explained in the introduction.

A proposed approach that would be challenging would be to extract the letters using sliding windows. This would be effective at extracting words with letters that are joined together. Using sliding windows would make our OCR more effective and realistic since most people would write joining their letters together. Also with the sliding windows it is possible to translate the text so that words are outputted instead of each individual letter.

For my project, the feature that we extracted from the images was simply the pixel intensity of the image. Each pixel intensity ranged from 0 to 1 since the image was in grey scale and normalised appropriately. However, upon research I found that to further improve the performance of the model I could further process the image by applying Otsu threshold method. This method was used to globally threshold each snippet so that each image would only consist of 0's and 1's, this is something that the extraction tool was meant to do however after it had resized the image, some noise was introduced. [2] Binaralizing an image is a great image preprocessing technique which is useful for much more complicated images in order to reduce significant amount of noise and separate an image's foreground from the background. [3] Figure 1,2 and 3 shows the after effects of the processing.

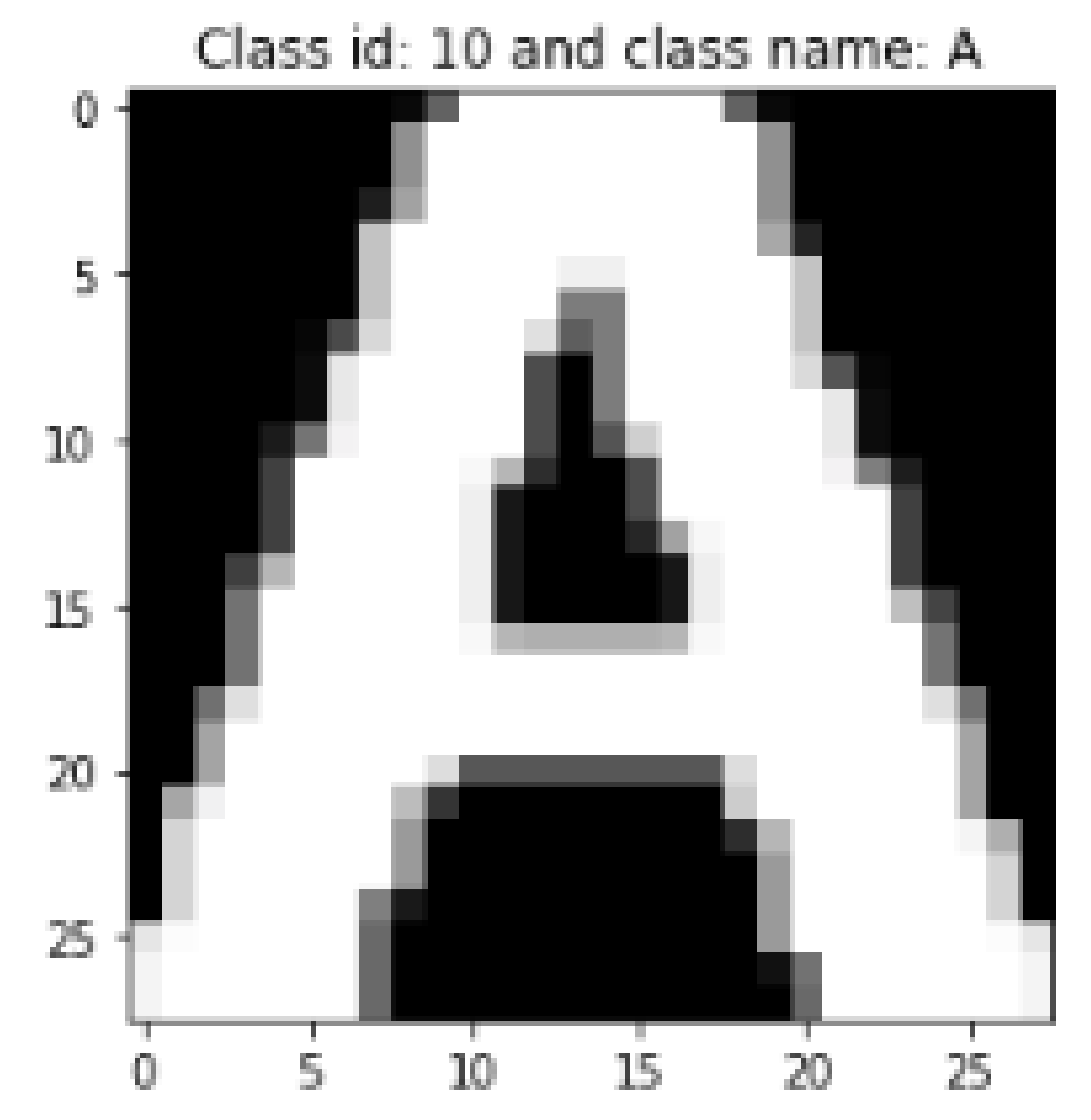


Figure 1

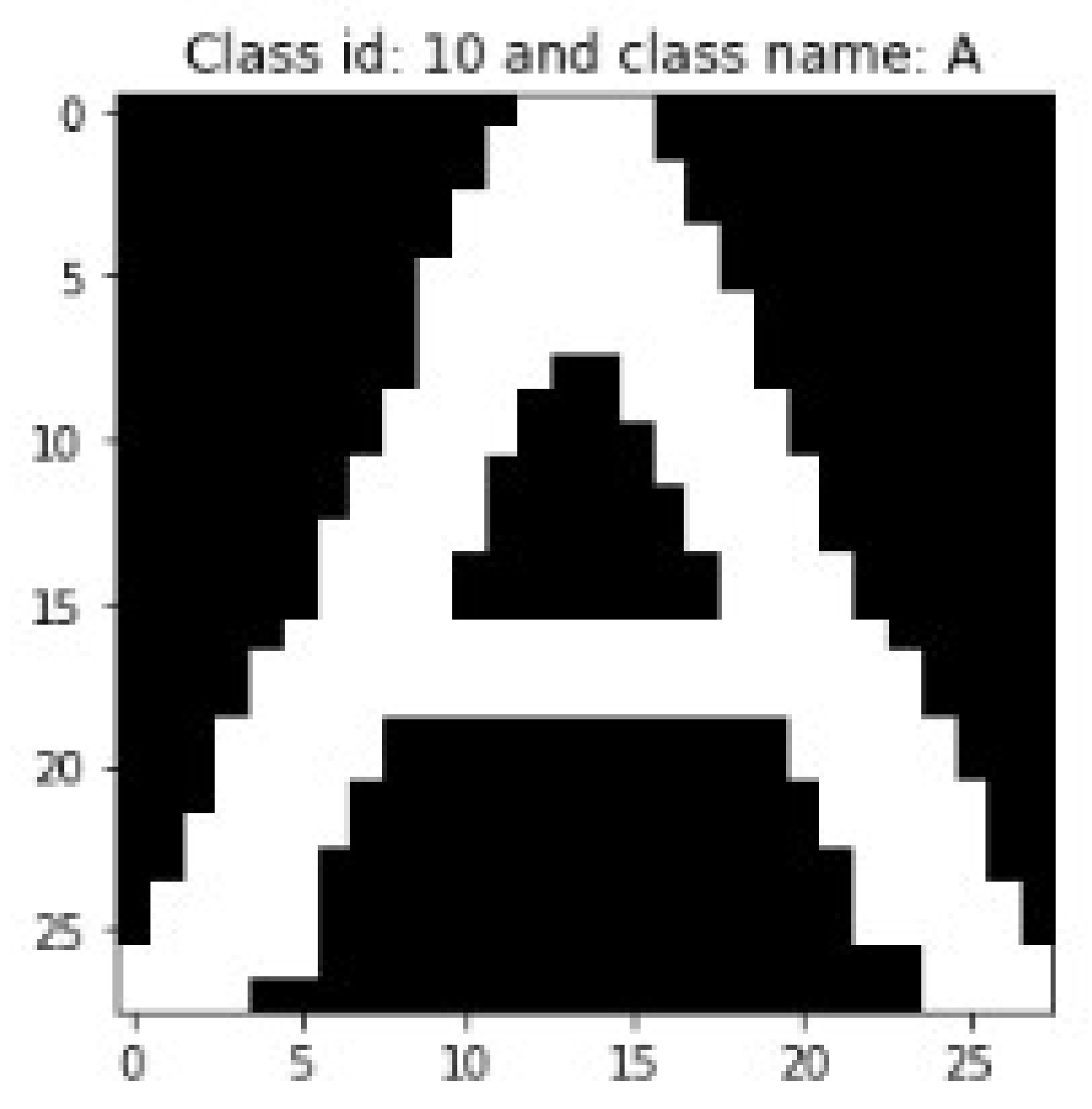


Figure 2

3. CNN, KNN, SVM
All three of these models are all supervised learning although CNN can be implemented to be either supervised learning or unsupervised learning.

[2] The support vector machine algorithm simply calculates a hyper-plane (decision boundary) between two classes.

CNN [3] Convolution Neural Network is able automatically extract relevant features from our input images and train its parameters in order to classify objects These features are extracted from the layers which make up the CNN, these being the input, hidden and output layers. Compared to Neural Network, CNN out performs CNN. From our results we can see the most models performed quite evenly.

KNN [4] Kth Nearest Neighbour is most effective when the training data is large. This is something that I found in my results, KNN performed better than CNN for the Handwritten notes. However a disadvantage of K nearest neigher is that we need to find the optimal value for K which requires a lot of pre-testing and experimenting. I have done this optimisation in my code.

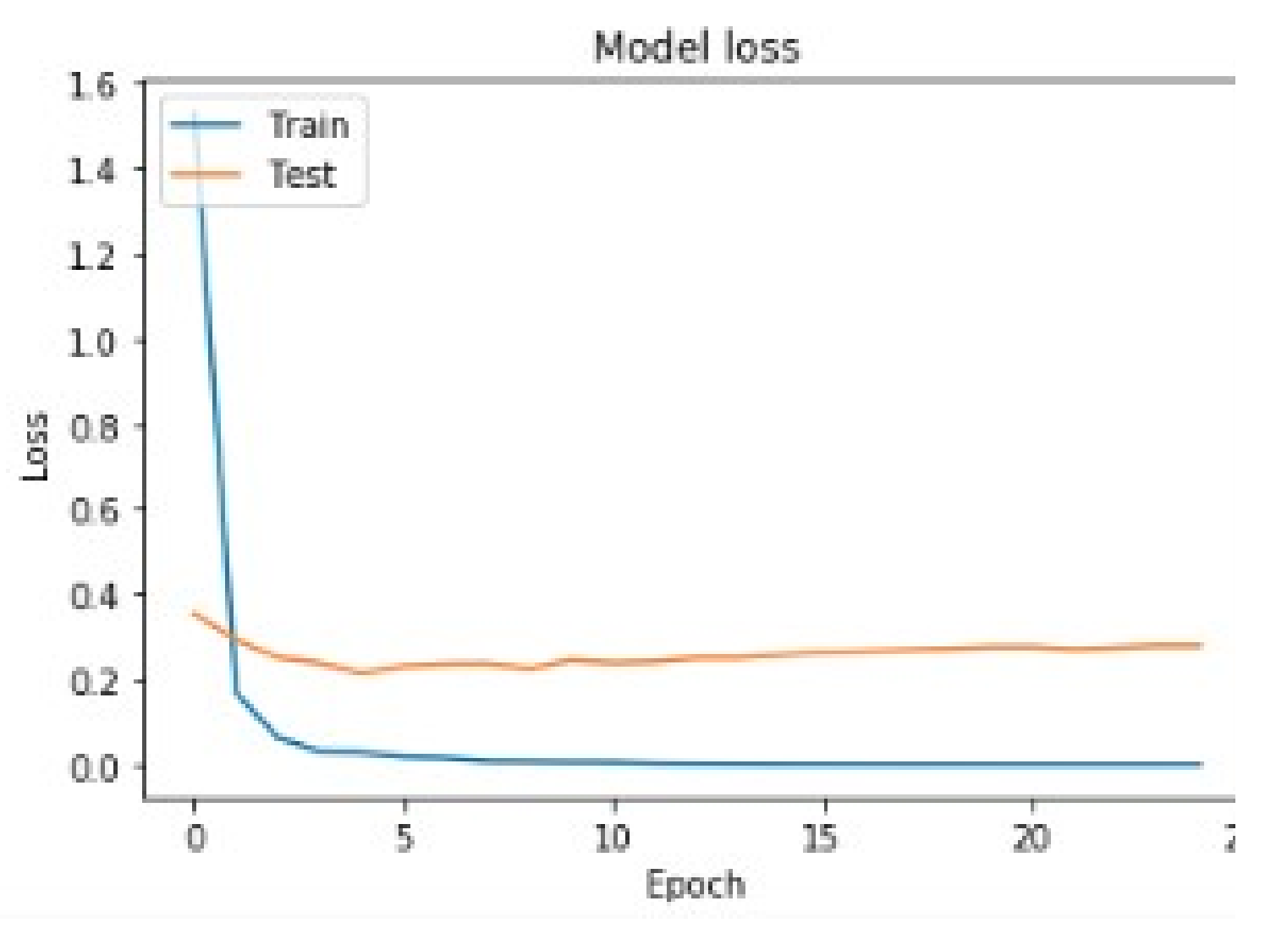
4.1 Recognising Printed Pages;
Note: Since the labels of the testing was not accurate, I had to manually check each individual letter with the correct letter in order to calculate the accuracy.

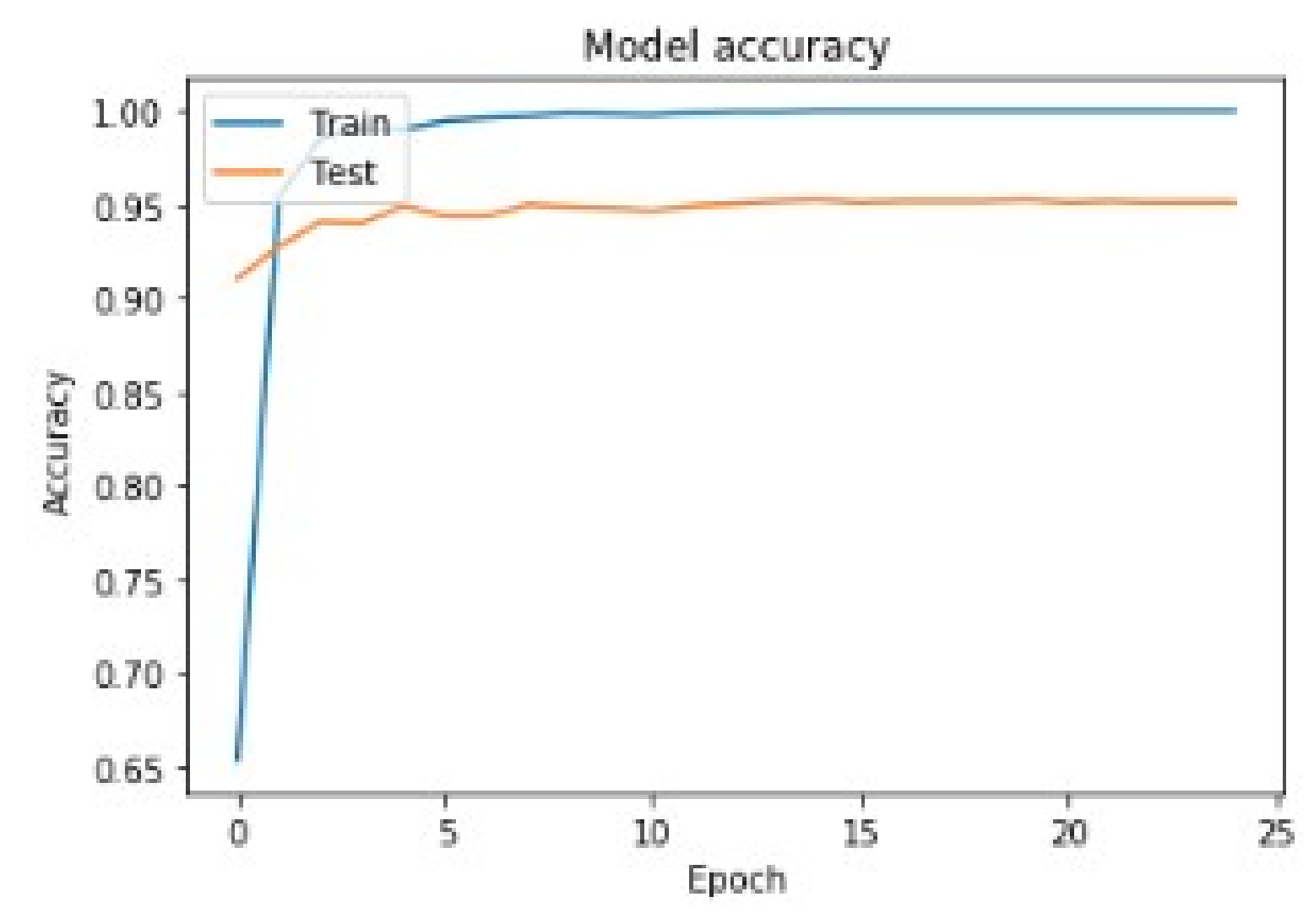
I was able to recognise printed document text and output text accordingly using a CNN model. Due to not having a testing set I did not implement an SVM or KNN alternative. However for the handwriting and signature task, CNN, SVM and KNN have been implemented. See Appendix to learn more.

4.2 Handwritten Notes
This task required my own dataset of handwritten notes. I began by writing down 100 characters for each letter in the alphabet and for each number from 1 to 9. Once the dataset was created I fed this training data through my convoluted neural network and tested against my test data (see figure 4 for test data).

One of the challenges that I came across when extracting the letters and appropriately labelling them each was that the extraction tool kept missing some letters messing up the labelling. This was fixed by creating an image file for each letter in the alphabet, that way even if the extraction tool missed some letters out, it would still label the snippet the correct character since whenever we move on to another image file the label will change too.

I began with CNN since it was already completed, the only changes that was required was the extraction method. After fitting the model we can see how our model is being trained. We can see for every epoch (how many times we go through the data set) we see that the val_loss calculated by the function 'categorical_crossentropy' decreases every time. This function uses gradient descent to calculate the optimal trainable parameters and will do so by minimising the loss function.





We can see through these charts that as we go through the dataset a number of times we can see that the model's accuracy greatly improves. This is due to the model's loss function being minimised.
Below is an image of what my system predicted and what it should have. From that I calculated the overall confidence and accuracy

From this set of results shown in Figure 5, Figure 6, Figure 7, we got that the SVM got accuracy 0.64, CNN got accuracy 0.68, KNN got accuracy 0.71.

Overall the results was quite similar with each classifier getting mixed up between letters,numbers, lower case letters and upper case letters. This is because we are relying on the pixels as a feature vector, pixel patterns that look the same would most likely get classified to a class that is quite similar. Instead of using pixel intensity I could have used Hog (histogram of orientated graph) descriptor.

After completing CNN implementation I moved on to implementing the task with SVM and KNN. My results are shown below.

4.3 Signature recognition;
This task required datasets of people's signatures. After finding 5 volunteers (including my self) I created a training set and testing set to be used for my models. The training and testing samples are shown in figure 8 and figure 9.

Testing using CNN, see Figure 10. We can see that we achieved a val_acc score, however this is because our classifier has been over fitted. The cause of this was because the training samples were split incorrectly. I was training the model with a validation set from the training data that was entered into the model. After splitting the training sample properly with a 70% to 30% ratio I ran the tests again and the results were surprisingly much better than the over fitted model. My results are shown in Figure 11.
Using CNN the model was evaluated with accuracy 0.87 (2f)

Testing using SVM I found that the model had an accuracy of 0.82 (2sf) and testing using KNN my model had an accuracy of 0.84 (2sf)

Again I believe that the classifier all performed quite similar, making similar mistakes. Again we could have used something such as histogram of orientated graph descriptor which works by calculating the frequency of a particular direction and is computed over the whole image.

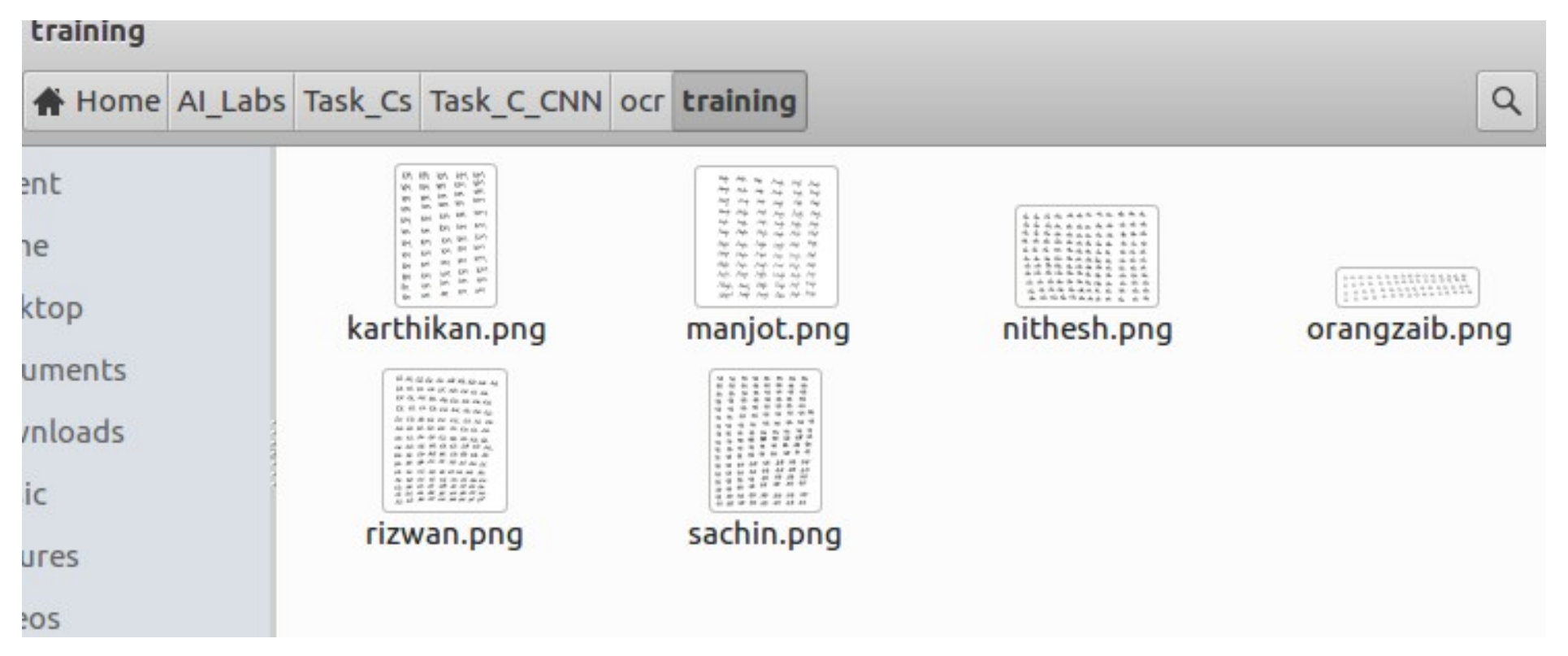


Figure 8

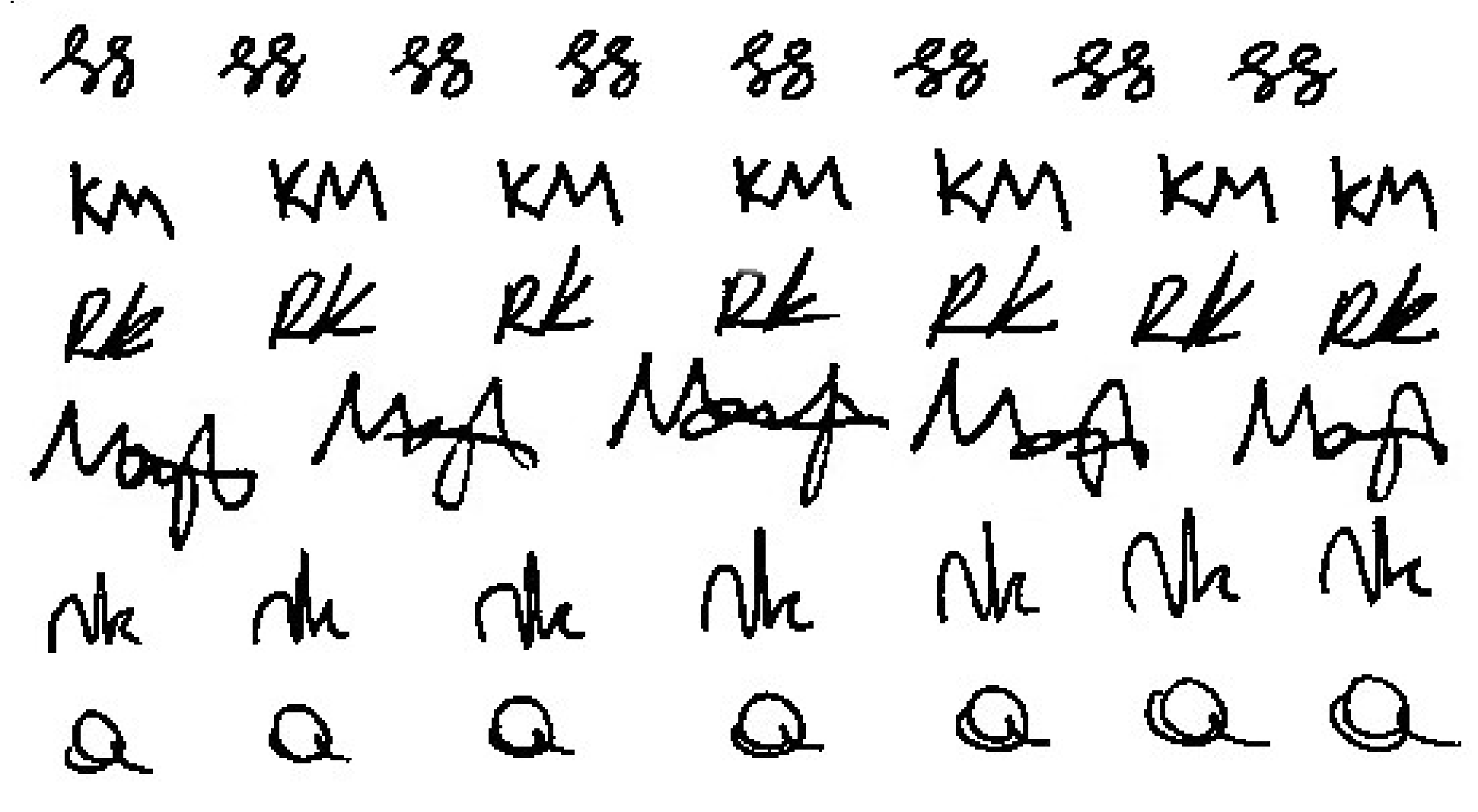


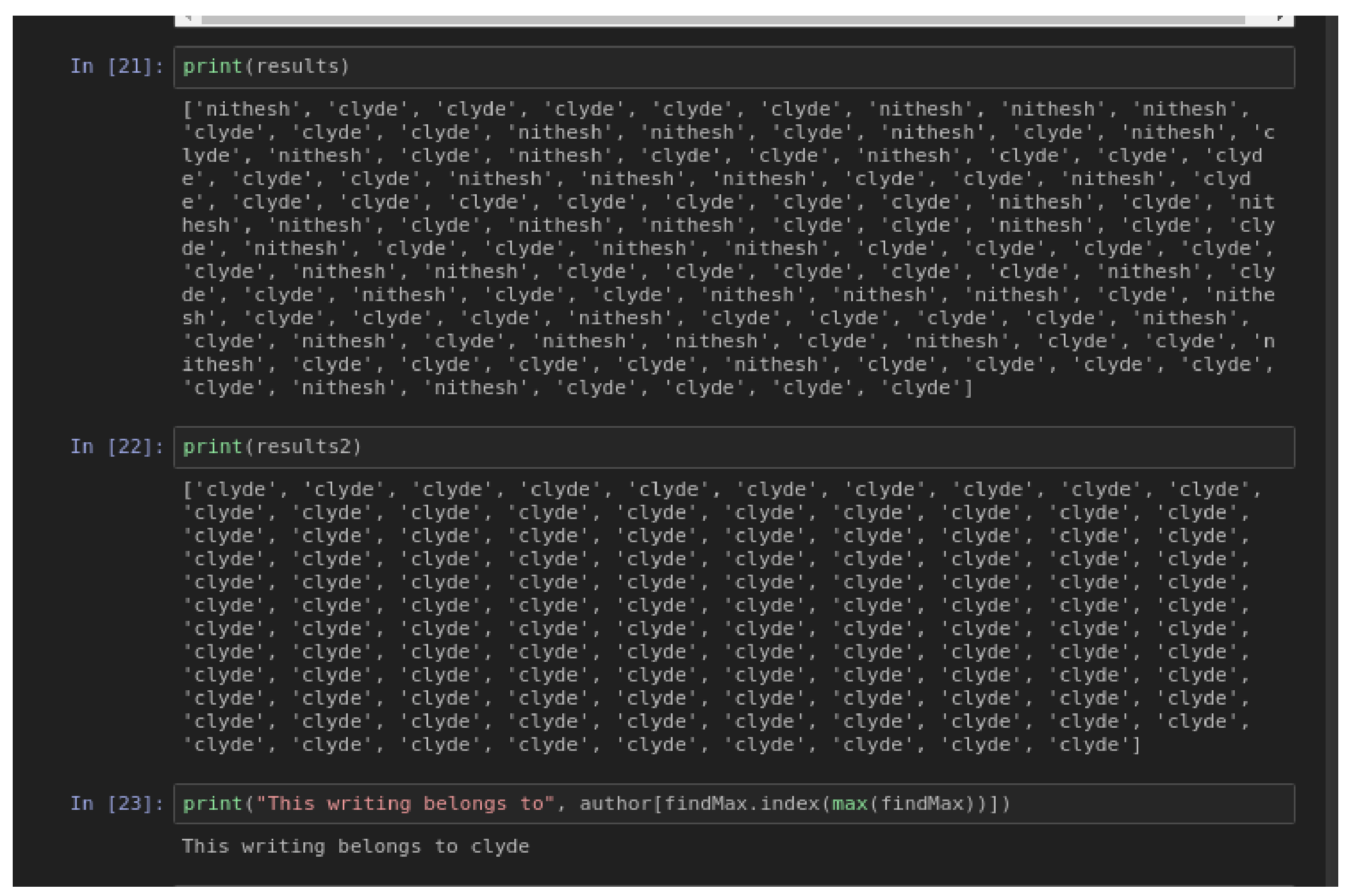
Figure 9

4.4 Building a writer identification system;
This task was challenging as it requires the system to identify whose handwriting belongs to whom. This combines both the handwriting and the signature task together.

The technical issues and challenges that I am faced with is that if we have two authors with similar handwriting it becomes difficult to set them apart and identify the author of a piece of writing. Another issue that my model faces is the number of training samples. Because I have an uneven spread of training sample for each author, the results can be quite biased.

From figure 14, we can see 4 folders; clyde, nithesh, rizwan, sachin. Together we have 9343 training samples however when we look at each folder's content. We see that 'clyde' has 2995 training samples, 'nithesh' has 3132 training samples, 'rizwan' has 1889 training samples and 'sachin' has 1327. We see here that the training data is uneven and thus will over fit our data since the model will train mostly on clyde and nithesh. Furthermore its possible for the system to get confused since the handwriting could be quite similar. The results are shown in figure15, figure 16, figure 17, figure 18

From the results shown in the figures, we can see that the model is over fitting. We are always seeing a predicted response of either nithesh or clyde and never sachin or rizwan. Again the reason for this is because of the uneven training samples and also the number of authors that we have. There is more of a chance for some of the characters to be quite similar.



```
In [21]: print(results)

['nithesh', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'nithesh', 'nithesh', 'nithesh',
'clyde', 'clyde', 'clyde', 'nithesh', 'nithesh', 'clyde', 'nithesh', 'clyde', 'nithesh', 'c
lyde', 'nithesh', 'clyde', 'nithesh', 'clyde', 'clyde', 'nithesh', 'clyde', 'clyde', 'clyd
e', 'clyde', 'clyde', 'nithesh', 'nithesh', 'nithesh', 'clyde', 'clyde', 'nithesh', 'clyd
e', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'nithesh', 'clyde', 'nit
hesh', 'nithesh', 'clyde', 'nithesh', 'nithesh', 'clyde', 'clyde', 'nithesh', 'clyde', 'cly
de', 'nithesh', 'clyde', 'clyde', 'nithesh', 'nithesh', 'clyde', 'clyde', 'clyde', 'clyde',
'clyde', 'nithesh', 'nithesh', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'nithesh', 'cly
de', 'clyde', 'nithesh', 'clyde', 'clyde', 'nithesh', 'nithesh', 'nithesh', 'clyde', 'nithe
sh', 'clyde', 'clyde', 'clyde', 'nithesh', 'clyde', 'clyde', 'clyde', 'clyde', 'nithesh',
'clyde', 'nithesh', 'clyde', 'nithesh', 'nithesh', 'clyde', 'nithesh', 'clyde', 'clyde', 'n
ithesh', 'clyde', 'clyde', 'clyde', 'clyde', 'nithesh', 'clyde', 'clyde', 'clyde', 'clyde',
'clyde', 'nithesh', 'nithesh', 'clyde', 'clyde', 'clyde', 'clyde']

In [22]: print(results2)

['clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde',
'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde',
'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde',
'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde',
'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde',
'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde',
'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde',
'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde',
'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde',
'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde',
'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde',
'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde', 'clyde']

In [23]: print("This writing belongs to", author[findMax.index(max(findMax))])

This writing belongs to clyde
```

Figure 17

5. Conclusion
To conclude all the models that I have implemented all predicted very similar predictions and all make quite similar mistakes.

I believe that I should have experimented with different feature extraction techniques such as HoG and tried some unsupervised learning to see the outcome and compare the results with my supervised learning.