AQA

A-level Computer Science

NEA: The Practical Project

Investigation on Image Processing Methods and how machines can learn to recognize hand-written English text

Candidate Name: Jeffrey Li

Candidate Number: ####

Centre Number: #####

Contents Page

Contents

Investigation on Image Processing Methods and how machines can learn to recognize hand-written English text

1. ANALYSIS

1.1. Identification of investigation

I have always had an interest in recognition of text by machines because I was enthused by an incident that occurred in 2013: when captcha was first announced to be defeated by AI made by Vicarious. After that, I have always wanted to do an investigation project related to computer vision.

'Computer vision is an interdisciplinary scientific field that deals with how computers can be made to gain high-level understanding from digital images or videos.'[1] Hence, the scheme of my project would be how machines can learn to do optical recognition on images of hand-written English text or any text written in English alphabet as well as various advance image processing methods.

In addition, I am conducting this investigation project to gain more clarity and consolidate my learning on image processing methods and deep learning.

Reviewers/users of this project would be able to follow the tutorial approach provided to develop a similar project, which should be useful for their own research and learning.

The main difficulty of recognizing handwritten text using machines is the fact that handwriting can be of very poor quality or illegible, for example, the typical "doctor's handwriting" which makes the process of capturing meaningful representations challenging as they are even hard for humans to read. Another example is that text can be written by the pen of low-grade quality, which makes the written letters extremely faint hence increasing the difficulty to recognize the text.

In addition, the process of separating individual letters out from handwritten text can also be dreadful, as the machine couldn't tell whether the 'letters' connected is just one single letter or are two letters, which may lead to the program crashing.

A general solution like comparing the input image with a pre-stored dataset of handwritten text wouldn't work and is very inefficient, for the following reasons:

Firstly, the program needs to loop through the entire dataset and compare the images one by one, which takes up a lot of time and processing power.

Secondly, even if the images are only slightly different. As an example, if the image of the letter has an extra spot of ink on the paper, the program would not recognize the letter or text as being the same, and resulting in not having any solution.

Therefore, I believe that machine learning especially deep learning will be very handy in this situation as it can be used to recognize certain characteristics within the letters, which allows the machine to recognize them without being explicitly programmed to do so.

[1] Wikipedia Definition of Computer Vision

Investigation on Image Processing Methods and how machines can learn to recognize hand-written English text

Dataset of images of different letters, and of different representations (font/handwriting style) within the English alphabet will be fed into the training loop. This allows the machine to learn specific patterns within each letter so that the output can be correctly mapped to the correct letter class.

To conclude, my investigation will base mainly on deep learning algorithms in Python 3. Specifically, in how machines can learn representations through breaking down the handwritten text and to be able to recognize them or make correct predictions on the input data.

1.2. Overview of Research

As an overview for the more detailed following section, the research of my investigation project will be done mainly using secondary research, which involves research using the internet, using books, websites[2] and using YouTube videos.

The reason for this is because this investigation project is not created for a specific client hence it would not be sensible to conduct primary research (research that involves an end client or target market through interviews). However, testers will be needed as explained in the introduction, in order to evaluate my final program.

The research began by learning the basics of Deep Learning in Python and their basic operations, as well as potential algorithms that I could use for pattern recognition and classification techniques of images of handwritten text using the book.

One method found was k-NN or k-nearest neighbours' algorithm, which at its basics is a type of instance-based learning.

k-NN finds the nearest 'neighbour' of the input instance within the instances seen in training, each of which will have a vote of what class the input instance belongs in.

Another method found was Recurrent Neural Network, which takes in two inputs, one of which is the current input, and the other is the output from the previous operation. It iterates through the elements while keeping the memory of what it has seen previously. For detailed information in both methods, please see section 1.5.

However, I believe that these are not the most suitable method for pattern recognition due to the existence of Convolutional Neural Networks[3] (ConvNet). This is a class of deep learning which accepts arrays of pixel values of the image in tensor form as input to the network The artificial neural network consists of many layers that will carry out feature extraction, and at the end, fully connected layer will attempt to predict the class that the image belongs in. For further explanation please see section 1.3.1 and 1.3.2.

[2] See section 1.7.1
[3] See section 1.7.2

The ways of implementing ConvNet in python can be found in the book and websites as well, but to reinforce my understanding, YouTube videos[4] are also useful resources to have a deeper look at Keras (neural-network library written in Python by Francois Chollet), and how it can be used to create the ConvNet.

Furthermore, research about image pre-processing methods are also needed in order to carry out operations on raw inputted image. This will require the use of OpenCV in python, which is a library useful for image editing[5]; this library can also be used for many image operations that will be useful, for example increasing contrast, scaling of the image and so on.[6]

In addition, example codes of how hand-written text recognition can be achieved in Python found on GitHub will also be very useful to give me more ideas on modules in python that can be used for deep learning as well as the ways that these modules are used.[7]

Moreover, my Computer Science teacher Ms Biletchi will supervise this investigation, and she will give me various bits of advice in optimizing my code and providing feedback on potential areas that I could improve on.

In conclusion, after carrying out researches in various aspects of the problem, objectives of my investigation project are ready to be set. Further research may be required as I start to program the solution to gain a better understanding of how the concepts researched can be implemented into python.

1.3. Detailed Researched Background

- 1.3.1 About deep learning

Before going on about ConvNet, it is very important to understand what machine learning and deep learning is, along with what gives machines the ability to 'learn'.

With classic programming, the programmer programs explicitly what the machine should do with the input data and expect the machine to do exactly as what it was told. An example of this would be a calculator where the instruction is to do various arithmetic operations upon the numerical input data.

However, for machine learning, the data and the expected answers are fed into the machine, and it is expected to come up with patterns or rules to "transform" input data into the expected answer. This is achieved through a series of training. Essentially the machine is learning representations or ways of looking at the input data that is useful for the current problem so that it can be easily solved.

[4] See section 1.7.3
[5] See section 1.7.4
[6] See section 1.3.3 and 1.3.4
[7] See section 1.7.5

Investigation on Image Processing Methods and how machines can learn to recognize hand-written English text

Deep learning is a subfield of machine learning, which relies on layers of increasingly more meaningful representations of data that are learned through feeding the various layers with training data.

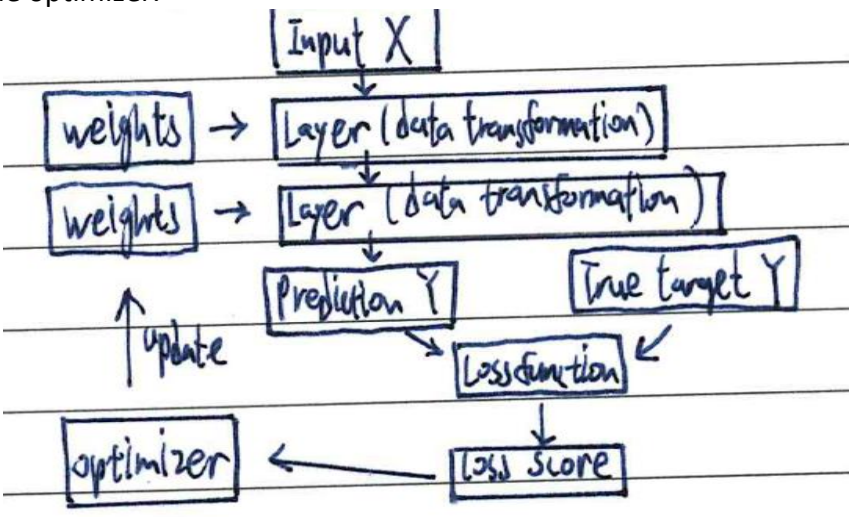
With the several layers stacking over each other (making it “deep” hence the name deep learning), they perform some data transformation which distils the input data and output something useful for the task. For example, in an image classification task, by feeding images of elephants, distinctive characteristics like big ears and long nose are distilled and extracted out, which helps makes it easier for the machine to make predictions.

Each connection between layers within the deep learning model will have a weight (numbers) associated with them, which specifies what the layer does to its input data and act as parameters. In this case, learning means finding the right value of weight for each layer within the model, so that the input value can be correctly mapped to the expected output/label. Each unit within the layer would also have biases which are an additional number added to the calculation.

As a simple example, if there is a layer of 400 units fully connected to another layer of 120 units, there would be 400 * 120 weights as there are 400 * 120 connections, and 400, 120 biases respectively. In simple terms, at each layer, the input is being manipulated using the equation: relu (dot product (W, input) + b), where W represents weight, b represents bias and relu is an activation function that will be talked about later on.

However, weights and biases of the network will be initially assigned some random numbers, which obviously means that the model will make predictions that are nowhere close to the true value, hence a mechanism of improving and updating the weight is required. Here is where loss function and optimizers kick in.

The job of the loss function is to compute a distance score/loss score between the current prediction and the true value, and this score is used as a feedback signal to adjust values of the weights a little in the direction that would lower to lose score through backpropagation algorithm, which is implemented in the optimizer.

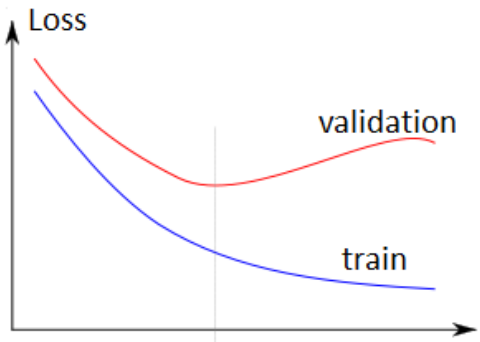


Learning Steps

Batches of training data and training labels (expected output, each having a one to one correspondence with the training data) will be fed into the model within a training loop. Note that labels need to be in arithmetic form so that they could be processed, hence labels such as 'Cats' or 'Dogs' need to be converted into some arithmetic form before they could be fed into the model

A subset of that batch will be separated out which act as validation data. This allows the model to be trained and validated on data that it has never seen before at the same time. With enough amount of iterations, the weight values will essentially be adjusted to some value, which has a minimal loss score, hence will give the best predictions.

Another key point to mention is the concepts of underfitting and overfitting. These describe how a model that does better on training data doesn't necessarily do better on data that it has never seen before, hence model that has the least loss score / highest accuracy doesn't mean that it will be the same case when being given a brand-new batch.



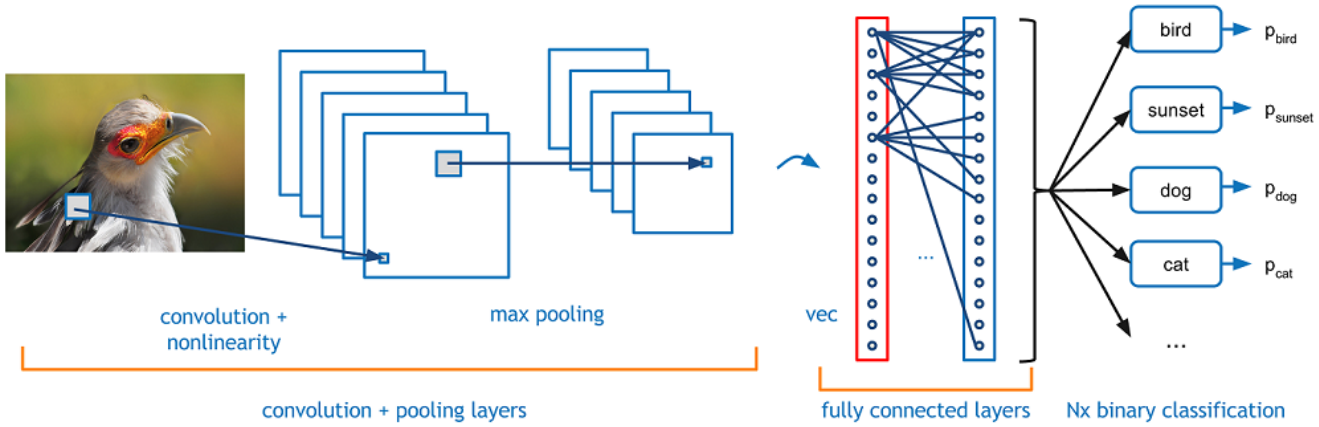
Source: https://medium.com/randomai/another-look-into-over-fitting-33e15b044a5e

As shown above, although training loss score is at its lowest towards the end of the training loop, which should be what we are expecting, the validation loss score seems to be lowest at earlier iterations. This is an example of overfitting as the model is being 'over-trained', hence would not be able to make the most accurate prediction.

On the other hand, we do not want the model to be 'under-trained' with a very little amount of training loop, as this would cause underfitting resulting in a model that has neither low training loss score nor validation loss score. The optimal situation would be when validation loss/accuracy is at its lowest/highest point when training loss/accuracy is as well.

- 1.3.2 Convolutional Neural Network[8]

[8] See section 1.7.6



Source: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

Convolutional Neural Network is a class of deep neural network (an artificial neural network that consists of many layers between input and output layer) and is a useful way of image classification, and maps the input data to specific classes/labels.

In my investigation project, the model made is mapping individual images letters, segmented out from the image of hand-written text into their specific alphabet classes. This is achieved as shown in the hand-drawn picture and the explanation below.

Firstly, the input image of letter 'X' is reshaped into set dimensions (e.g. 28 * 28 pixels), converted to grayscale and fed into the artificial neural network's input layer (meaning that shape of input tensor will be 28 * 28 * 1 which would be discussed in sections 1.3.3 and 1.3.5).

Specific characteristic such as the diagonal lines (attained through training) within the letter 'X' will act as filters of fixed dimension (e.g. 3*3 pixels) to filter the image. For example, the diagonal lines and the centre square will be one of the filters of letter X. Each dark pixel represents a one and each white pixel represents minus one in the current scenario.

As these filters go through the image, the numbered pixels in the filter will be multiplied by the pixel value of the original image, hence, if the characteristics match, 1 * 1 = 1, and if the characteristics do not match then 1 * -1 will return -1.

After the multiplication, all numbers within the filtered region are added up and divided by the total amount of components within the region, which means that if all features match, the filter will finally become a one.

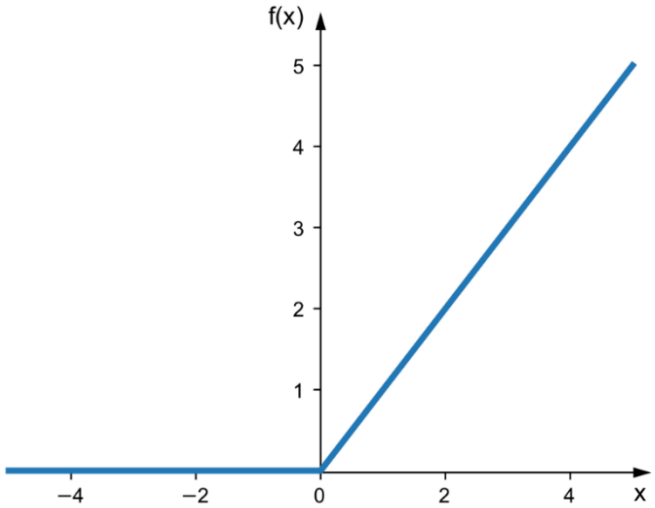
However, if some features do not match, the filter becomes a lower number or even becomes negative as shown. This filtering process is repeated to every possible position across the entire image and for different filters, which finally produces convolution or filtered image of the original one.

This entire process is done within the convolution layer, and no matter where the feature is within the image, whether the image is distorted or upside-down, it can still be extracted.

The next layer or sometimes called the activation function, called RELU is applied to turn all negative numbers into zero and other numbers remain the same, this is done so that the data is easier to handle with, as negative numbers are not significant in this case.

Furthermore, relu is an activation function as said before, which introduce non-linearity to the model, as it returns 0 for all negative values and keeps all positive value the same. This allows predictions made to be more interesting as, without this function, the model made would only be a linear regression model as only linear transformations can be performed at each layer, hence unable to learn and perform tasks that are more complex.

Continue on next page

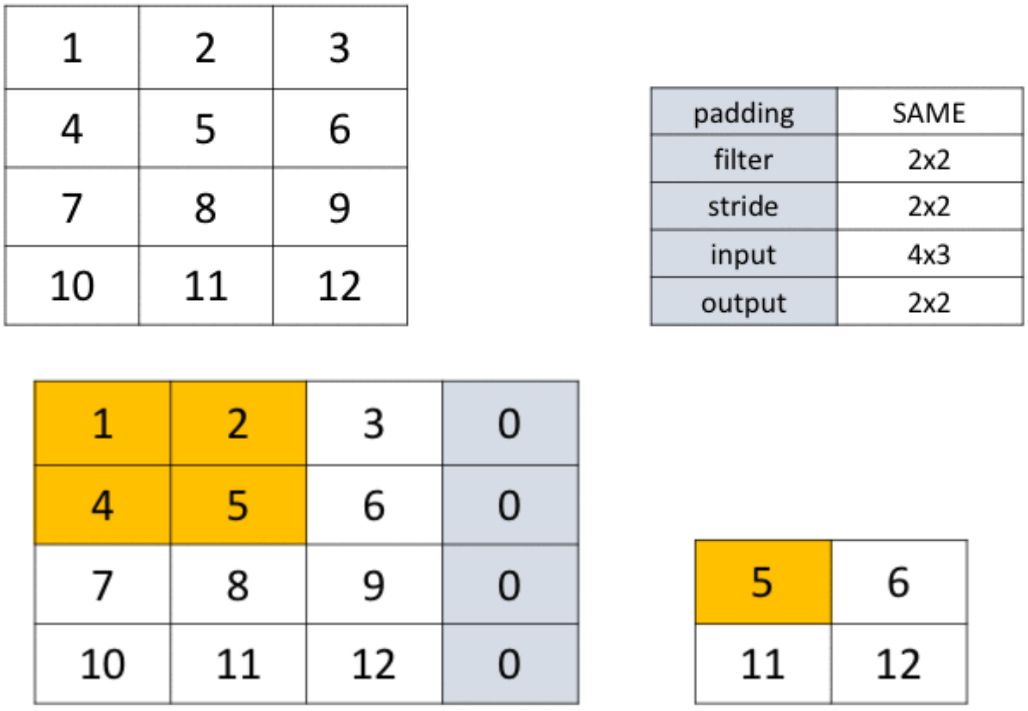


Source: https://sebastianraschka.com/faq/docs/relu-derivative.html

Afterwards, the max-pooling layer applies a window of set size (e.g. 2 * 2 pixels) to the convoluted image passed down by convolution layer, and runs through the image in every possible position, taking the maximum number within the window. The outcome will be a shrunk image of the convoluted image, potentially half as big.

For both convolution and max-pooling layer, there will be strides associated with the filter/window. A stride of x means that the window will move x pixels across every time. This introduces another problem, which is that the filter/window might move out of the picture, hence, padding of images is needed to avoid this problem.

In simplest terms, padding means adding additional pixels/values to the border of the image, so that the filter/window wouldn't go outside of the image. For my project, I will be using the SAME padding, as shown below, where orange squares represent the max-pooling window dimension and the grey area represents padding:



Source: ***https://www.corvil.com/content/kb/31-what-is-the-difference-between-same-and-valid-padding-in-tf-nn-max-pool-of-tensorflow/***

Initially, the image is of size 3 pixels by three pixels with a filter size of 2 pixels by 2 pixels and a stride of 2. Without padding, the filter would start off alright as shown above, however as it moves along by 2, there would be 2 'null' values within the filter which could lead to an error. Hence the grey areas are added to overcome this issue.

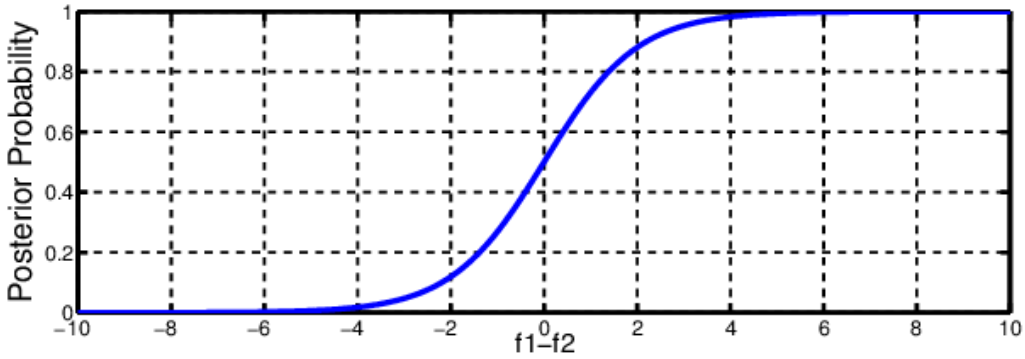
The layers described above can be stacked upon one and the other, with the output of the previous layer being the input of the next, which makes up the hidden layer. This is essentially creating 'deep' layers of artificial neural network. Multiple max pooling or convolutional layer can be applied to allow a more shrunk or filtered image.

In the end, the output from the hidden layer will be flattened, which means that no matter how many dimensions it has it would become one dimensional. As an example, if the output tensors have shapes 10 * 10 * 64, then as it gets flattened, it would have shape 6400 * 1. This would then be fed into a fully connected layer (e.g. 6400 units) as shown in the image below.

This layer is fully connected to another fully connected layer of a smaller number of units (e.g. 120 units). Each line connecting the layers will have weights associated with them and each unit within the layer will have a bias as explained in section 1.3.1. This final fully connected layer would use softmax as activation rather than relu.

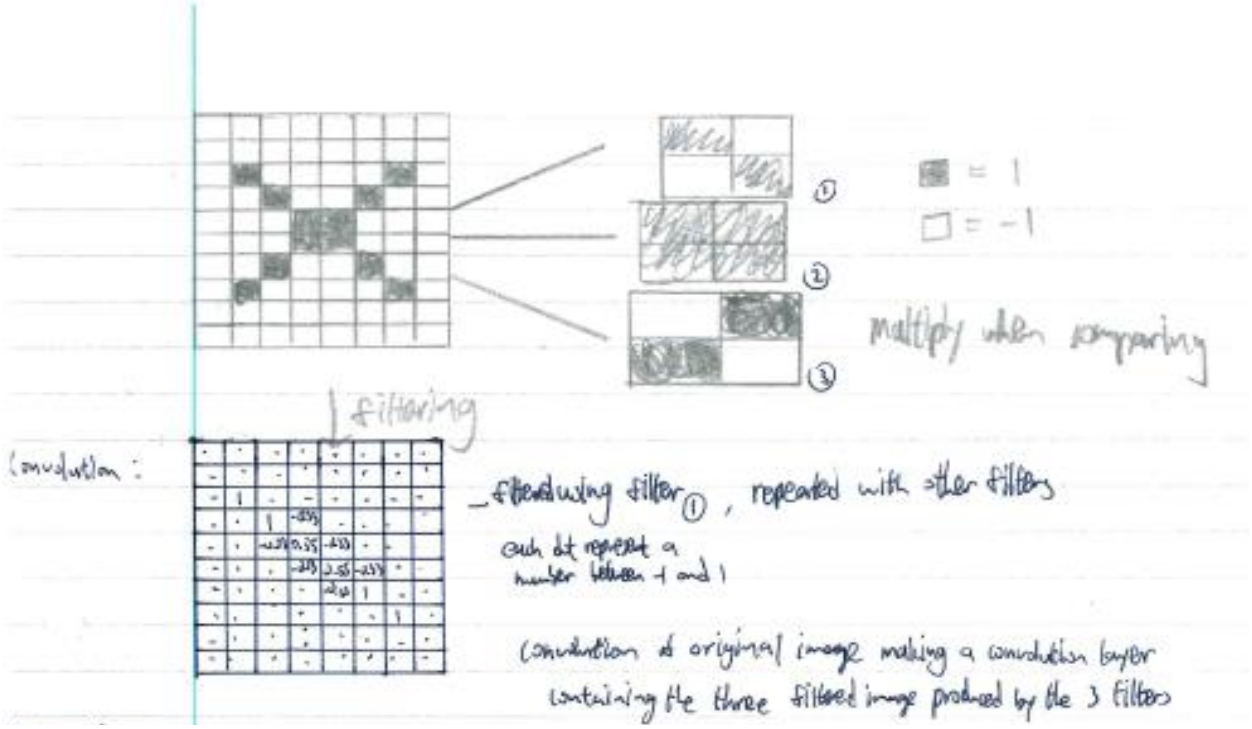
Below is the graph of softmax which returns probability values between 0 and 1 and with 26 outputs as there are 26 letters in the alphabet. In the current case, the output for letter 'X' would

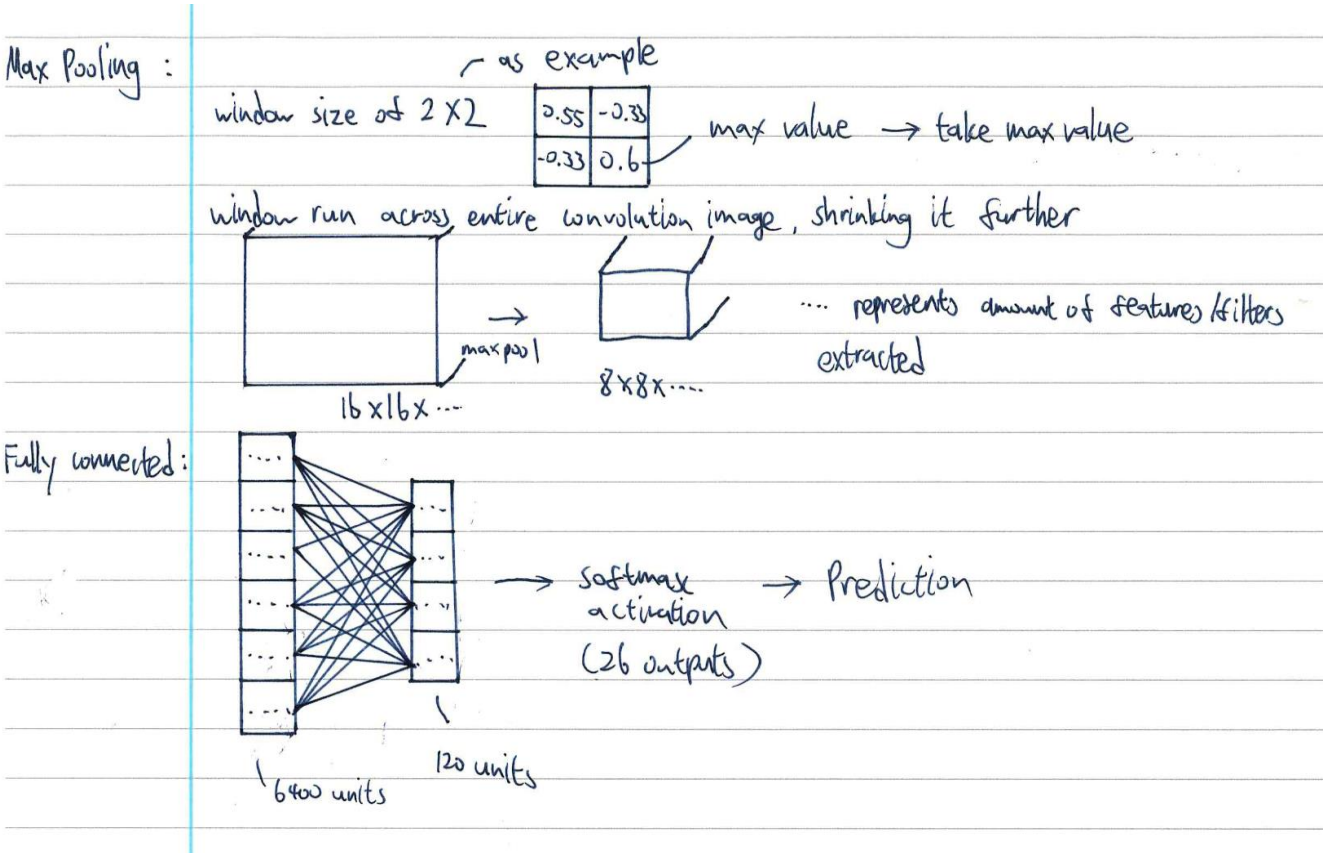
optimistically have a relatively high probability compare to other letters hence the model would predict the input image is the letter 'X'.



Source: ***httpswww.researchgate.netfigureSoftmax-activation-function_fig2_319121953***

In conclusion, this prediction process will be iterated multiple times throughout each letter one at a time in order to recognize the entire text within the image. The process of separating out individual letters from the text are explained in section 1.3.3 and 1.3.4.





- 1.3.3 Why OpenCV and image pre-processing?

OpenCV is a library in python, which allows many image processing to be achieved, hence making it very useful for my investigation as many pre-processing of the image inputs need to be done before they can be either mapped to a class to make a prediction or to be fed into the training loop to attain a trained artificial neural network.

There are many different aspects of image pre-processing, in the current scenario only useful operation for my investigation will be discussed.

Firstly, RGB images need to be converted into grayscale because of the massive difference in size and in the amount of data that needs to be processed. This is one of the main factors affecting efficiency while training the ConvNet. Imagine a 3 by 3-pixel grayscale image, which can be represented using arrays as shown below:

```
[[189, 189, 189],
[189, 189, 189],
[189, 189, 189]]
```

Each value, in this case, represents the grayscale value of each pixel, forming a 2D array (array of arrays) of one colour channel. However, if an RGB image is used instead, each of the elements shown

above will become an array of three elements to represent the pixel colour value, which forms a 3D array (array of arrays of arrays) of three colour channels. This increase in dimension means that the number of values stored will be 3 times of the grayscale. As colour is not very important in handwritten text recognition, it would have been a smarter decision to do the conversion to grayscale.

Secondly, the scaling of the image needs to be done as well. All images need to be in standard shape/dimensions for ConvNet to function properly. As explained previously, the filter used to filter image within the convolutional layer as well as the window used in the max-pooling layer are both fixed in size.

Hence, if the filters or windows are either attained by skewed (different dimensions) training the dataset, or are applied to input image of varied dimensions, the final prediction made would not be very accurate. In addition, by minimizing the size of the image, it would also make the processing of images by the artificial neural network faster.

Thirdly, increasing contrast and denoising of the image are used to make the details within the image more distinguishable. In the case of handwritten text recognition with a grayscale image, it makes the letters more recognizable which makes the task of image segmentation easier, as well as providing the ConvNet with better / clearer features which will act as filters for the convolution layer.

One potential method of increasing contrast is Binary thresholding which will be explained thoroughly in the next section. This is applied onto the grayscale image to convert the image into the binary form: only has either white or black colour. This means the text itself will become completely white whereas the remaining background will become completely black, hence increasing contrast.

Lastly, the segmentation of the image is the most important operation that will be applied to the image. This operation is needed to separate out individual letters within the text so that the ConvNet can handle letters within the text individually.

It would have been very inefficient and time-consuming to train the artificial neural network to recognize the whole text all at once as there are millions of possible texts in the world, hence will require a humongous dataset used for training. Therefore, by splitting the text into letters, the artificial neural network will only need to learn to recognize 26 letters in the alphabet, which requires much fewer resources and time for training.

In conclusion, most of the important image operations that will be used for my investigation project can be achieved by using the OpenCV library, hence will be used very frequently.

- ## 1.3.4 Important Image Operations

Some important image operations are worth spending another section to explain, and they are as follows:

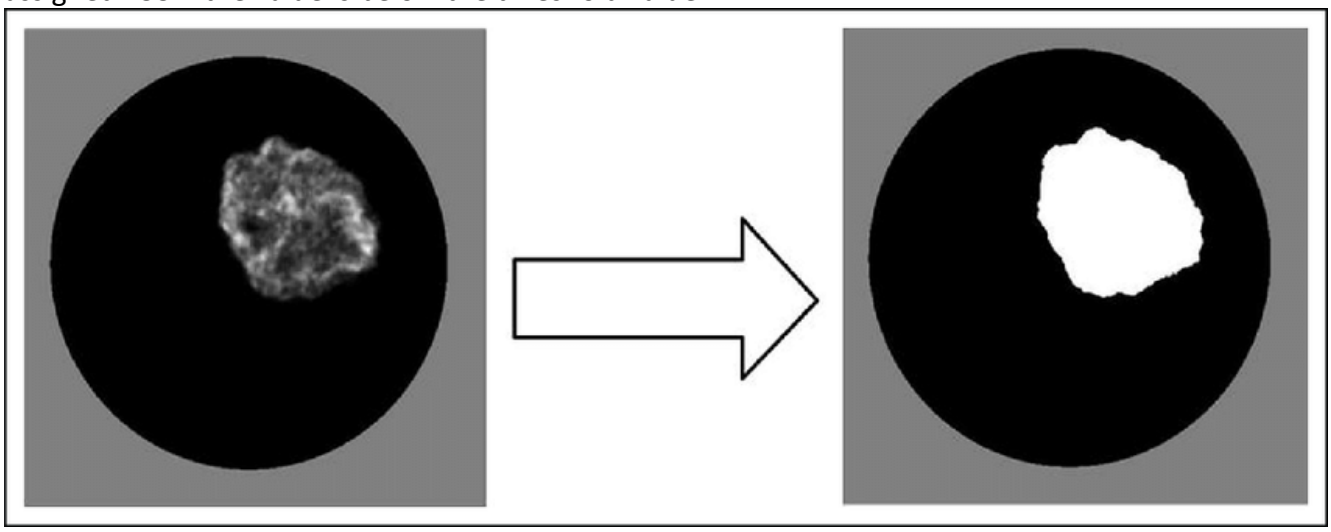
Start off with how to apply Binary Threshold, a threshold value needs to be decided which should be set so that the pixel value at any coordinate (x, y) which is supposed to be shown is greater than the threshold value in order to separate out the foreground and background sections.

An iteration structure should be used that loops through the whole image, which should have been converted to grayscale already, hence could be represented in a 2D array. Each array within the array represents one row of the image and each value stored in the array within the array represents a pixel value.

For each pixel value iterated, a selection structure is needed to compare it with the threshold value. If the pixel value is higher than the threshold value, then it will be assigned as being a black pixel with the maximum value 255, replacing the original value and if it is lower than the threshold value than the pixel will be assigned the minimum value 0 and become a white pixel.

The resulting image should only be black and white. The image below shows a grayscale image being transformed as binary thresholding is applied with a pretty high threshold value.

Wherever Inverse binary threshold is mentioned, it is essentially the same process but done oppositely, where if the value is greater than the threshold value then it would be assigned a 0, and assigned 255 if the value is below the threshold value.



Source: ***https://www.researchgate.net/figure/Flame-images-before-binarizing-thresholding-left-and-after-binarizing-thresholding_fig3_272590290***

In addition, to remove noises from the image, this requires Morphological Transformation[9] on binary form image. There are two types of transformations, erosion or dilation. Erosion meaning eroding away boundary of the known foreground, which is useful for removing small white noises around the boundary or disconnect connected objects. Dilation, on the other hand, increase the region of the foreground object.

In the current situation, the Opening operation will be used which stands for erosion followed by dilation, in order to remove noises and enlarge the shrunk letter back to original size. The Closing operation, which stands for dilation followed by erosion, will be used as well in order to remove dots

[9] See section 1.7.10

or impurities on the letters. In addition, a structuring element which basically is a matrix consisting of 0s and 1s is also needed to probe input image.

The images below show the transformations described above, with the first image showing erosion and dilation, second one showing closing and the third image showing opening.

Source: ***https://docs.opencv.org/trunk/d9/d61/tutorial_py_morphological_ops.html***

Conclusively, the watershed algorithm[10] along with the distance transform operator[11] is used upon the image after applying the inverse binary threshold and denoising, in order to segment individual letters out from the handwritten text.

Firstly, areas that the program can be sure to be background and areas where the program can be sure to be the foreground, as well as regions that are unknown for the time being, are found.

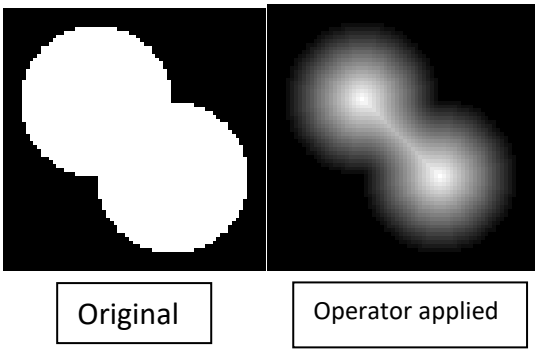
[10] See section 1.7.9

[11] See section 1.7.11

The sure background area is found by using the dilation morphological transformation mentioned previously, which enlarges every single letter within the text so that the remaining black area can be certain to be the background.

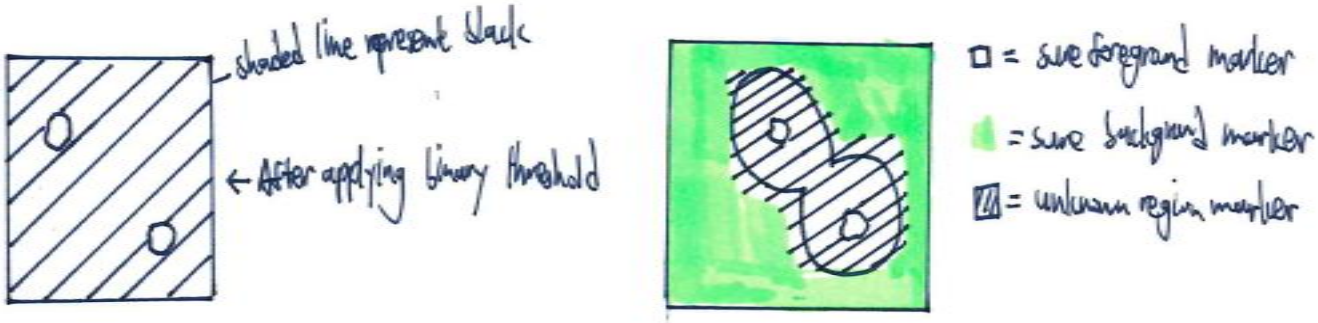
Sure foreground can be found using two different methods. The first one is obviously using erosion morphological transformation, which erodes away boundaries so that the remaining object can be sure to be of interest. However, this method would only work when objects within the image are not connected to each other, so erosion won't be sufficient for my project. The other method is the distance transform operator and applies a threshold.

The operator changes the values of pixels within the foreground region to distance their respective distance from the closest boundary, which essentially creates a 'Grey-scale image' as shown below. Then, a binary threshold can be applied to the resulting image from the operator that completely detach the connected objects, extracting sure foreground.



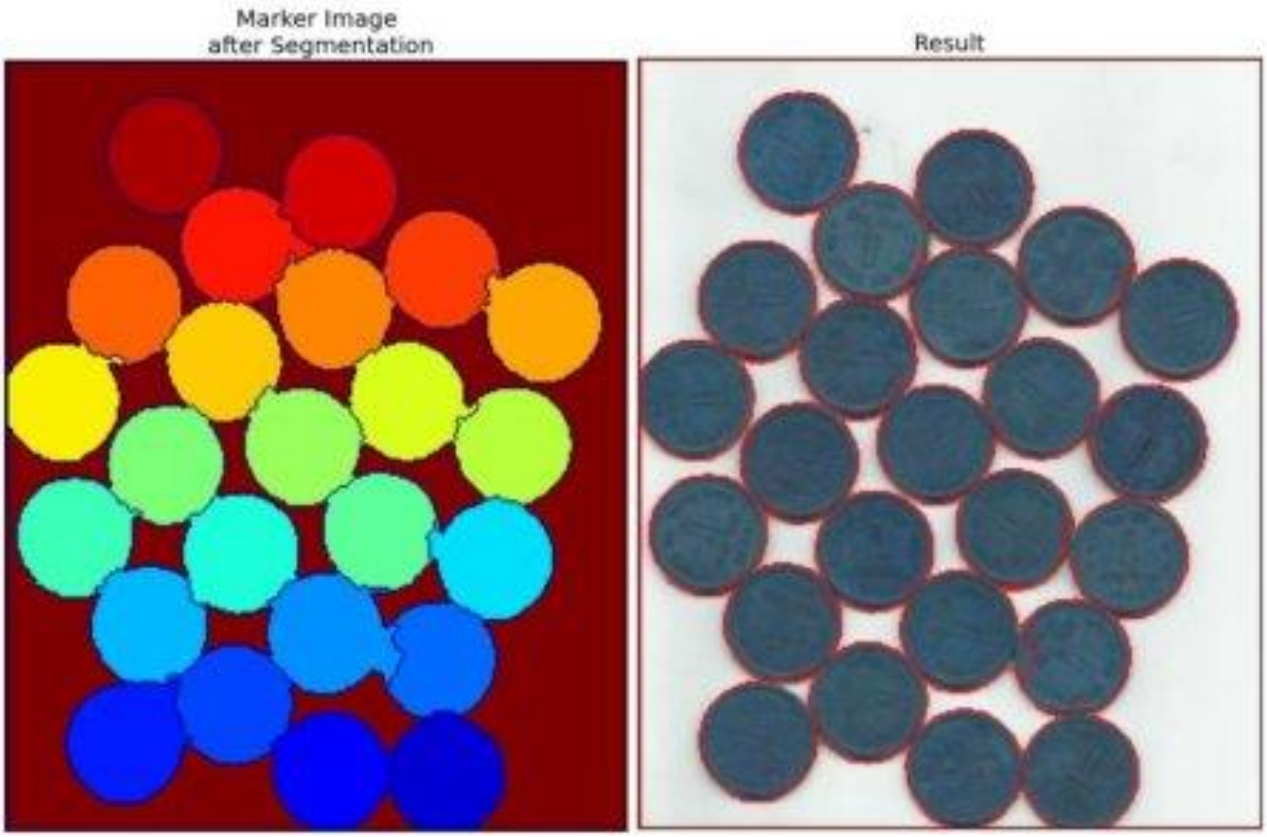
Source: https://www.mathworks.com/help/images/distance-transform-of-a-binary-image.html

For the last step, with sure fore/background extracted and unknown regions are obtained by subtracting known foreground region from known background region, sure fore/background regions will be labelled with a positive integer marker but with different numbers and remaining unknown regions are labelled as 0 or anything constant. As a result, with everything set up, the watershed algorithm will be used to properly segment out the objects, by finding their actual boundaries.



Follow on from image example shown above

An example of the resulting image would be the image shown below, with individual coins clearly separated out from each other:



Source: ***https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_watershed/py_watershed.html***

- 1.3.5 About Python

In this section, only specific use Libraries/ Modules will be mentioned and general-purpose Libraries/Modules (Random, Maths…) would not.

Firstly, is the OpenCV library as mentioned before, which will help me to process input images as well as training images. The main usage of this library is explained in detail in section 1.3.3 and 1.3.4. As this library will contain several useful operations /subroutines that can just be called and used, which will make my resulting programmed solution less complex, I may need to consider to program some of the operations completely by myself in order to make it A-Level standard.

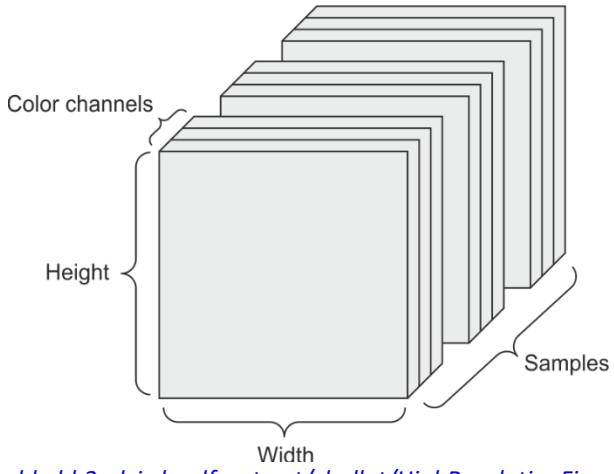
Moving onto the NumPy library, which allows the usage of tensors (a type of data structure/container of data that is very similar to arrays) and various tensor operations. Tensors are very important for the ConvNet or any deep learning model to function properly as the artificial neural network only take tensors as input.

An example of usage of a tensor is to store vector data, as clients of a bank account where typically a 2D tensor will be used. Imagine having 2 clients that use the bank, each having a bank account ID and password. This means that each tensor used to store the clients will be like (samples, person) and have a shape of (2, 2). First dimension “samples” is representing which “sample” of the client is being dealt with and the second dimension “person” will store information about account ID and password.

The first two represents the maximum number of samples that are taken and the second two represents the number of elements that each sample is storing as shown below:

[["Jeff_Li", 123456],
["John_Edwards", 654321]]

Another example, which is more related to my project, is to store samples of images where typically a four-dimensional tensor will be used as shown:



Source: https://dpzbhybb2pdcj.cloudfront.net/chollet/HighResolutionFigures/figure_2-4.png

The tensor will be consisting of (samples, height, width, colour depth) where the colour depth represents the colour channels of the image (whether it is RGB (3) or Grayscale (1)). If for example, the tensors are storing 256 samples of 30 by 30 RGB images, the shape of each tensor will be (256, 30, 30, 3) which can then be fed into the artificial neural network in smaller batches in order to train it.

Conclusively, the Keras Library is the soul of my entire project, which enables programming of deep neural networks in Python. The main usage of the library is that it acts as an API that sits on top of TensorFlow, to create a deep learning model with all sorts of layers within the model. This relies on the Sequential layer, which is like a sandbox that allows a linear stack of layers to be added.

In addition, the Dense layer (or the fully connected layer) will be used very frequently, as it is a regular layer of neurons in an artificial neural network that receives an input, do some operations on the input tensor and pass it on.

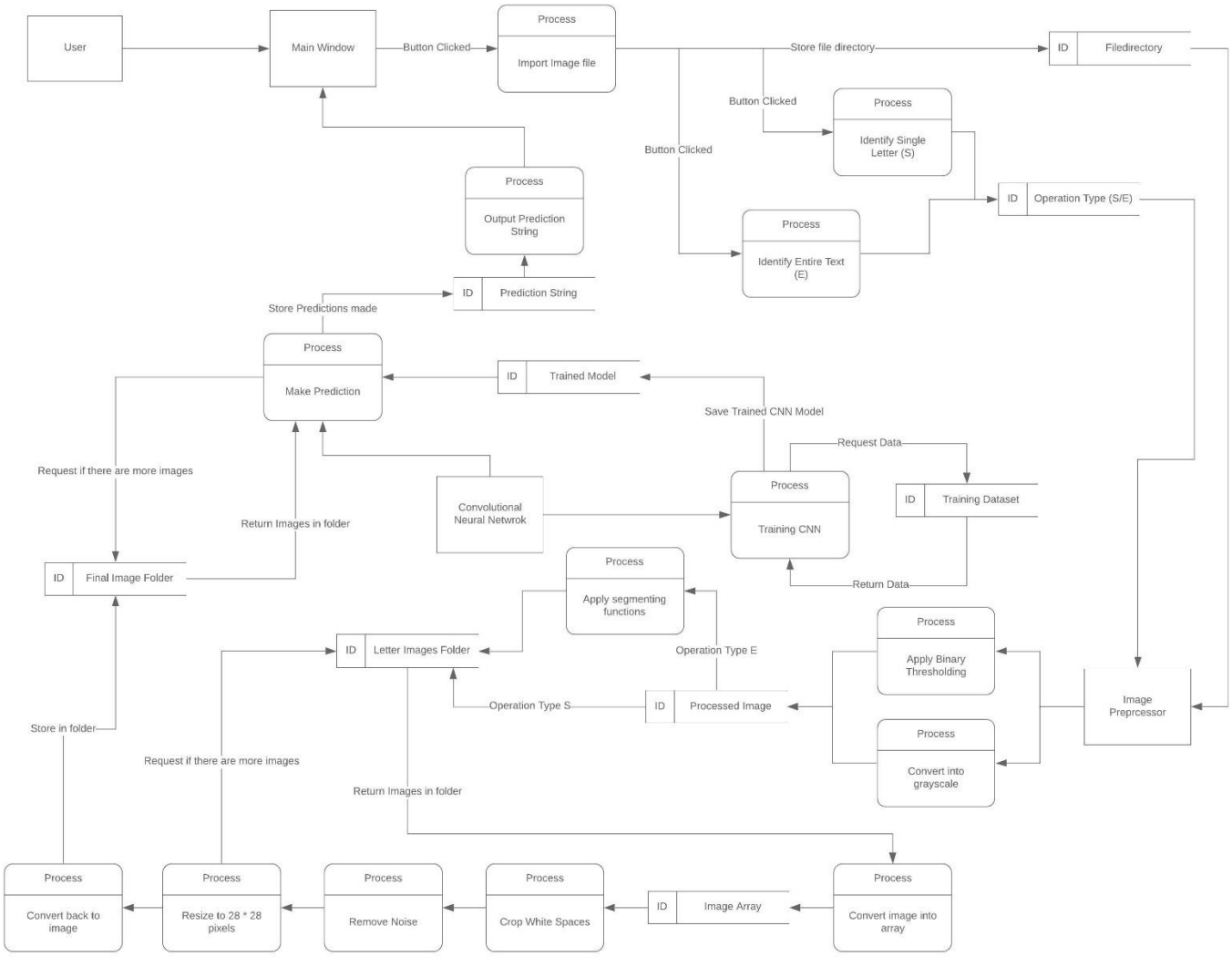
The Keras library also includes all sorts of loss functions, optimizers, activation functions (purpose of them are explained in section 1.3.1) and allows the model to be trained through a series of iterations. Nevertheless, as my project relies on the usage of the convolutional neural network, Keras also have essential layers and operations related to ConvNet as mentioned in section 1.3.2.

I have thought about using TensorFlow and already learned how to create a CNN model, however, due to the efficiency of the Keras API and with TensorFlow being extremely convoluted about saving

and importing models, I have finalised with using Keras. However, I will be using some TensorFlow code in Design section to explain my Keras code thoroughly.

- 1.3.6 DFD

Below is the DFD of my investigation project:



1.4. Objectives

- 1.4.1 General Objectives

The general objective of this investigation will be to create a programmed model, which is able to recognize hand-written text from an image and output the corresponding text onto console digitally.

- 1.4.2 Specific Objectives

The general objective can be split into four main sections:

- Graphical User Interface
- Image Pre-processing (Pre-processing of the input image)
- Creation of ConvNet Model

- Training of the ConvNet Model

1. Graphical User Interface - the program must:
    a. Have a main window that contains 3 buttons
        i. Buttons are used to choose different options:
            a. Import image file
            b. Instructions
            c. Exit
    b. User clicks the Import image file button to import image of hand-written text
        i. the main window is hidden and a file opening window is displayed
        ii. The user can search for/import image file types only
    c. Error detection method
        i. An error message is sent when the user imports wrong file type. User is asked to load the correct file type
        ii. Program doesn't exit when an error occurs
            1. The main window is re-displayed
    d. After importing the image, Operation window is displayed with more options selected using buttons
        i. An option where the user only wants to identify a single letter
        ii. An option where the user wants to identify the entire text
    e. Depending on the option selected in the Operation window:
        i. Corresponding operation type "S" or "E" should be passed into the image pre-processor
        ii. Entire text option would require the segmentation method to segment out individual letters out of the text
        iii. Both options would result in an image being pre-processed as mentioned in part 2.
    f. The main window mentioned in part a) should be displayed again
        i. After the program has produced a prediction of the previously imported image
        ii. When user made an error during any process
    g. Instruction window is displayed when the Instruction button is pressed
        i. Main window is hidden
        ii. Instruction window is displayed
        iii. An exit button is used to close the window and display the main window again
    h. Exit button displayed on the main window should terminate the program completely

2. Image Pre-processing - the program must (see section 1.3.4 for extra detail):
    a. Convert input image from RGB to grayscale
    b. Scale Image
        i. Scale all images to 28* 28 dimension to normalize them
        ii. Resized image should be of legible quality
    c. Apply Binary Threshold
        i. A threshold value is decided where the value:
            1. At any coordinate (x, y) which is supposed to be shown, need to be greater than the threshold value.

2. Greater than remaining / background pixel values

ii. A maximum value is decided which would be 255

iii. If the pixel value is greater than threshold value, it is set to maximum value (255)

iv. If the pixel value is lower than the threshold value, it is set to 0

d. Remove noise, achieved through Morphological Transformation (See section 1.3.4) and shadow removal

i. The image is transformed into binary form first (see part c.)

ii. Use the Opening transformation to erode then dilate the image

1. Transformation is achieved by calling the OpenCV library function cv2.morphologyEx()

iii. Use Closing transformation to dilate then erode the image

iv. Use shadow removal methods to remove shadow within the image

e. Segment Letters out from text (See section 1.3.4)

i. Achieved by using ROI (Region of Interest, see section 2.5.7)

1. Should be already in binary form (see part c.)
2. Extract contour lines within the image (see part f.)
3. Use extracted contour lines to draw appropriate boundary rectangles around desired regions
    a. This is achieved by using cv2.retangle()
    b. Colour of drawn rectangles has to be the same as the background (not visible)
4. Only rectangles of required dimensions are saved into folder, others are ignored
    a. Dimension need to be large enough to avoid small separated sections of the letter being identified as a different region
    b. Dimension need to be small enough to avoid chunks of unwanted are to be identified

f. Finding Contour lines (See section 2.5.6)

i. Should be already in binary form (see part c.) and cropped (see part g.)

ii. Contours are found using the cv2.findContours() function with appropriate parameters

1. Reasons of choosing the parameters are discussed in section 2.5.6

iii. Found contour lines are thickened and drawn with black colour onto original image

1. Need to exclude the contour lines drawn for boundaries of the image as these are unnecessary information

iv. Thickness of contour lines need to ensure that the letter is legible after resizing to 28 * 28 pixels.

g. Cropping of the image, which can be split into 2 operations, crop top and crop bottom. (See section 2.5.5)

i. Image need to be in binary form

ii. To crop top, program iterates through the image array

1. For each iteration, find the max valued element within that array
2. If the max value is 0 which represents a white pixel then that array will be deleted

3. If the max value is greater than a threshold value (e.g. 10) then pause the iteration

iii. To crop bottom, the same process ii is done but in reverse order

iv. Rotate the image by 90 degrees, apply the two operations described above to crop left and right

3. To create ConvNet model, the program must have (see section 1.3.2 and 1.3.5 for extra detail):
    a. Convolutional Layer which filters / convolute input image
        i. Parameters that requires consideration are the number of units per layer, number of layers needed in order to obtain the best results
        ii. The values for these parameters can be taken from well-known CNN architectures such as LeNet, VGG and AlexNet which are well tested and has extremely high validation accuracies
    b. Max Pooling Layer used along with convolution layer to achieve filtering
        i. Applies a window of set dimension to the convoluted image passed down from the previous layer
            1. SAME[12] Padding may be required
        ii. Runs through the image and take the maximum number within the window and replaces the region with the maximum number
        iii. The layers can stack upon one and another to allow a more shrunk/distilled image along with the convolutional layer
    c. Activation function (RELU)
        i. Introduce non-linearity to the model
        ii. Applied onto the convoluted image or after applying max pooling to remove/zero out any negative numbers
            1. Negatives numbers are not of interest as only features that match the filter matters
    d. Output tensor from the Max-pooling and Convolutional layer stack is flattened before being passed into the fully connected layers
    e. Fully Connected Layer takes in output of the stack of layers and output prediction
        i. The output would be a tensor
        ii. Rather than RELU, Sigmoid activation function is used for the final Fully Connected layer to enable a prediction of the alphabet class
        iii. The appropriate amount of fully connected layers needs to be obtained through testing of various models
        iv. The appropriate number of units per fully connected layer also requires consideration through various testing

4. To train the Convolutional Neural Network (see section 1.3.1 for extra detail):
    a. A big data set of handwritten letter images
        i. Images split into folders, each containing only one type of English letter but written by different people
            1. Separate upper case and lower case of same letter

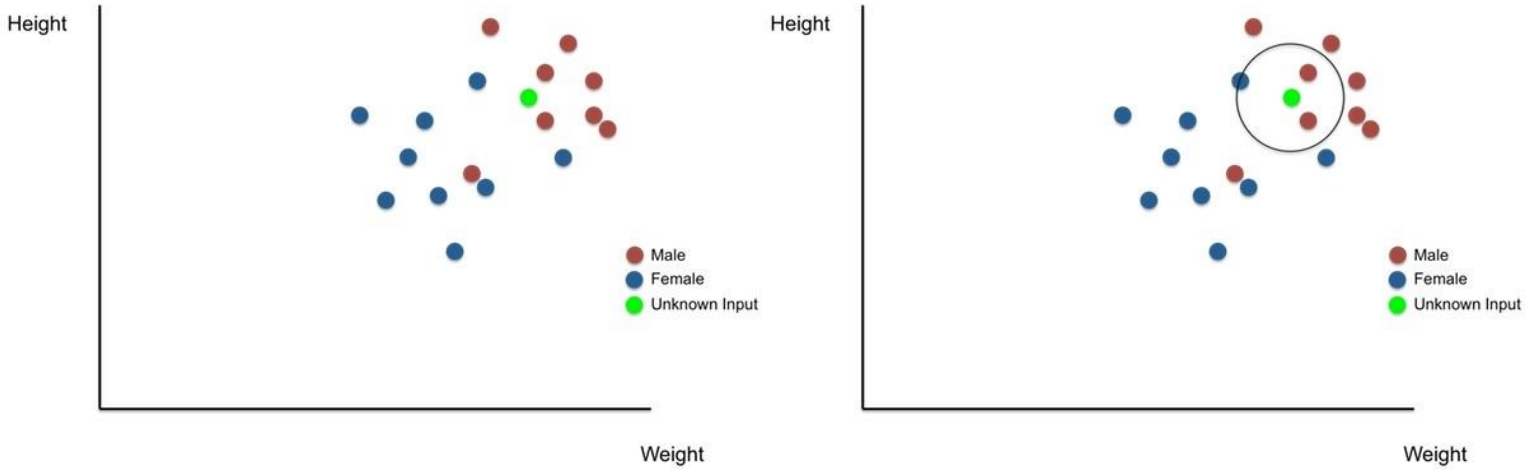
[12] See section 1.3.2

ii. Images should be of acceptable quality
    1. Not unreadable (e.g. Even human couldn't recognize the letters)
iii. Dataset that can be used is the IAM handwriting database[13], NIST handwriting database[14], or database found on kaggle[15]

b. All images inside folders are pre-processed before being fed into the CNN model
c. Labels / expected output (such as alphabet letters)
    i. Corresponding labels are 'attached' to every single image that would be used as training data
    ii. Labels are converted to one-hot encoded array
d. Processed images which are linked with their corresponding labels are saved as a pickle file
    i. Enables easy importing and exporting
e. The saved pickle files of labels and images are imported and fed into the ConvNet
    i. Fed in smaller batches in multiple epochs / training loops
    ii. Number of epochs required needs to be enough to produce good accuracy
    iii. Need to take overfitting and underfitting into consideration
        1. Achieved by using Tensor board[16] to monitor the training process
    iv. Final trained model needs to be saved into a '. model' file
f. The trained model is imported and used to create prediction on images of handwritten letter
    i. Accuracy of the model should be around 80% for it to be useful for handwritten letter image recognition task

1.5. Justification of chosen solution

- 1.5.1 K Nearest Neighbour (KNN)



Source: https://brilliant.org/wiki/k-nearest-neighbors/

[13] See section 1.7.12
[14] See section 1.7.13
[15] See section 1.7.14
[16] See section 1.7.15

KNN is a simple classification method that works well on simple recognition problems. It works by finding 'neighbours' of the input instance within the feature space which contains the training data, as shown in the image above.

The size of the imaginary circle is decided by the value of k (attained through training loop), and each member within the circle will have a vote on what class the unknown input belong in. In the example above, there are two votes on male and one vote on female; hence, the unknown input will be more likely to be a male than female.

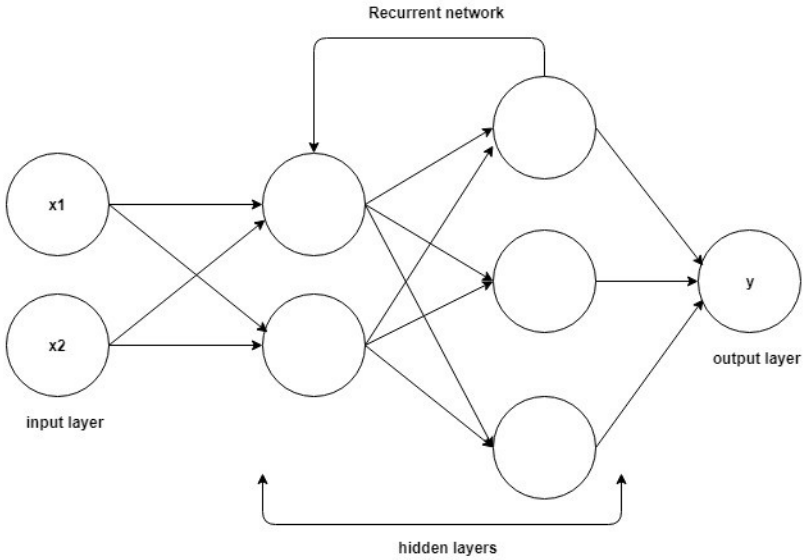
The main disadvantage of KNN is the fact that it relies too much on the training data, the algorithm itself is not 'learning' and is only comparing the data for classification.

In addition, KNN needs to compute the distance between data records at each prediction, in order to find the nearest neighbours, which can be very slow if there are a large number of training samples, making it less time-efficient while making predictions.

In conclusion, I do not think that KNN will be most suitable for this task because:

- A large number of training samples (images of handwritten texts in this case) are needed to ensure that the predictions made are as accurate as possible, however, KNN isn't very time-efficient in handling a large number of samples.
2. Images of handwritten text may be of different qualities and styles; this means that predictions made may be drastically different even if the images are only off slightly (e.g. an extra dot beside the letter ……) as KNN is only comparing the data rather than actually learning the patterns of the letters.

- ## 1.5.2 Recurrent Neural Network (RNN)



Source: https://hackernoon.com/rnn-or-recurrent-neural-network-for-noobs-a9afbb00e860

RNN is a type of Artificial Neural Network, which unlike other artificial neural networks; it has its own 'memory'. This characteristic makes it very useful when processing sequential data like processing sentences or machine translation where the previous contents will alter the following contents (e.g. I love cats / He loves cats)

RNN has an internal loop as shown in the image above which iterates through every single element within the sequence that it is processing, and maintaining 'memory' for what it has seen so far. It uses the previous output, in addition to the current input. The 'memory' will reset when an independent/different sequence is fed into the network.

Although no significant disadvantages can be found for RNN as it is described as being 'Unreasonably Effective'[17], and it certainly can be used to do image classification. However, I have a better understanding of CNN and it is a better option for computer vision tasks, hence I will be using CNN for my investigation project.

1.6. *Analysis Activity Log*

25/02/19: Brainstorm potential projects for NEA, the only idea I had was Investigation project in machine learning algorithm but not sure about which aspect of the machine learning algorithm.

06/03/19: Discussed with Ms Biletchi and decided to do handwritten word recognition with a deep learning algorithm. Ms Biletchi agreed on being the supervisor for the project. Decided on doing the NEA Project using Python.

08/03/19: Started learning about deep learning in python using the book[7], and started breaking down the problem of identifying entire word into identifying individual letters. The research was done on the internet for handwritten letter recognition. Finished the introduction section.

11/03/19: Found KNN method RNN method and ConvNet method, require more research into these methods to decide on the optimal solution.

15/03/19: Finally decided on doing investigation in ConvNet. Found videos on YouTube about ConvNet in Python, to be watched over the week, started writing up the Research section, more information to be added to this section as research goes on.

18/03/19: Created a basic ConvNet which recognizes pictures of Cats and Dogs by following the tutorial video. This method could be improved to be able to recognize handwritten letters.

19/03/19: Realizes the difficulty in handwritten word recognition, e.g. how the letters can be separated out from the whole word to be processed individually. Method of doing this to be investigated over the week.

23/03/19: Have a clear understanding of ConvNet, writing up Detailed Background section to explain the basic operation of ConvNet.

[17] See section 1.7.7

25/03/19: Found out that OpenCV library in python can be used to segment out specific parts of the image, which can be used to separate out letters from an image of handwritten text. Finished writing OpenCV section. How it should be implemented to be learned.

27/03/19: Looking through examples on GitHub on Handwritten text recognition in python for more inspiration and ideas on completing the project.

29/03/19: Had a meeting with the supervisor, and is given the following suggestions:

- The supervisor suggested finding a dataset of handwriting with a variety of handwriting styles for the training of the ConvNet
- Suggestion on an interface: options on choosing to recognize the single letter, two letters and gradually increasing the number of letters to potentially a full sentence
- Suggestion on how I can use the basic idea of ConvNet and implement it in python.
- Suggestion on collecting images of handwritten letter/word from peers/teachers/ parents, used for final testing of the algorithm

03/04/19: Finished researching and completing the research section, started working on potential modules/libraries that may be used for the project

10/04/19: Finalizing general and specific objectives, typing the objectives down into the project report.

24/04/19: Fully researched KNN and RNN method and how their disadvantages comparing to CNN for the current investigation task. Finished justification of chosen solution section.

13/06/19: Started researching potential modules and libraries that will be used for the project, and typing up the About Python section.

15/06/19: Finished about Python section, starting to work on explaining what machine learning and deep learning is and how do they work.

20/06/19: Finished about Deep Learning section, specific objectives need to be finalised.

24/06/19: Completed entire Analysis section.

1.7. Sources used for this section

[1]'Deep Learning with Python' written by Francois Chollet
'Deep Learning basics with Python, TensorFlow and Keras' available at https://pythonprogramming.net/introduction-deep-learning-python-tensorflow-keras/
[2]'Convolutional Neural Network' Available at https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/
[3]'Deep Learning with Python, Tensor Flow, and Keras tutorial' Available at https://www.youtube.com/watch?v=wQ8BIBpya2k&list=PLQVvvaa0QuDfhTox0AjmQ6tvTgMBZBEXN

'Convolutional Neural Networks' Available at https://www.youtube.com/watch?v=smHa2442Ah4&list=PLkDaE6sCZn6Gl29AoE31iwdVwSG-KnDzF&index=4

'Convolutional Neural Networks' Available at https://www.youtube.com/watch?v=ArPaAX_PhIs&list=PLkDaE6sCZn6Gl29AoE31iwdVwSG-KnDzF

[4]'OpenCV Tutorial', available at https://opencv-python-tutroals.readthedocs.io/en/latest/index.html

[5]'Example codes used for research found on GitHub, available at:

- ***https://github.com/githubharald/SimpleHTR***
- ***https://github.com/solivr/tf-crnn***

[6]'how convolutional neural networks work', available at https://www.youtube.com/watch?v=FmpDIaiMIeA&t=1049s

[7] The Unreasonable Effectiveness of Recurrent Neural Networks', available at http://karpathy.github.io/2015/05/21/rnn-effectiveness/

[8]'Weighted method or luminosity method', available at https://www.tutorialspoint.com/dip/grayscale_to_rgb_conversion.htm

[9]'Image Segmentation with Watershed Algorithm', available at https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_watershed/py_watershed.html

[10]'Morphological Transformations', available at https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html

[11]'OpenCV – Distance Transformation', available at https://www.tutorialspoint.com/opencv/opencv_distance_transformation.htm

[12]'IAM Handwriting Database', available at http://www.fki.inf.unibe.ch/databases/iam-handwriting-database

[13]'NIST Special Database 19', available at https://www.nist.gov/srd/nist-special-database-19

[14]'A-Z Handwritten Alphabets in .csv format', available at: https://www.kaggle.com/sachinpatel21/az-handwritten-alphabets-in-csv-format

[15]'Get Started with TensorBoard', available at: https://www.tensorflow.org/tensorboard/get_started

2. DESIGN

2.1. Design overview

For my investigation project, the CNN model trained will be implemented into the main program to allow predictions to be made on the user's images.

When launching the program, the user needs to click on MAIN python file, which allows the main graphical user interface to be displayed with three separate buttons:

- Instructions
- Open Image File
- Exit button

Other than the three buttons, there would be some blank space reserved which will be used to display potential error messages.

When the user clicks on the Instructions button, this allows a new Instruction window to be displayed, which contains a sequence of steps that the user should follow in order to use the program

correctly. There would also be an exit button at the bottom of the window which closes the window and allow the user to return back to main interface

When the user clicks on the Open Image File button, this would lead to a file opener being opened that allows the user to select the image that contains the letter/text that they wrote. After the file is selected, if the filetype doesn't match the pre-determined requirements, and an error message would be displayed.

If the user chose a file of correct format, a new operation window would be displayed with two additional buttons displayed:

- Recognize single letter
- Recognize entire text

Afterwards, the image's directory would be returned which allows image pre-processing to be operated on that image.

To do image pre-processing, the main program would import functions defined in the Image-Pre-processing python file within the same folder, with operations as discussed in the Analysis section.

If the user selected the Recognize entire text option, image segmentation functions will be used to separate the entire image into separate individual images of letters to allow the CNN model to make predictions on the letters one by one.

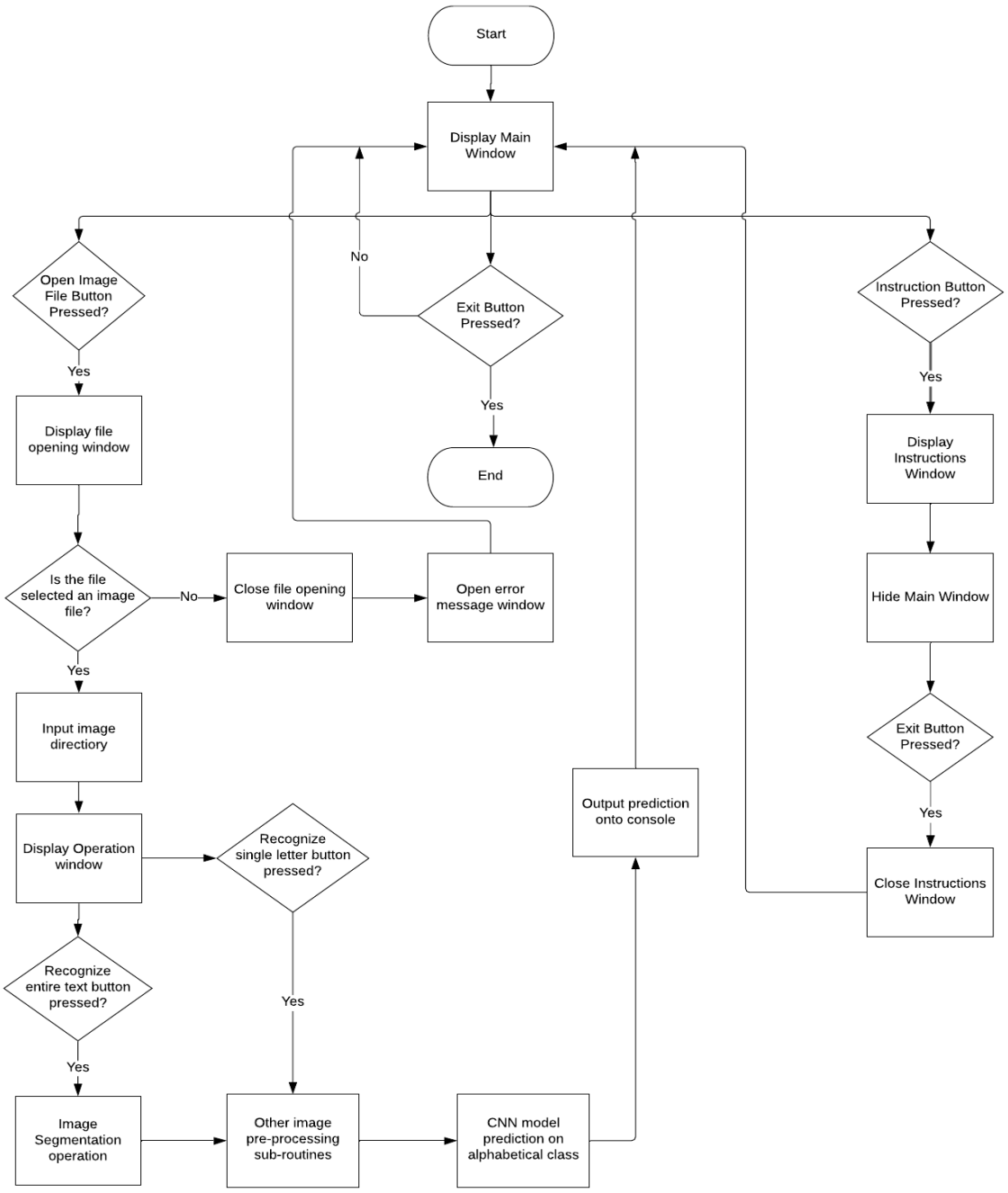
The processed image will then be passed into the trained CNN model to make a prediction on what alphabetical class does the letter belong in. In addition, the prediction will either be displayed onto console digitally directly, or assembled into sentences beforehand if necessary.

Finally, the user can continue to import images into the program until they no longer want to, and by clicking the exit button, the main program will be terminated.

Continue on the next page

Investigation on Image Processing Methods and how machines can learn to recognize hand-written English text

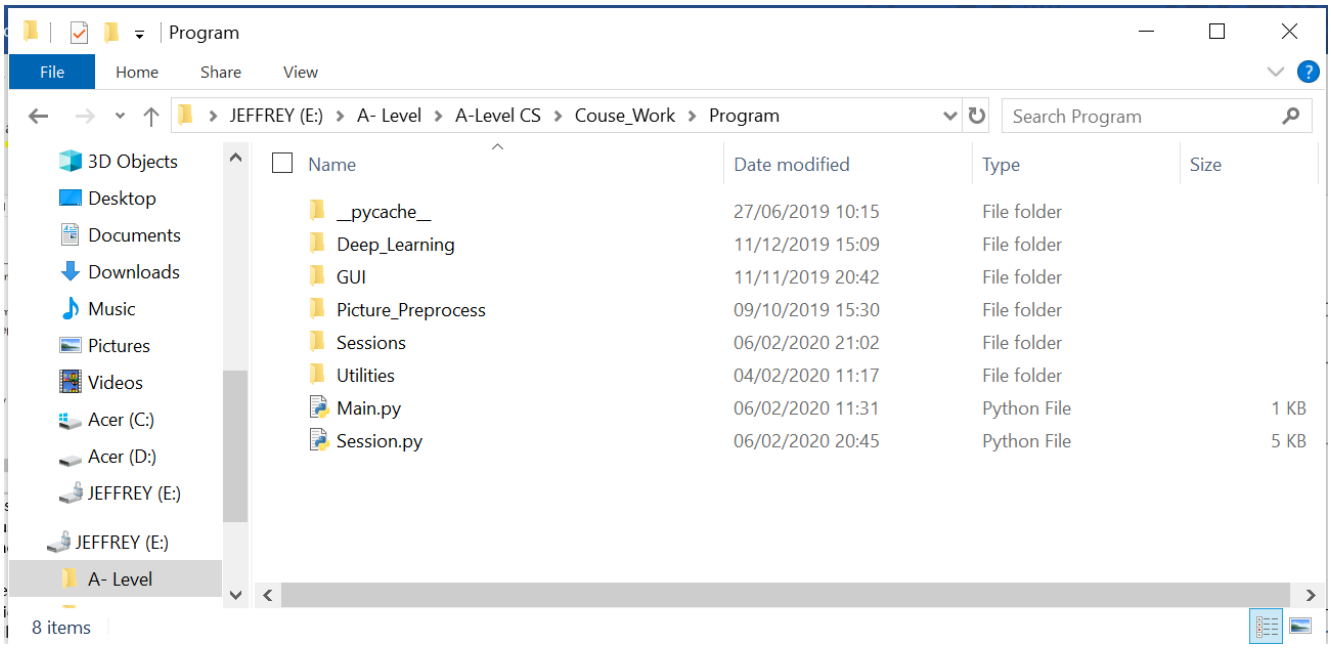
The flowchart below gives a better illustration of what would happen when the main program is launched, Start representing when the user launches the Main program:



Overall, my investigation project can be split into three sections and will be further broken-down section by section:

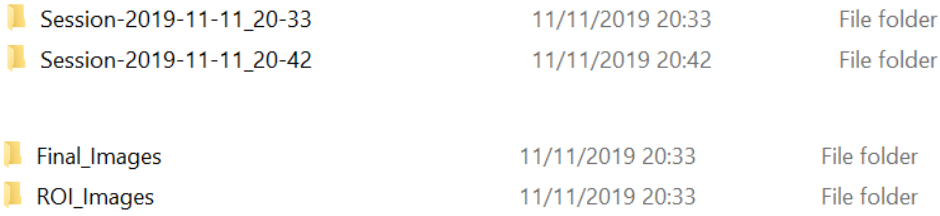
- Graphical User Interface
- Image Pre-processing
- Convolutional Neural Network Model

2.2. Folder and File Structure

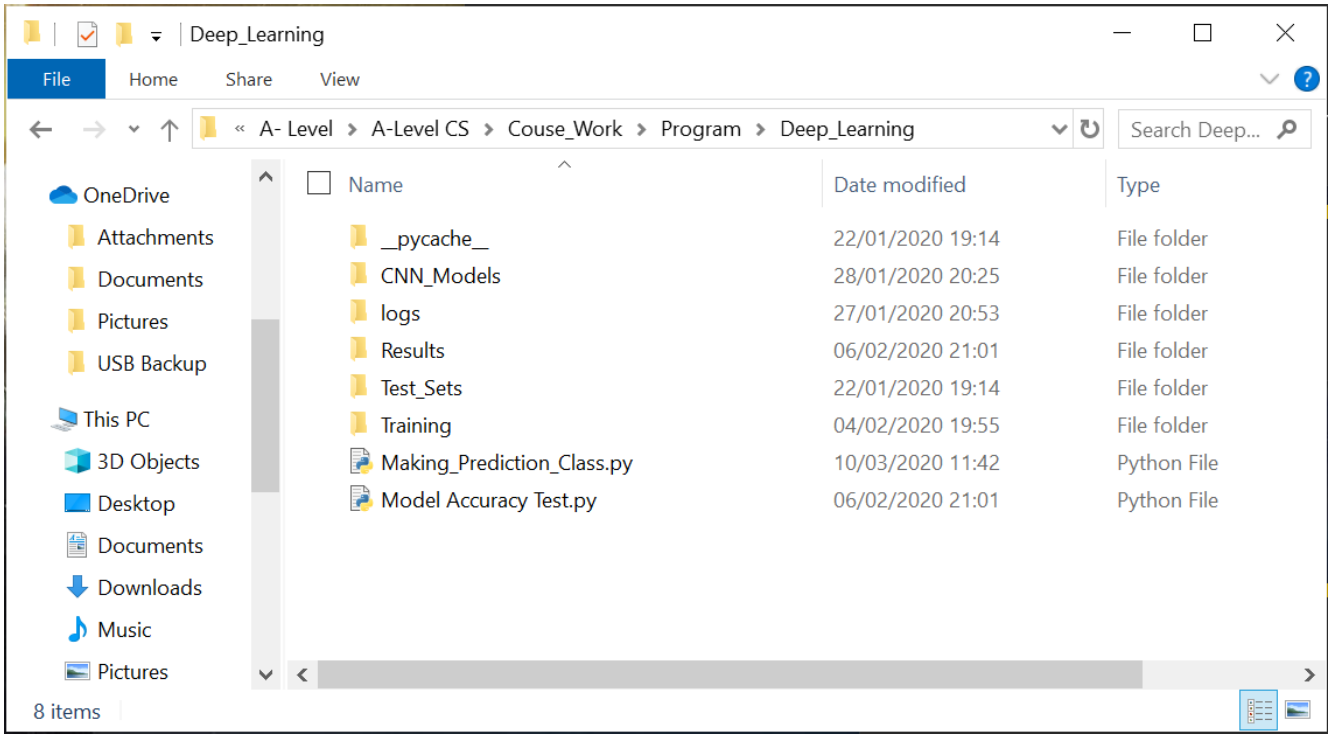


The entire program would consist of 5 main folders with a Main and Session python script and a pycache folder. The program is launched when the user clicks on the Main python file, and a 'Session' would be initiated whenever the user imported a new image.

Each 'Session' would create a new folder within the Sessions folder and named with the date and time that it's created. Each Session folder would contain 2 extra folders called ROI and Final_Images. Functionality of these folders will be explained in section 2.5.

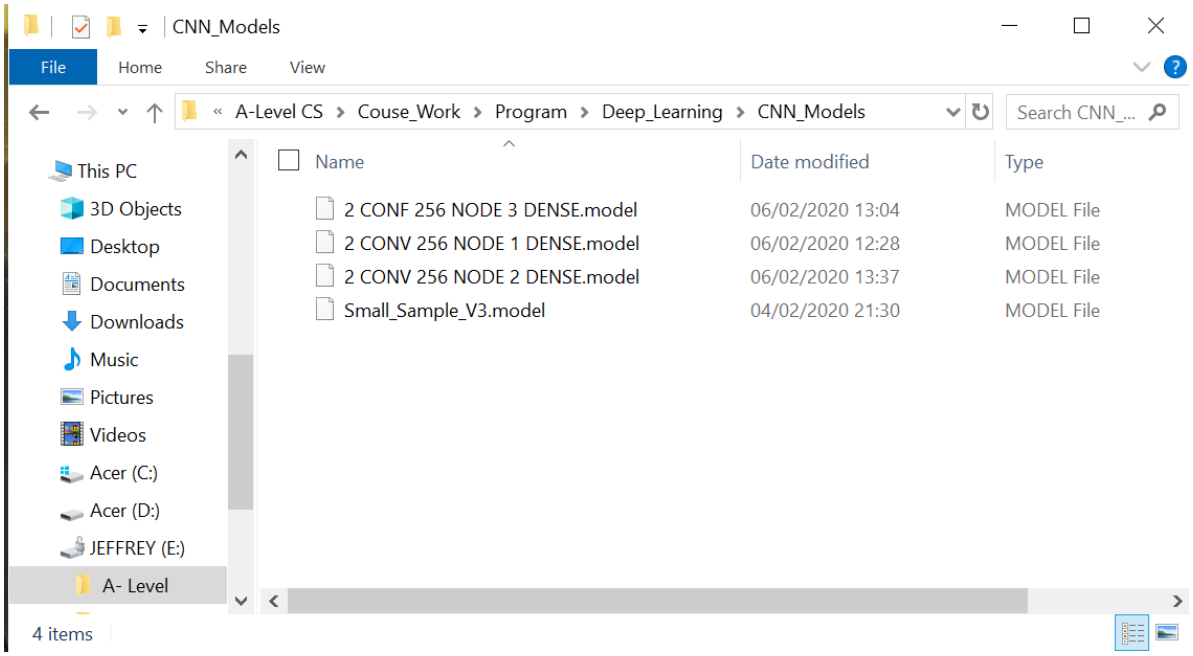


The structure of the Deep_Learning folder is as follow:

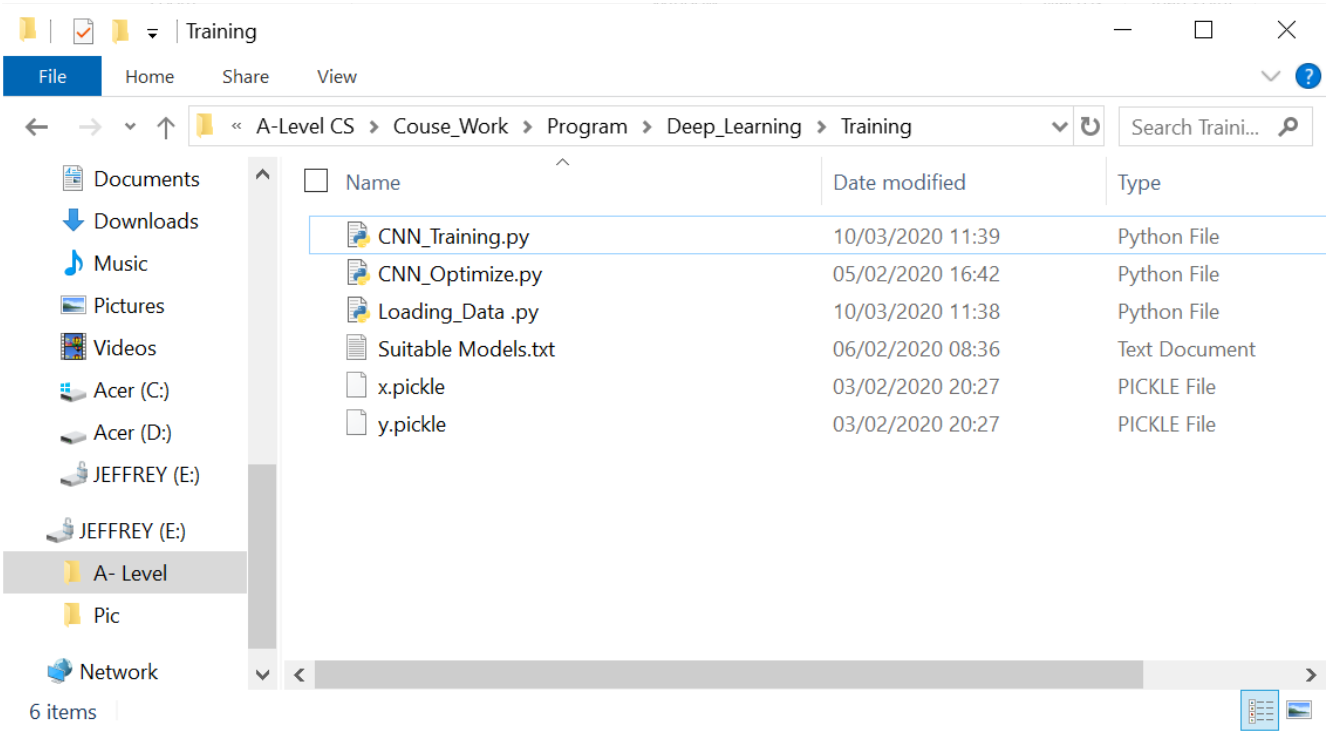


The Making_Prediction_Class file stores the class the will be used to make prediction on imported pre-processed image of letter, the Model Accuracy Test file is an automated program that is used to test the accuracy of models stored within the CNN_Models folder (see below) using the images stored within the Test_Sets folder. The result will be stored within the Results folder

The logs folder stores the log files that is used along with TensorBoard to monitor training process and the CNN_Models folder stores all the trained models as shown below:



The Training folder stores all the files that are used to create and train CNN models:

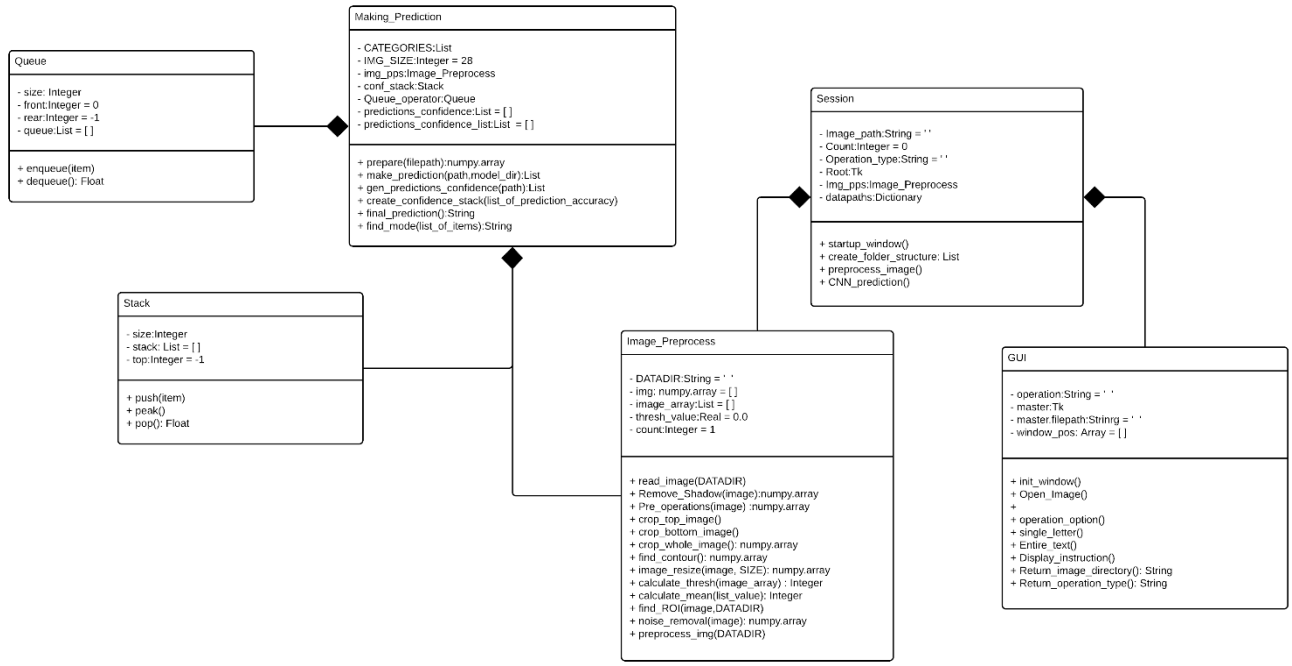


Lastly, the Utilities folder within the main Program folder is used to store my classes of Stack, Queue and my algorithm for merge sort. As for other folders, as specified by their names, are storing the program and files for different sections mentioned in section 2.1.

Please refer to section 3 for the codes that are stored within the python files.

2.3. OOP and Class Definitions

There are 4 main classes in my program: Session, GUI ,Image Pre-processing and Making_Prediction, other minor classes such as Stack and Queue that I have defined by myself wouldn't be talked about.



- Session

Session class establishes communication with all other classes with the use of composition.

Class Session			
Private	**Attributes**	**Datatype**	**Description**
	Count	Integer	
	Image_path	String	Stores directory of the image selected by the user
	Operation_type	String	Determines type of operation that the current session is working on
	Root	Tk	Used to instantiate an object of the built in Tk class in the Tkinter package
	Img_pps	Image_ Preprocess	Used to Instantiate an object of the Image_Preprocess class
	Now	/	Stores the result returned by the now() method of the datetime class (the current date-time)

	datapaths	Dictionary	Stores strings of data that represents directories of folders used for storing images.
			`self.datapaths = {'SESSION':'Picture_Preprocess_Final\\Sessions\\',`
			`'ROI_PATH':'ROI_Images',`
			`'FINAL_PATH':'Final_Images',`
			`'CURSESSION':'Session-{now}\\'.format`
			`(now = self.now.strftime('%Y-%m-%d_%H-%M'))}`

Public	**Methods**	**Description**	

	Startup_window	Instantiate an object of the GUI class, set the dimensions of the main frame. Assigns values of file directory and operation type stored within the GUI object to image_path and operation_type attributes respectively as the mainloop halts, which is when the main frame is destroyed.
	Create_folder_structure	Creates the basic folder structures used to store images. This method is called whenever a new session is started and returns a list that contains directory of ROI and Final folders of the current session. For more information about the structure of the folders and files, please see section 2.2.
	Preprocess_image	Reads the image which has the file directory stored in Image path attribute. If the operation type stores 'Single' then pre-process the image directly, if it's 'Entire', another segmentation method would be called in order to separate out individual letters within the image and then pre-process extracted images individually.
	CNN_Prediction	Use pretrained CNN model to make predictions of images within the final_images folder of current session

- GUI

GUI is the class responsible for creating the main graphical user interface, this class inherits the Frame class which is built into the Tkinter tool kit. GUI class is also composited by the Session class. Please see section 2.4 for more examples and explanations.

Class GUI			
Private	**Attributes**	**Datatype**	**Description**
	operation	String	Stores the type of operation that the user wants the program to do, identified by either 'Single' or 'Entire'
	master	Tk	Stores object of the Tk class which has already been instantiated in the Session class. Essential to utilise the tool kit (such as creating frames)
	master.filepath	String	Stores the file path of the image that the user chose
	Window_pos	array	Stores the predetermined x and y values so that windows placed will be centred in the middle of computer screen
Public	**Methods**	**Description**	
	init_window	Creates the main frame with essential buttons for the user to input image, exit the program and to see the instructions	

	Open_Image	Allows user to select the desired image by opening a selection frame, and saves the file path of the image to the master.filepath attribute
	Operation_option	Creates another frame on top of main frame with buttons that allow user to select desired mode of the program: Identify a single letter or Identify entire text within the image
	Single_letter	Assigns the string value 'Single' to the operation attribute and destroys the main frame
	Entire_text	Assigns the string value 'Entire' to the operation attribute and destroys the main frame
	Display_instruction	Creates another frame on top of main frame with series of text instructions laid out
	Return_image_directory	A getter method for returning the value of master.filepath attribute
	Return_operation_type	A getter method for returning the value of operation attribute

- Image_Preprocess

This class is used to do image operations on desired image, which will be used to process images imported by the user, and images that are used for training before feeding them into the model.

Class Image_Preprocess			
Private	**Attributes**	**Datatype**	**Description**
	DATADIR	String	Stores directory of the selected image
	Img	NumPy Array	Stores the image file imported as a NumPy array
	Image_array	List	Stores the image file that has been converted into a list
	thresh_value	Real	Stores the calculated threshold value of the current session. Used in binary thresholding
	count	Integer	A counter
Public	**Methods**	**Description**	
	Read_image	Reads the image, assigning the DATADIR attribute the directory of the image and assign img attribute the return value of cv2.imread(DATADIR) which would be the image in NumPy array form	
	Remove_Shadow	Removes shadow within the inputted image, returns the resulting image	
	Pre_operations	Apply Gaussian blur to the image, then convert it the grey scale and calculates its threshold value (store in thresh_value attribute). Then apply binary threshold using the calculated	

		threshold value. Convert the resulting image to a list and assign this list to image_array attribute, returns threshold image
	Crop_top_image	Crops any blank space at the top of the image
	Crop_bottom_image	Crops any blank space at the bottom of the image
	Crop_whole_image	Calls the crop_top_image and crop_bottom_image methods, then rotate 90 degrees and crop top and bottom again. Convert resulting list back to NumPy array and return the image array.
	Find_contour	Finds contour lines within the image and draws the contour line onto the image, returns the resulting image at the end
	Image_resize	Resize the image to dimension (SIZE, SIZE) specified in parameter and returns resulting image
	Calculate_thresh	Calculates threshold value of the current image array and returns the value
	Calculate_mean	Calculates mean value of the inputted list and returns the value
	Find_ROI	Select certain boundary boxes to separate out individual letters from the image. Writes the extracted images into the ROI folder of the current session
	Noise_removal	Uses morphological transformations to remove noises within the inputted image and returns resulting image
	Preprocess_img	Function which gathers all image pre-processing methods and apply them onto the inputted image. Writes the resulting processed image image into the Final_Images folder of current session

- Making_Prediction

This class is used make the final prediction on what alphabet class does the image belong in using trained CNN models, and output this prediction onto console

Class Making_Prediction			
Private	**Attributes**	**Datatype**	**Description**
	CATEGORIES	List	Stores all the labels for alphabet classes
	IMG_SIZE	Integer	Stores the set resolution (28 * 28) that all images should be in
	Img_pps	Image_Preprocess	Image pre-processor used to pre-process images before being used to make prediction
	Conf_stack	Stack	Stack used for holding the confidence levels of predictions made in order
	Queue_operator	Queue	Queue used to store all confidence levels of predictions made

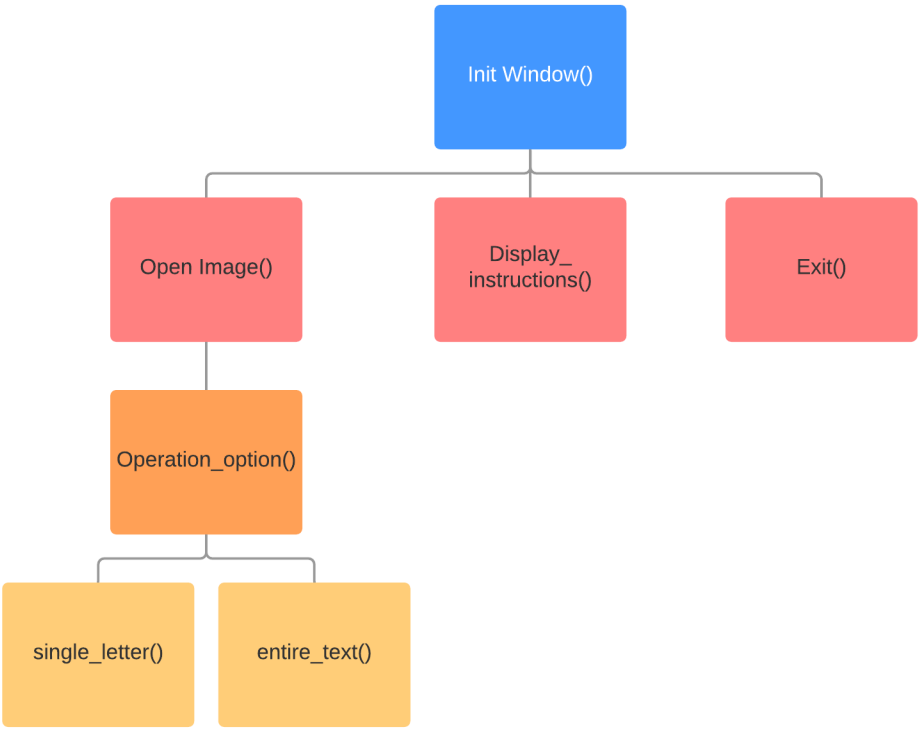
	Predictions_confidence	List	Used store predictions made along with their confidence levels in 3 dimensions
	Predictions_confidence_list	List	Used store predictions made along with their confidence levels in 2 dimensions
Public	**Methods**	**Description**	
	Prepare	Convert all images to 28 * 28 resolution, crop the image and ensure that all input image has white background black foreground and return resulting image	
	Make_prediction	Make prediction on imported image using CNN model specified by parameter and return list of [prediction, confidence level]	
	Gen_predictions_confidence	Generate multiple predictions using different CNN models by calling Make_prediction method and return the return value of that method	
	Create_confidene_stack	Put the confidence levels into conf_stack attribute with the top being highest confidence level	
	Final_prediction	Make final predictions based on various conditions, details specified in section 2.6.4	
	Find_mode	Find the mode of inputted list and return the mode	

2.4. Graphical User Interface

- 2.4.1 Section Outline

In order to create a user interface in python, I will be using Tkinter which is an API that binds python to the Tkinter toolkit. It allows the creation of graphical frames and buttons which could come in extremely handy.

In addition to the GUI, I will be programming the interface in Object-Oriented Programming paradigm to ensure its reusability and to allow easier 'communication' with classes that I would create for other sections. There will be differences in appearance between the current GUI prototype and the final GUI, however the structure will be the same.



- 2.4.2 Imported Libraries

```
from tkinter import *
from tkinter.filedialog import askopenfilename
from tkinter.messagebox import showerror
```

Tkinter is the only external library that will be used to complete the GUI, I've imported **askopenfilename** module in order for the file opening frame to be displayed, and **showerror** module to display error messages when user inputted wrong file type. The * wildcard is used to import other essential modules and widgets within the library.

- 2.4.3 Constructor

```
class Window(Frame):
    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.master = master
        self.operation = ""
        self.master.filepath = ""
        self.init_window()
```

The class Window inherits the Frame class, which is a built-in class within the Tkinter library
Calls constructor of the Frame parent class to initialise all inherited attributes. Master is a widget within the parent class and is None by default. However, to utilise the frames, a root widget Tk() will be passed as master.
operation attribute is a string variable that stores the operation choice that the user entered in Operation Option frame
self.master.filepath attribute is a string variable that stores the file directory of the imported image

	Init_window() method creates the main menu frame, it is called within the constructor so that whenever an object of Window class is instantiated, the main menu frame would be displayed automatically.

- 2.4.4 Init_Window()

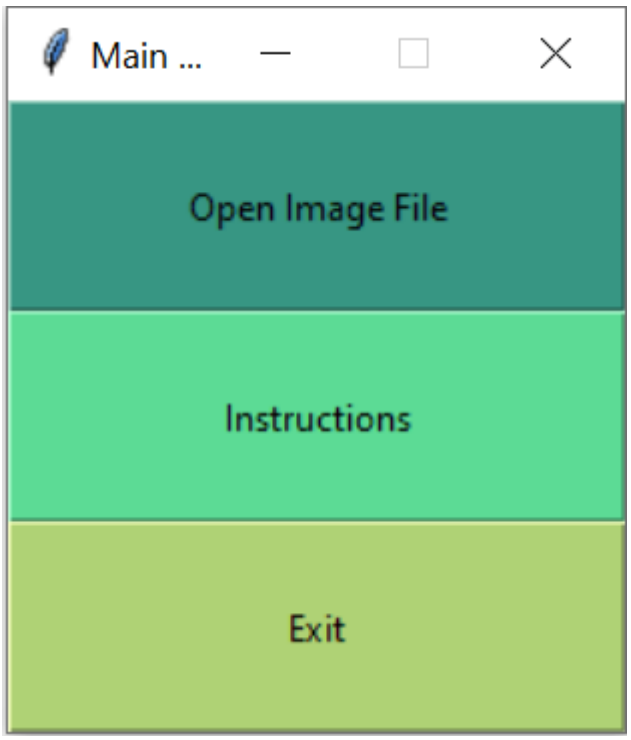
```
def init_window(self):
    self.master.title("Main Menu") #Name of the frame
    self.pack(fill=BOTH, expand=1) #Widget fill empty space
    input_button = Button(self,text = "Open Image File",width = 28,
                          height = 4, bg = "#379683", fg ="Black",
                          command = self.Open_Image) #Open_Image button
    instruction_button =Button(self,text = "Instructions",width = 28,
                               height = 4,bg = "#5CDB95",fg ="Black",
                               command = self.display_instruction)
                               #Instructions Button
    exit_button = Button(self,text = "Exit", width = 28, height = 4,
                         bg = "#AFD275",fg ="Black",
                         command = self.master.destroy)#Exit button
    input_button.place(x=0,y=0)
    instruction_button.place(x=0,y=70)
    exit_button.place(x=0,y=140)
    #Placing of buttons with specified coordinates and place them on frame
    #Set coordinates of buttons and place them on the frame
```

Geometry organizer/packer which allows widgets to be organized.

Creates buttons and specifies its definition: text displayed on the button, height/width, fore/background colour and the method that will be called onclick

This command destroys Root window hence destroying all present frames

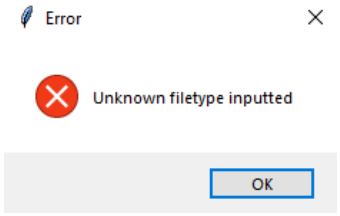
Placing of the buttons that were defined above with their specific x and y coordinates.
Currently I want all buttons to be aligned and are place on the very left of the interface



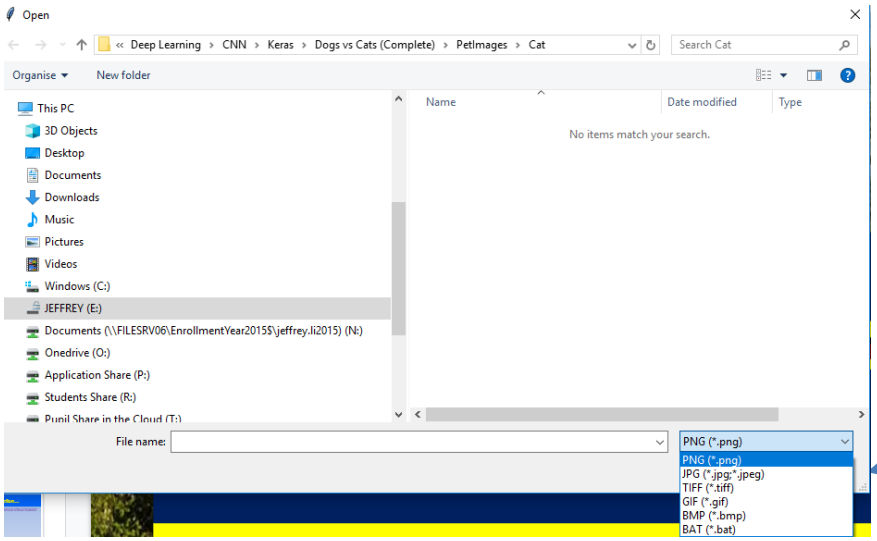
- 2.4.5 Open_Image ()

```
def Open_Image(self):
    extensionsToCheck =
[".png",".jpg",".jpeg",".tiff",".gif",".bmp",".bat"]
    file = askopenfilename(filetypes=(("PNG", "*.png"),
                        ("JPG", "*.jpg;*.jpeg"),
                        ("TIFF", "*.tiff"),
                        ("GIF", "*.gif"),
                        ("BMP", "*.bmp"),
                        ("BAT", "*.bat")))
    if not any(ext in file for ext in extensionsToCheck):
      showerror(title = "Error",message = "Unknown filetype inputted")
      label = Label(self,text = ("Please input an image file"))
      label.config(font=("Courier", 10))
      label.place(x=250,y=100)
    else:
      self.master.filepath = file
      self.operation_option()
```

List that stores string of all file extensions that will be accepted by the validation statement
askopenfilename opens the File Opening window, filetypes specifies the file extension that the user can choose from
The file directory is returned and assigned to this file variable
This selection statement checks whether the file directory string contains any of the string stored within the extensionsToCheck list.
If not then the Error message frame is shown, otherwise the file directory string is assigned to the self.master.filepath attribute and calls the operation_option method



Error message frame



File opening window

Allow user to select desired file extension

- 2.4.6 Operation_option()

```
def operation_option(self):
    window = Toplevel()
    self.master.withdraw()
    label = Label(window,text = ("Please select an option"),width = 40)
    label.config(font=("Courier", 9))
    label.pack()
    single_letter_button = Button(window,text = "Single Letter Prediction",
                          width = 30, bg = "#379683", fg ="Black",
                          command = self.single_letter).pack()
    entire_text_button = Button(window,text = "Entire Text Prediction",
                         width = 30, bg = "#5CDB95",fg ="Black",
                         command = self.entire_text).pack()
def single_letter(self):
    self.operation = ("Single")
    self.master.destroy()
def entire_text(self):
    self.operation =("Entire")
    self.master.destroy()
```

Top level widget is created here, whose parent is the root widget (self.master)

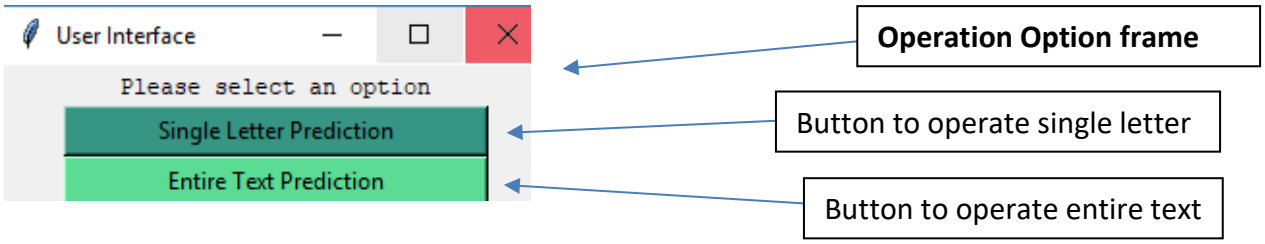
Hides the main menu frame, not closing it though

Specifies placing of text 'Please select an option', its font type and size

Placing of buttons, and other properties. Calls the method specified in command, single_letter() and entire_text().

Assign either 'Single' or 'Entire' to the operation attribute in order for the main program to identify the type of operation chosen. Then destroys

	master, stopping the mainloop



- 2.4.7 Display_instructions()

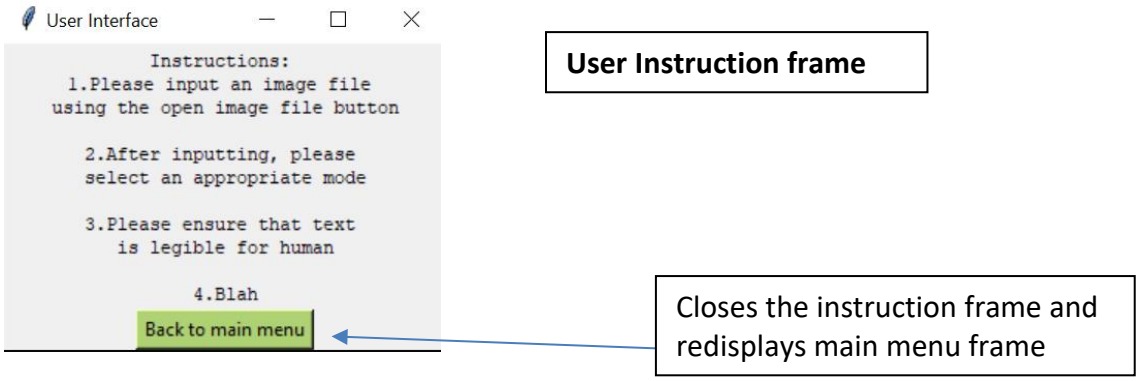
```
def display_instruction(self):
    window = Toplevel() #Create another window
    self.master.withdraw() #Hide mainFrame window
    label = Label(window,text = ("Instructions: \n1.Please input an
image file \nusing the open image file button\n\n2.After inputting,
please \nselect an appropriate mode\n\n3.Please ensure that text \nis
legible for human\n\n4.Blah" ),width = 40)
    label.config(font=("Courier", 9))
    label.pack()
    return_menu_button = Button(window,text = "Back to main menu",
                    width = 15, bg = "#AFD275", fg ="Black",
                    command =
lambda:[window.destroy(),self.master.deiconify()]).pack()
```

Creates another frame for displaying instructions on top of the main frame and hides the main frame

Sample instructions that will be displayed as shown in the images above

Creates a button that allows the user to return back to the main frame (essential 'unhide' the main frame and destroy the current frame).

The commands are specified by the lambda expression to allow the use of anonymous functions



- 2.4.8 Getter Method

```
def return_image_directory(self): #Return extracted image directory
    return self.master.filepath
def return_operation_type(self):
    return self.operation
```

The is a getter method used to return the file directory of the imported image and operation type. This is used so that the Session class within the main program (which will be mentioned later on) which composites Window class, can utilise the values of these attributes while achieving information hiding and encapsulation of the Window class.

- 2.4.9 Exit() Subroutine

There isn't a specific method implemented for exiting the main loop, Exit() is achieved by calling the self.master.destroy parent method to destroy the root hence quitting main loop and closing all existing frames, which is frequently used in the code shown above.

2.5. Image Pre-Processing

- 2.5.1 Section Outline

As mentioned in earlier Analysis sections, to achieve image pre-processing, the OpenCV library will be used. Output image at the end of the processing would be saved in the Final_Image folder of the current session folder and then will be fed into the CNN model for it to make a prediction.

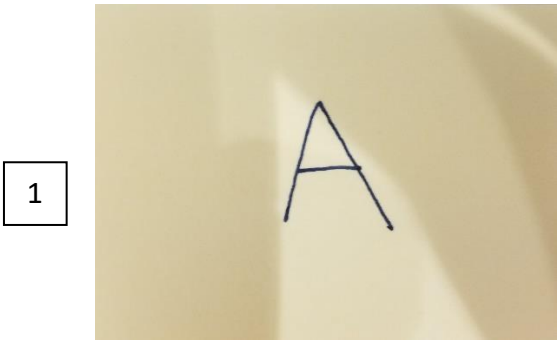
ROI folder is used to store images of letters separated from the text image so that they can each be processed in turn and saved in the Final_Image folder.

Please note that example codes within this section are taken out from the original class of Image Pre-Processing hence some variables may not have been explicitly declared within that section of code.

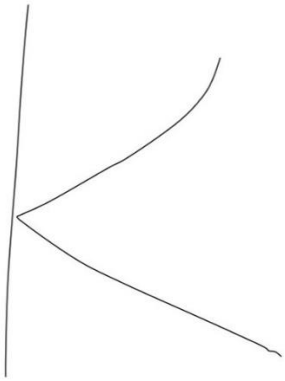
- 2.5.2 Further Analysis on images

This section would act as an extension to the Analysis section before, to explore the problem of Image pre-processing more thoroughly.

Below are some of the images with distinct features that I used for analysis and testing on my Image Pre-processing class, that are taken from myself, peers and the internet. They are each labelled with a number:



1



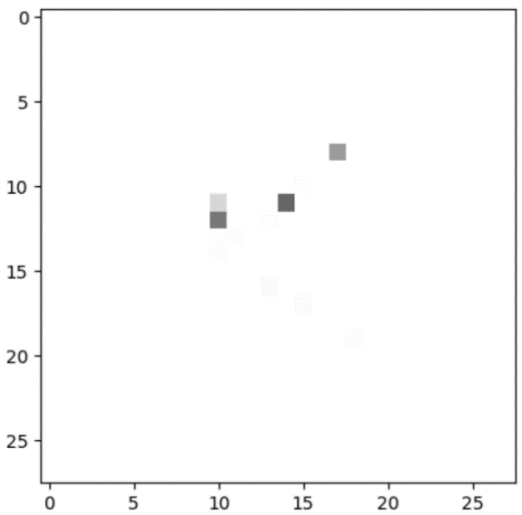
2

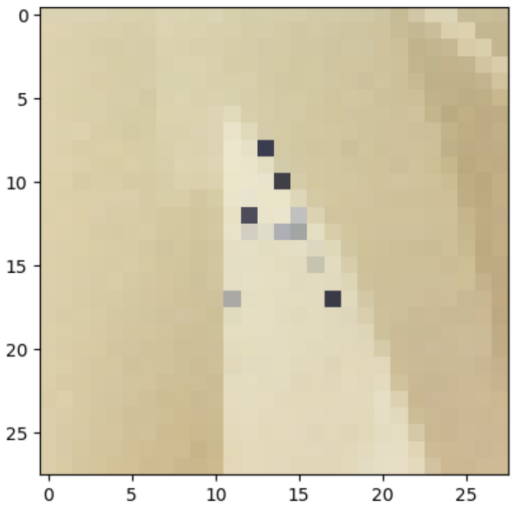
3

The images include hand writing written using different coloured pen, different background colour, different thickness of handwriting and different size.

With the image operations that have already been mentioned in section 1.3.3 and 1.3.4 in mind, I have encountered various other problems while developing the program for this section.

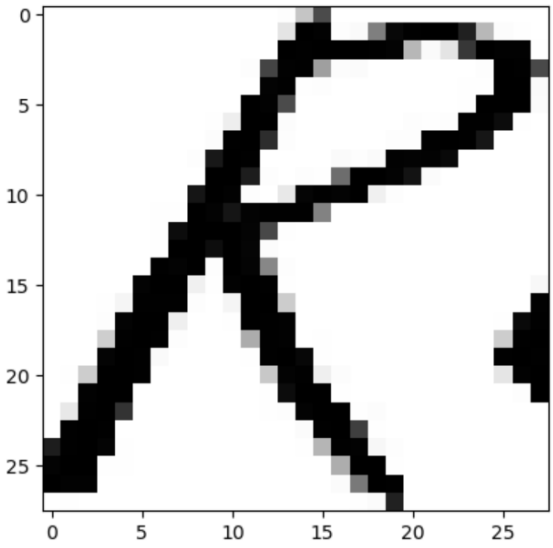
Firstly, as all images need to be normalised to 28 pixels by 28 pixels so that they could be processed by the CNN as explained before, this introduces a problem which is shown by the images below:





These two images are the results of resize image 1 and image 2 to 28 pixel by 28 pixels. As you can see, there are hardly any pixels left within the images as a result of squashing a 2651 by 1926 pixels Image into such a small dimension.

Although the effect of these problems can be slightly reduced by cropping out unnecessary empty spaces within the image, the results still aren't satisfactory for myself. Luckily, as I was testing the resizing method on various images, I found one which isn't affected that much through resizing:



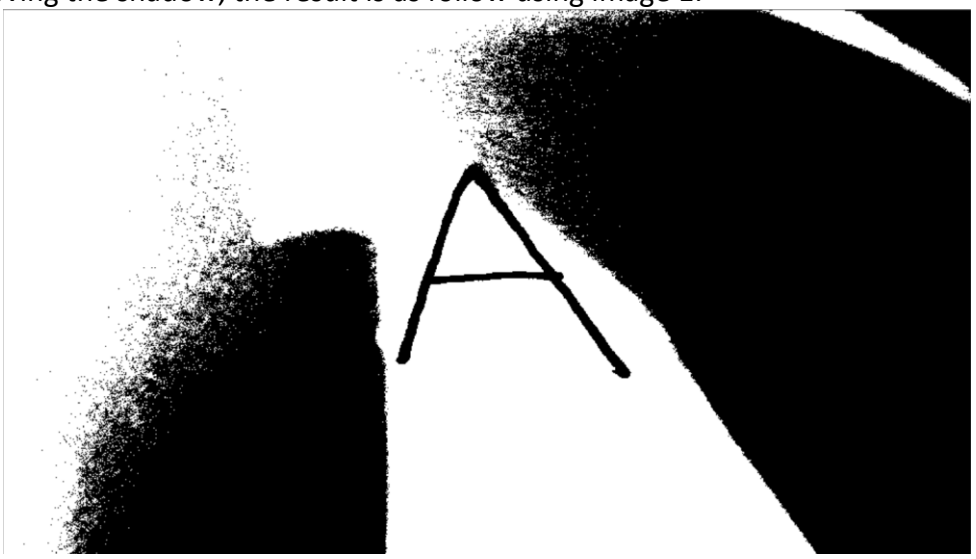
This image, which originally is image 3, has a very thick handwriting along and is very well cropped with very little empty space.

This I believed was the solution to the problem, and eventually after many experiments on resizing well cropped images with thick hand writing, turns out to be true. The methods of thickening letters and cropping will be discussed in sections 2.5.5 and 2.5.6.

The second problem is the Watershed algorithm, which I have mentioned in detail at the end of section 1.3.4. The result after applying the watershed algorithm along with distance transform operator is shown below with the green lines representing the boundaries.

Although the algorithm has been successful in segmenting the letters, however, I have been having difficulties in extracting the content within the segmented regions. Hence ROI is used (See section 2.5.7).

The third problem is the problem with shadows within the image, which I struggled the most with. When images with shadow is converted to grayscale and straight through binary thresholding without removing the shadow, the result is as follow using image 1:



This problem is emphasized when the colour of the handwriting is very faint, hence has a similar pixel value with the shadow in grayscale, which makes it very difficult to calculate a threshold value for the image in order to remove shadow completely, without affecting the content and make it impossible for CNN to predict it.

The solution to this problem is by using an algorithm to remove the shadow from the image before doing other operations on it. The algorithm is in section 2.5.8.

- ## 2.5.3 Pre-operations

Pre-operations involves simple operations such as applying Gaussian Blur (Blurs the image to achieve simple noise removal), converting image into grayscale, apply binary threshold and convert the image into a 2D array to make it easier to operate on the image:

```
def Pre_operations(self,image):
    shadow_removed = self.Remove_Shadow(image)
    blurr_image = cv2.GaussianBlur(shadow_removed,(5,5),cv2.BORDER_DEFAULT)
    #Objective 2.a
    grey_image = cv2.cvtColor(blurr_image,cv2.COLOR_RGB2GRAY)
    self.image_array = grey_image.tolist()
    self.thresh_value = self.calculate_thresh(self.image_array)
    #Objective 2.c
    ret, thresholded = cv2.threshold(grey_image,self.thresh_value,255,cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    self.image_array = thresholded.tolist()
    return thresholded
```

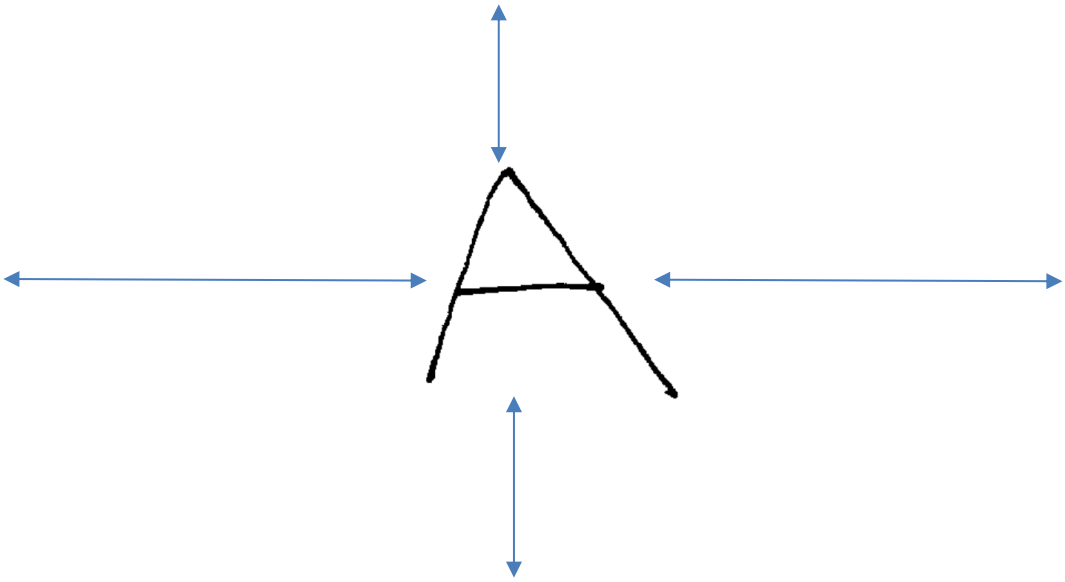
- ## 2.5.4 Calculating Thresh

Calculate_thresh method works along with calculate_mean, where calculate_mean takes in a list as parameter and calculates the mean of the inputted list. Since the image is converted into a 2D list/array, calculate mean would be utilised to calculate the overall mean of the 2D list which would be the threshold value.

```
def calculate_thresh(self,image_array):
    mean_values = []
    for x in range(len(image_array)):
        mean_values.append(self.calculate_mean(image_array[x]))
    return self.calculate_mean(mean_values)

def calculate_mean(self,list_value):
    total = 0
    for element in (list_value):
        total += element
    mean = round(total / len(list_value))
    return mean
```

- ## 2.5.5 Cropping of the image

Below is the processed binary image of image 1 without resizing, cropping and contouring and blank space are marked with the arrows. The main principle of how a binary image can be cropped is by removing the white spaces around the desired section, which utilises the min () built-in function.



```
def crop_top_image(self):
    empty_space_count = 0
    length = len(self.image_array)
    for x in range(length):#FOR EACH ARRAY WITHIN THE 2D ARRAY
      if min(self.image_array[x]) == 255: #IF ALL ELEMENTS = 255, ADD 1 TO EMPTY SPACE COUNT
        empty_space_count += 1
      else:
        break #IF 1 ELEMENT WAS FOUND TO NOT BE 255 THEN MOVE ONTO THE NEXT ARRAY
    for y in range(empty_space_count - 10): #RESERVE TO PIXELS TO NOT BE DELETED
      self.image_array.remove(self.image_array[0]) #FOR THE NUMBER OF COUNTS, REMOVE THE WHITE SPACES

def crop_bottom_image(self): #SIMILAR PRINCIPLE AS CROP_TOP_IMAGE() FUNCTION
    empty_space_count = 0
    for x in range ((len(self.image_array)-1),-1,-1): #ITERATE IN REVERSE
      if min(self.image_array[x]) == 255:
        empty_space_count += 1
      else:
        break

    for y in range(empty_space_count - 10):
      del self.image_array[-1]

def crop_whole_image(self):
    self.crop_top_image()
    self.crop_bottom_image()#CROP TOP AND BOTTOM
    rotate_image = np.rot90(self.image_array,3) #ROTATE BY 270 DEGREES, EASIER TO CROP LEFT/RIGHT
    self.image_array = rotate_image.tolist() #CONVERT NUMPY ARRAY TO LIST
    self.crop_top_image()
    self.crop_bottom_image() #CROP LEFT AND RIGHT
```

```
fully_cropped_image = np.rot90(self.image_array)#ROTATE IMAGE BACK TO ORIGINAL
return np.asarray(fully_cropped_image, dtype = np.float32)
#CONVERT LIST BACK TO NUMPY ARRAY
```

The cropping function would iterate through the arrays within 2D array (the image) and calculate the minimum value within that array. If the minimum value is 255, which represents white pixels, add 1 to a counter n. Otherwise the loop is exited. Afterwards, delete n number of arrays within the 2D array to get rid of the white spaces.

What this is doing is checking whether the array / row of pixels contains at least 1 darker pixel, which would be features that need to stay, and take out any array/ row of pixels that only contains white pixels.

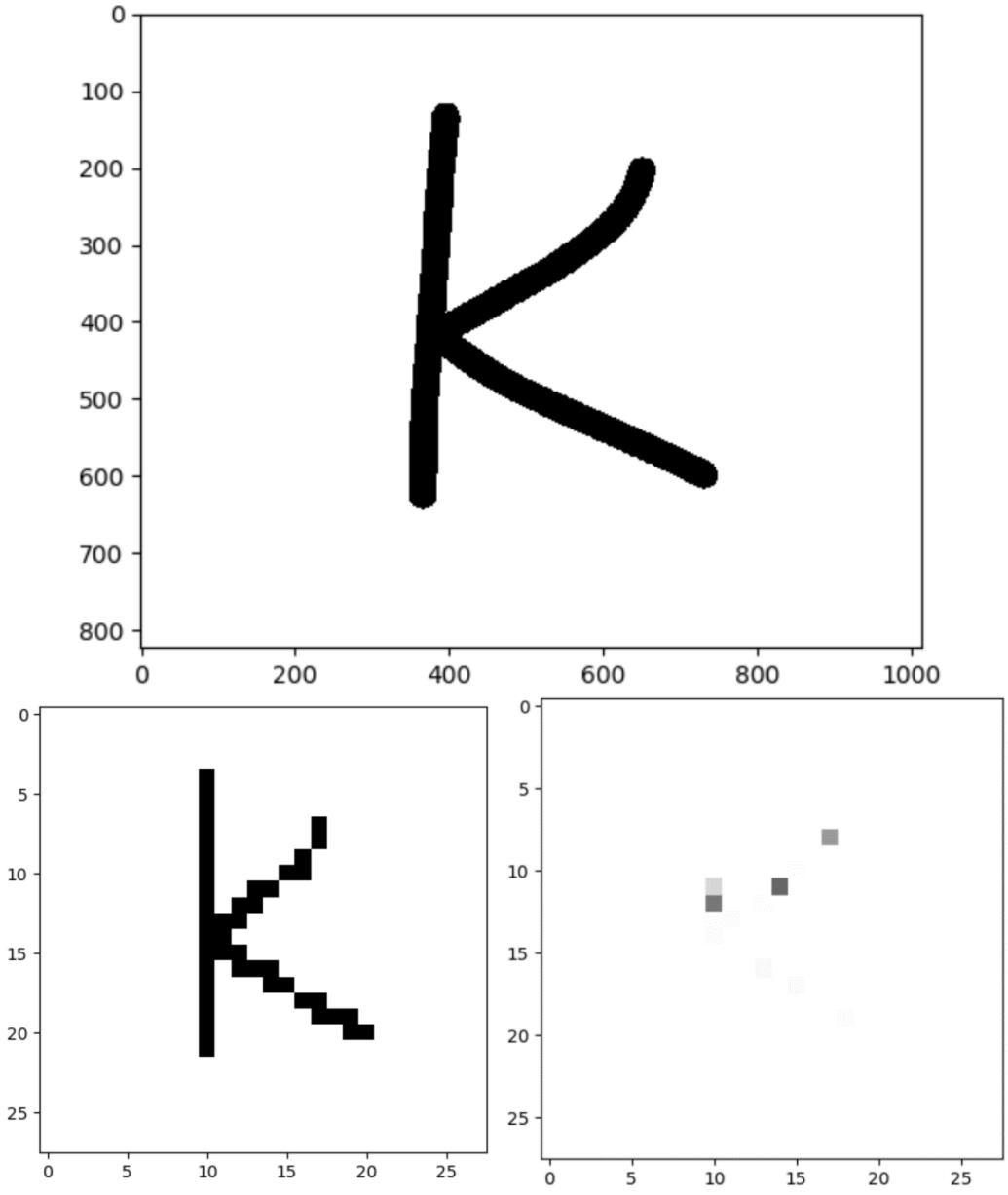
The same principle is applied for removing white spaces at the bottom where the iteration and deleting process is done in reverse, as well as cropping left and right where the image is rotated by 270 degrees then apply crop top and crop bottom function and rotate by 90 degrees at the end.

- 2.5.6 Find contour

The sample image 2 will be used. Before finding and drawing the contour lines, the image should already be in binary form (black and white pixels only) for better accuracy of finding contour lines.

Contour lines within the image are found and thicker version of contour line which can be adjusted by changing the parameter, will be drawn onto the original image to thicken the original letter, which eliminates the problem described in section 2.5.2

Numbers on the axis within the image represent the number of pixels, with the first image showing the thicker contour line drawn onto the original image, and the second array of images is showing the comparison of the results after resizing the thickened letter (left) and not thickened letter (right) to 28 pixels by 28 pixels.




```
def find_contour(self):
    cropped_img = cv2.imread('tmp.png')
    blurr_image = cv2.GaussianBlur(cropped_img,(5,5),cv2.BORDER_DEFAULT)
    grey_image = cv2.cvtColor(blurr_image,cv2.COLOR_RGB2GRAY)
    ret,thresh = cv2.threshold(grey_image,self.thresh_value,255,cv2.THRESH_BINARY)
    contours, hierachy = cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_NONE)
    for x in range(1,len(contours)):
        cnt = contours[x]
        cv2.drawContours(thresh, [cnt], 0, (0,0,0),5)
    return thresh
```

The image that has already been processed into binary form is passed into findContours function. The findContours function in the current case takes image array, contour retrieval mode and contour approximation method.

The function returns contours found which are stored in a one-dimensional list, with each element being a NumPy array of coordinates of boundary points of the object, and a hierarchy variable as a NumPy array as well.

The contour retrieval mode that I used is RETR_TREE which retrieves all of the contours within the image and construct a hierarchy of nested contours

The contour approximation method used is CHAIN_APPROX_NONE, where all boundary points within the image are stored, this is used rather than the other method CHAIN_APPROX_SIMPLE to obtain the most accurate boundary lines as CHAIN_APPROX_SIMPLE removes all redundant points and compresses the contour.

At last, the contours are drawn using the drawContours function with parameters in this case (image, contours, contourIdx, color, thickness). It draws the object using boundary points stored within the contours list, using colour black represented by (0,0,0) and with specific thickness, which is 5 in this case.

The contour of the image's boundary isn't drawn which is specified by the contourIdx parameter which is currently 0, if a negative value is used then all contours will be drawn.

- ## 2.5.7 ROI

ROI stands for rectangular region of interest; it is used to select boundary boxes which tends to be in rectangular shapes.

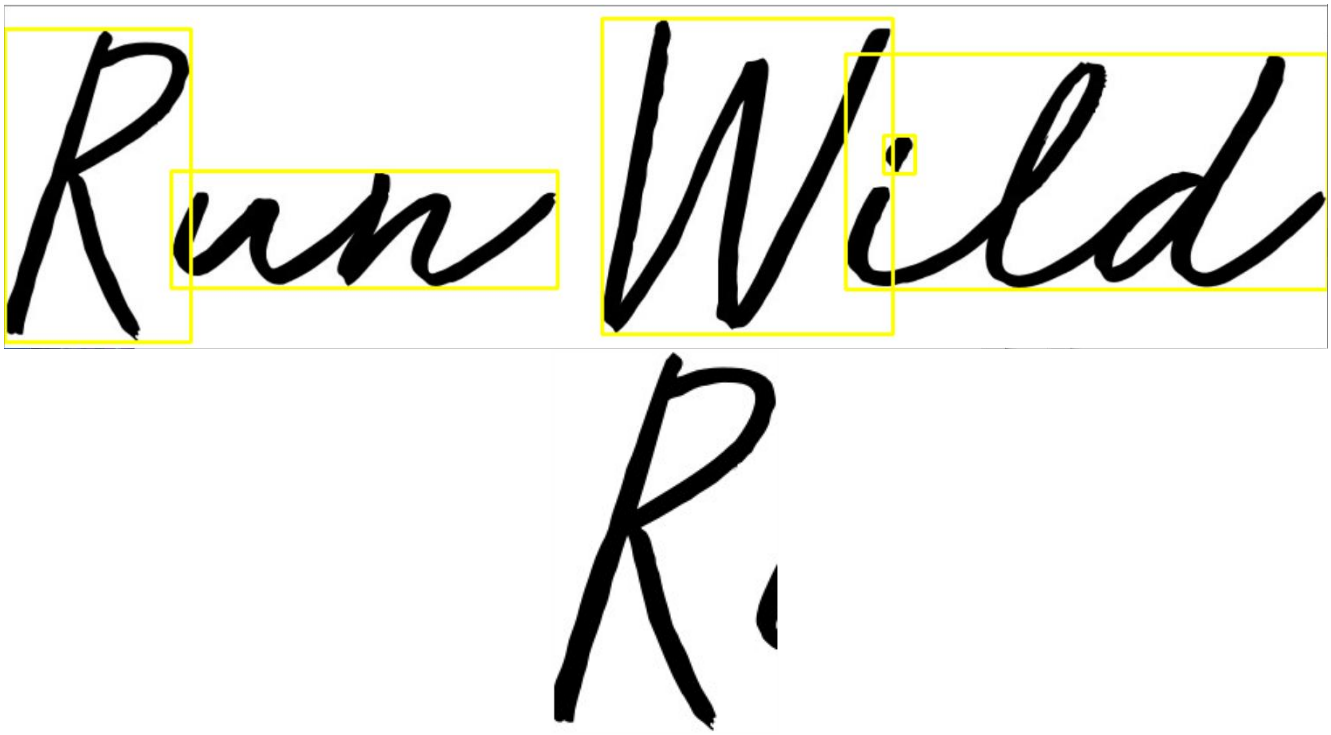
This method is a replacement to the watershed segmentation method mentioned in the Analysis section, as practically, it is very difficult to utilise the watershed algorithm to separate out the letters, whereas ROI is a more efficient method.

As shown in the first image below, I am trying to separate the hand written text into individual letters so that they can each be processed and predicted separately.

ROI selects and draws boundary boxes onto the image which are the yellow boxes as shown below, and the contents within the boundary boxes can each be written as separate image files and stored within the ROI folder of current session.

The second image is an example image of the letter R separated from the handwritten text image. I have taken reference of the algorithm from this website[18] while trying to code the method. Various adaptions have been made to make the code work with my program.

[18] https://cvisiondemy.com/extract-roi-from-image-with-python-and-opencv/



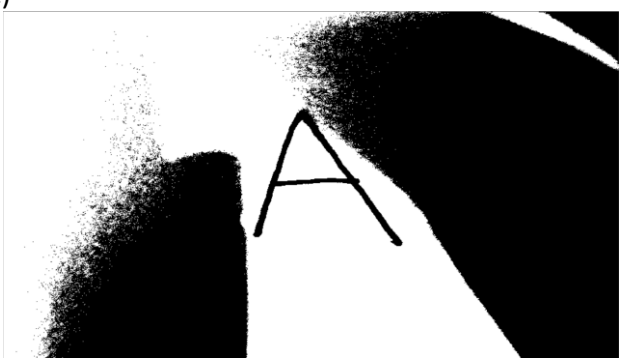
```
def find_ROI(self,image,DATADIR):
    #https://cvisiondemy.com/extract-roi-from-image-with-python-and-opencv/
    image = image.copy()
    '''image = np.reshape(image,(np.shape(image)[0],np.shape(image)[1],1))'''
    ret, thresh = cv2.threshold(image, self.thresh_value, 255, cv2.THRESH_BINARY_INV)
    kernel = np.ones((5,5),np.uint8)
    img_dilation = cv2.dilate(thresh, kernel, iterations=1)
    contours, hierachy = cv2.findContours(img_dilation.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    # sort contours
    sorted_contours = sorted(contours, key=lambda ctr: cv2.boundingRect(ctr)[0])

    for i, ctr in enumerate(sorted_contours):
        # Get bounding box
        x, y, w, h = cv2.boundingRect(ctr)

        # Getting ROI
        roi = image[y:y + h, x:x + w]
        cv2.rectangle(image, (x, y), (x + w, y + h), (255,255,255), 2)
        if w > 40 and h > 40:
            save_dir = os.path.join(DATADIR, 'ROI' + str(i) + "." + 'png')
            cv2.imwrite(save_dir, roi)
```

- 2.5.8 Remove Shadow

Remove shadow is used to reduce the negative effects of having shadows while applying binary thresholding as explained in section 2.5.2.

While looking for ways in which shadow can be removed from the image, I found a descent algorithm[19] of which I have adapted to fit the purpose of my program better. This method returns the image array with shadow removed as shown below.

As you can see, the image with shadow removed becomes extremely faint and has many other colours, hence, the processed image can then have binary threshold applied to it to make the image binary coloured. This has a better result than the image without shadow removed (1st image with shadow, 2nd image without)



[19] https://stackoverflow.com/questions/44752240/how-to-remove-shadow-from-scanned-images-using-opencv

```
def Remove_Shadow(self, image):
    #https://stackoverflow.com/questions/44752240/how-to-remove-shadow-from-scanned-images-using-opencv
    rgb_planes = cv2.split(image)
    result = []
    for plane in rgb_planes:
      dilated_img = cv2.dilate(plane, np.ones((7,7), np.uint8))
      bg_img = cv2.medianBlur(dilated_img, 21)
      diff_img = 255 - cv2.absdiff(plane, bg_img)
      norm_img = cv2.normalize(diff_img,None, alpha=0, beta=255, norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_8UC1)
      result.append(norm_img)
    return cv2.merge(result)
```

- 2.5.9 Image Pre-processing recursion

The first preprocess_img method is a method located within the image pre-processing class, this is used as an interface to utilise all operations described in previous sub-sections.

The second preprocess_img method would be a subroutine that will be located within the Session class, used to process the image differently according to the operation type selected by the user. After processing the image, resulting image/images will be saved into the folder of the current session.

Recursion is used with the second preprocess_image method, while processing images of entire text as the image would be split into several ROI images of individual letters and saved within the ROI folder of current session. Each of them will then be read and processed by using recursion.

```
def preprocess_img(self,DATADIR):
    image = self.img.copy()
    cropped = self.crop_whole_image()
    cv2.imwrite('tmp.png',cropped)
    found_contour = self.find_contour()
    final_image = self.noise_removal(self.image_resize(found_contour,28))
    os.remove('tmp.png')
    cv2.imwrite(os.path.join(DATADIR,'Final_image{number}.png'.format(number = self.count)),final_image)
    #cv2.imwrite(os.path.join(DATADIR,'Original_image.png'),self.img)
    self.count += 1
```

```
def preprocess_image(self,image_path,folder_structure):
    roi_final = folder_structure #index 0 to ROI folder, index 1 to Final folder
    self.img_pps.read_image(image_path)
    operated_image = self.img_pps.Pre_operations(self.img_pps.img)
    if self.operation_type == 'Single':
        self.img_pps.preprocess_img(roi_final[1])
    elif self.operation_type == 'Entire':
        self.img_pps.find_ROI(operated_image,roi_final[0])
        self.operation_type = 'Single'
        for element in os.listdir(roi_final[0]):
            path_to_element = roi_final[0] + ('\\' + element)
            self.preprocess_image(path_to_element,roi_final) #Recursion
    else:
        print('An error has occured, restarting program')
        self.operation_type = 'Error'
```

2.6. Convolution Neural Network[20]

- 2.6.1 Keras and TensorFlow

As I have mentioned in section 1.3.5, although I won't be using TensorFlow for my investigation project due to it being highly inefficient in exporting models, but since this is an investigation project, I need to understand what is going on and what is hidden by the highly efficient Keras API that sits on top of TensorFlow.

I have learned how to use TensorFlow with the aid of YouTube videos made by sentdex[21], and for this section, I would be using TensorFlow code learned from sentdex alongside with some Keras code to explain what is going on. I would start off explaining how a simple neural network model is created in TensorFlow and then move onto the more advance CNN model.

2.6.1.1 About TensorFlow

As its name suggests, TensorFlow is essentially about tensor 'flowing' in a stream. In order to utilise TensorFlow, a Session needs to be created which is an object that encapsulates the environment which the tensors are evaluated and operations are executed[22]. It allows the creation of dataflow diagram which describes how tensor is flowing through series of nodes.

Before anything could be done within the Session, all variables need to be initialised of which in TensorFlow can be a placeholder or a variable, to prepare them to have data being fed into them.

Practical examples are shown in later sections.

[20] All terms used in this section have already been explained in section 1.3.1 and 1.3.2
[21] https://www.youtube.com/watch?v=mynJtLhhcXk&list=PLQVvvaa0QuDfKTOs3Keq_kaG2P55YRn5v&index=57
[22] https://www.tensorflow.org/api_docs/python/tf/compat/v1/Session

2.6.1.2 Simple Neural Network[23]

Below is a simple neural network model in TensorFlow that I have learned from pythonprogramming.net which is a website made by sentdex[24], which is equivalent with the Keras code that I have made and does the job of recognizing handwritten digits from the MNIST dataset:

```
#TensorFlow

import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data

mnist = input_data.read_data_sets("/tmp/data/",one_hot = True)

n_nodes_hl1 = 500
n_nodes_hl2 = 500
n_nodes_hl3 = 500

n_classes = 10
batch_size = 100

x = tf.compat.v1.placeholder("float")
y = tf.compat.v1.placeholder("float")

def neural_network_model(data):
   hidden_1_layer = {"weights":tf.Variable(tf.random.normal([784,n_nodes_hl1])),
                     "biases":tf.Variable(tf.random.normal([n_nodes_hl1]))}

   hidden_2_layer = {"weights":tf.Variable(tf.random.normal([n_nodes_hl1,n_nodes_hl2])),
                     "biases":tf.Variable(tf.random.normal([n_nodes_hl2]))}

   hidden_3_layer = {"weights":tf.Variable(tf.random.normal([n_nodes_hl2,n_nodes_hl3])),
                     "biases":tf.Variable(tf.random.normal([n_nodes_hl3]))}

   output_layer = {"weights":tf.Variable(tf.random.normal([n_nodes_hl3,n_classes])),
                     "biases":tf.Variable(tf.random.normal([n_classes]))}

   l1 = tf.add(tf.matmul(data, hidden_1_layer["weights"]), hidden_1_layer["biases"])
   l1 = tf.nn.relu(l1)

   l2 = tf.add(tf.matmul(l1, hidden_2_layer["weights"]) , hidden_2_layer["biases"])
   l2 = tf.nn.relu(l2)
```

[23] Please see section 1.3.1 for extra clarity

[24] https://pythonprogramming.net/tensorflow-deep-neural-network-machine-learning-tutorial/?completed=/tensorflow-introduction-machine-learning-tutorial/

```
    l3 = tf.add(tf.matmul(l2, hidden_3_layer["weights"]) , hidden_3_layer["biases"])
    l3 = tf.nn.relu(l3)

    output = tf.matmul(l3, output_layer["weights"]) + output_layer["biases"]
    return output

def train_neural_network(x):
    prediction = neural_network_model(x)
    cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=prediction, labels=y))
    optimizer = tf.compat.v1.train.AdamOptimizer().minimize(cost)
    hm_epochs = 10
    with tf.compat.v1.Session() as sess:
        sess.run(tf.compat.v1.global_variables_initializer())
        for epoch in range(hm_epochs):
            epoch_loss = 0
            for _ in range(int(mnist.train.num_examples/batch_size)):
                epoch_x, epoch_y = mnist.train.next_batch(batch_size)
                _, c = sess.run([optimizer, cost], feed_dict = {x: epoch_x, y:epoch_y})

train_neural_network(x)
```

```
#Keras

from keras.models import Sequential
from keras.layers import Dense,Activation, Flatten
from keras.datasets import mnist

 (x_train, y_train),(x_test, y_test) = mnist.load_data()

model = Sequential()
model.add(Flatten())
model.add(Dense(500))
model.add(Activation('relu'))
model.add(Dense(500))
model.add(Activation('relu'))
model.add(Dense(500))
model.add(Activation('relu'))
model.add(Dense(10,activation = 'softmax'))

model.compile(loss='categorical_crossentropy',
            optimizer='adam',  # Good default optimizer to start with
            metrics=['accuracy'])  # what to track

model.fit(x_train, y_train, batch_size = 100 ,epochs=10)  # train the model
```

Investigation on Image Processing Methods and how machines can learn to recognize hand-written English text

MNIST is a dataset that is included in TensorFlow(tf) and Keras with 28 * 28 resolution images of handwritten digits, their labels are included as well. A clear difference between the 2 codes is that Keras is obviously more compact and shorter, and tf code is more difficult for programmers to understand what the code is actually doing.

The first few lines for both codes are just import the MNIST dataset, where x represents the actual images and y represents their labels. In tf code, x and y are assigned as placeholders which are variables that will be assigned data later on, in the current case its values are assigned within the train_neural_network() function during the epochs.

Both neural networks utilise 4 simples Fully connected (dense) layers, and uses Adam optimizer. I was unable to find a softmax loss function which has been used in the tf code, hence I have replaced it with categorical crossentropy loss function which essentials does the same job. Both models will go through 10 epochs with batch size of 100 as well.

Within the neural_network_model() function, the layer of the model are declared as a dictionary and each layers' weights and biases would be stored as tensors identified by keys 'weights' and 'biases' respectively.

```
hidden_2_layer = {"weights":tf.Variable(tf.random.normal([n_nodes_hl1,n_nodes_hl2])),
                  "biases":tf.Variable(tf.random.normal([n_nodes_hl2]))}
```

As explained in section 1.3.1, each connection between layers and nodes will have their separate weight and each node of each layer will have a bias, meaning that weights need to be stored in tensor of shape:

(number_of_nodes_in_first_layer, number_of_nodes_in_second_layer)

And biases will be stored in tensor of shape (number_of_nodes_in_first_layer), as shown in the code. All values within the tensors would start off being random numbers.

After the assignment of the layers, one of the instructions that you would see is as follow:

```
l1 = tf.add(tf.matmul(data, hidden_1_layer["weights"]), hidden_1_layer["biases"])
l1 = tf.nn.relu(l1)
```

What this is doing is exactly the same as what was mentioned in section 1.3.1: 'relu (dot product (W, input) + b), where W represents weight, b represents bias' which is a linear transformation on inputted data with non-linearity introduced through the relu activation function.

The entire tf code explained above is represented in Keras through 2 simple lines of code:

```
model.add(Dense(500))
model.add(Activation('relu'))
```

Moving onto the train_neural_network() function, first few lines are straight forward, where the where the loss(cost) function, optimizer and the number of epochs are assigned.

A session is then created and ran and the placeholders x and y are initialised. It then goes into a training loop of 10 epochs of which for each iteration, a nested loop of (total number of train data / batch size) iterations is ran.

```
with tf.compat.v1.Session() as sess:
    sess.run(tf.compat.v1.global_variables_initializer())
    for epoch in range(hm_epochs):
        epoch_loss = 0
        for _ in range(int(mnist.train.num_examples/batch_size)):
            epoch_x, epoch_y = mnist.train.next_batch(batch_size)
            _, c = sess.run([optimizer, cost], feed_dict = {x: epoch_x, y:epoch_y})
```

Within this nested loop, each iteration draws small batches (100) of images from the dataset and is fed into the x placeholder, their respective labels are fed into the y placeholder.

These are then fed into the neural network model along with the optimizer and loss(cost) function to adjust its weights and biases.

This entire function can is equivalent with the Keras code:

```
model.compile(loss='categorical_crossentropy',
              optimizer='adam',  # Good default optimizer to start with
              metrics=['accuracy'])  # what to track

model.fit(x_train, y_train, batch_size = 100 ,epochs=10)  # train the model
```

2.6.1.3 CNN Model

The previous section should give a general overview of how Keras and tf code are linked with each other, this section would discuss more about CNN which is more relevant to my investigation project.

The code below is excluding the train_neural_network() function which would be exactly the same as the one in previous section, other than the optimizer and loss function used.

It is also used to recognize digit images within the MNIST dataset, the code in tf is borrowed from sentdex[25], and below that is the equivalent Keras code that I have coded.

```
mnist = input_data.read_data_sets("/tmp/data/",one_hot = True)

n_classes = 10

x = tf.compat.v1.placeholder("float",name ="input/images")
```

[25] https://pythonprogramming.net/cnn-tensorflow-convolutional-nerual-network-machine-learning-tutorial/

```
y = tf.compat.v1.placeholder("float", name = "input/labels")


def conv2d(x,W):
   return tf.nn.conv2d(x,W,strides= [1,1,1,1],padding = "SAME")

def maxpool2d(x):
   return tf.nn.max_pool(x,ksize = [1,2,2,1],strides=[1,2,2,1],padding = "SAME")


def convolutional_neural_network(x):
   weights = {"W_conv1":tf.Variable(tf.random.normal([5,5,1,32])),
           "W_conv2":tf.Variable(tf.random.normal([5,5,32,64])),
           "W_fc":tf.Variable(tf.random.normal([7*7*64,1024])),
           "out":tf.Variable(tf.random.normal([1024,n_classes]))}

   biases = {"b_conv1":tf.Variable(tf.random.normal([32])),
           "b_conv2":tf.Variable(tf.random.normal([64])),
           "b_fc":tf.Variable(tf.random.normal([1024])),
           "out":tf.Variable(tf.random.normal([n_classes]))}
   x = tf.reshape(x,shape = [-1,28,28,1])
   conv1 = tf.nn.relu(conv2d(x,weights["W_conv1"]) + biases["b_conv1"])
   conv1 = maxpool2d(conv1)
   conv2 = tf.nn.relu(conv2d(conv1,weights["W_conv2"]) + biases["b_conv2"])
   conv2 = maxpool2d(conv2)
   fc = tf.reshape(conv2, shape = [-1,7*7*64])
   fc = tf.nn.relu(tf.matmul(fc,weights["W_fc"]) + biases["b_fc"])
   output = tf.matmul(fc,weights["out"]) + biases["out"]
   return output
```

```
(x_train, y_train),(x_test, y_test) = mnist.load_data()

model = Sequential()

model.add(Conv2D(32, (5, 5), input_shape=X.shape[1:], padding = "SAME"))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (5, 5), input_shape=X.shape[1:], padding = "SAME"))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(1024,activation = 'relu'))
```

```
model.add(Dense(10,activation = 'relu'))
```

The first 2 functions conv2d () and maxpool2d () are essentially used so that the kernel size (2 * 2), strides and padding ('SAME') parameters don't have to be repeated every time the built-in tf.nn.conv2d() is called.

Within the convolutional_neural_network() function, weights and biases are stored in a dictionary in the same way as before in the previous section, although they might look slightly different.

2 stacks on convolutional layer and max pooling layer are used within this model. The first convolutional layer has 32 nodes and uses a window size of 5*5, and the second convolutional layer has 64 nodes and uses the same window size.

Note that in Keras, the dimension of input tensor doesn't need to be specified as it automatically adjusts according to the input, only the output dimension need to be specified.

The output tensor of the 2 stacks are flattened to 1 dimension and fed into a fully connected layer of 1024 nodes which is fed into another fully connected layer of 10 nodes as there are only 10 classes (0……9). All layers use relu as activation function.

- ## 2.6.2 Development Stages for CNN Neural Network

There have been several stages in development and learning of the CNN which can be split into three sub sections: Binary class classification, Multiclass classification and finally 26 letters classification.

All images shown are generated with the aid of TensorBoard along with TensorFlow.

2.6.2.1 Binary class classification problems

I have started off my learning by using the book[26] and using various YouTube videos made by sentdex[27]. One of the videos taught me and led me into making my first Binary class classifications on recognizing images of cats and dogs.

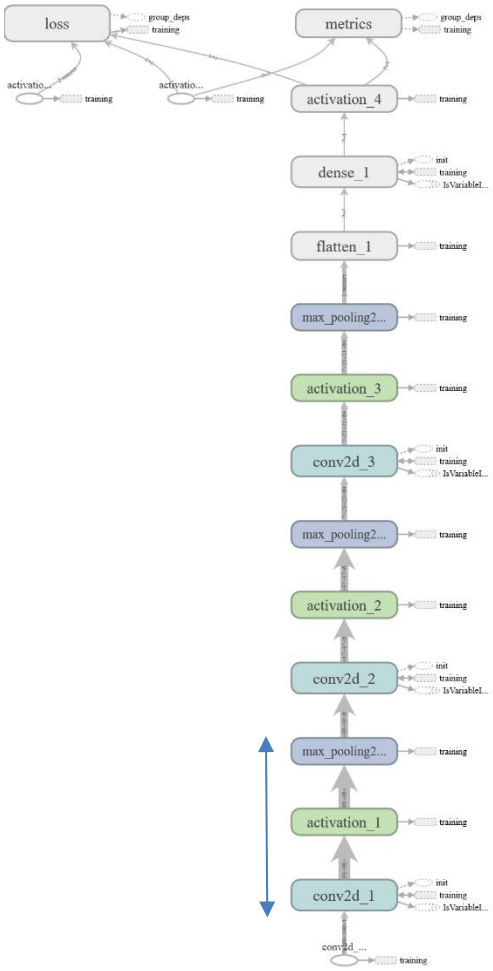
The CNN was trained using images found of kaggle and uses CNN architecture that is pre-designed by sentdex.

The architecture is as shown below, with a continuous stacks of max pooling layer on top of the convolutional layer (conv2d) along with activation functions (represented by the blue arrows), which is then flattened and fed into fully connected layer (dense layer) at the end.

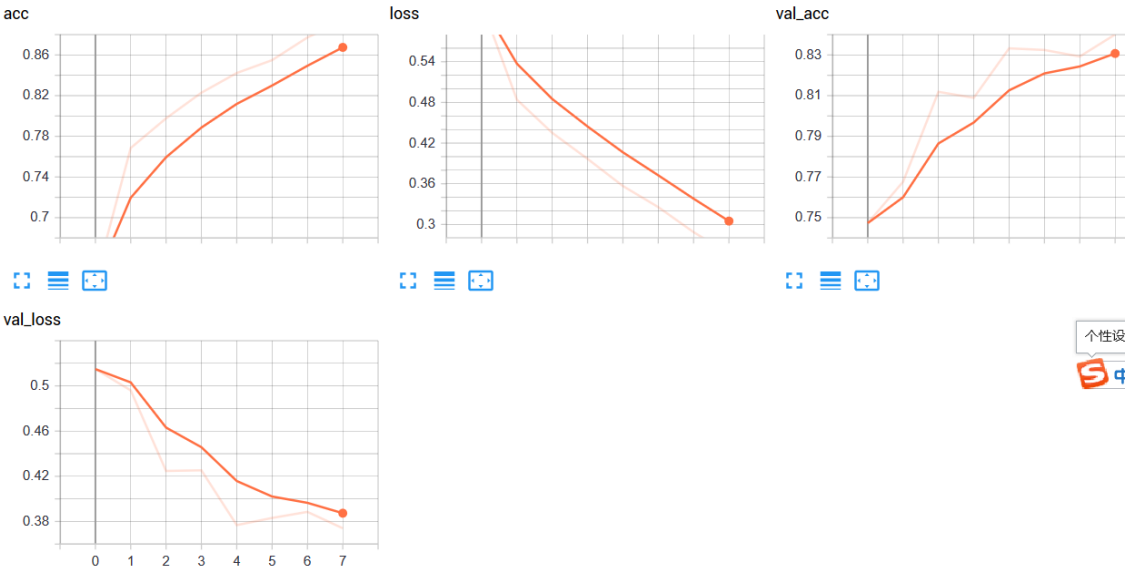
Loss on the top left corner represents the loss function. Details of these layers and functions have already been explained thoroughly in sections 1.3.1 and 1.3.2:

[26] See section 1.7.1

[27] https://www.youtube.com/watch?v=wQ8BIBpya2k



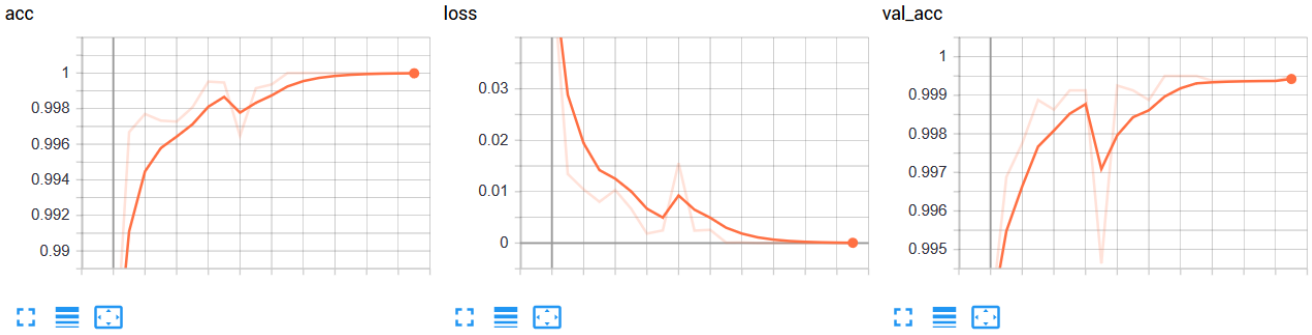
The overall performance is slightly disappointing though with a validation accuracy of 83 percent and a validation loss of 39 percent at the end of the epochs. Which shows that this architecture might not have been the best for this type of task.



Investigation on Image Processing Methods and how machines can learn to recognize hand-written English text

After finishing the Dogs and Cats project, I moved onto recognizing images of handwritten letters of capital A and B and adapted the model to raise its validation accuracy, which ended up as a great success, with a validation accuracy of almost 100 percent and negligible validation loss.

There are also no obvious signs of overfitting or underfitting which is satisfying.



2.6.2.2 Multiclass classification problems

After my various attempts in Binary class classification problem, and believe that I have fully mastered it, I moved onto multiclass classification problems which are slightly more difficult and more relevant with my investigation project.

I have decided to start off with some small datasets rather than going straight into the fully alphabet, for the purpose of learning and testing as it takes less time to process input data and to train.

The initial dataset that I chose was capital handwritten letters of A, B and C, and after completing this model, I adapted it so that it could also recognize handwritten lower-case letters a, b and c.

The dataset of these images is obtained through the NIST Special Database 19[28].

Unlike Binary classification problems, where labels of the classes can be represented by 0 and 1, Multiclass classification problems relies on something called the one hot encoded array. The image below gives a good demonstration of what it is:

Label Encoding

Food Name	Categorical #	Calories
Apple	1	95
Chicken	2	231
Broccoli	3	50

→

One Hot Encoding

Apple	Chicken	Broccoli	Calories
1	0	0	95
0	1	0	231
0	0	1	50

https://medium.com/@michaeldelsole/what-is-one-hot-encoding-and-how-to-do-it-f0ae272f1179

[28] See section 1.7.13

As you can see, the one hot encoded array is an array of equal length of the number of labels there are, where only one index within the array is a 1 and others are 0. The location of where the 1 is represents what class it is representing.

Below is an algorithm that I have created in order to convert labels into one hot encoded array, specifically for the A, B, C recognition task:

```
def one_hot(labels, dimension = 3):
    results = np.zeros((len(labels),dimension))
    for i, label in enumerate(labels):
        results[i,label] = 1
    return results
```

Before converting to one hot encoded array, the labels must be assigned a categorical number as shown above. In the current case, labels would be [0, 1, 2] rather than [1, 2, 3].

Dimension is set to 3 as there are only 3 classes, same as the image above, and the results variable will be initialised as a 3 * 3 array with all values being 0.

Then the program enumerates through the labels list, with i being the counter and label being the value within the labels list (e.g. 1, 1; 2, 2; 3, 3 would be the values of i and label respectively) and assigning index [i, lable] the value 1.

- ## 2.6.3 Recognizing Handwritten Letters

The complete version of the model is still in progress; hence I would be using the finished model that recognizes images of capital A, B, C as an example.

Hopefully the architecture of both models would be similar, which completely depends on more evaluation of the model through testing, and obviously the complete version would also have a far larger amount of class labels and training data.

The architecture[29] of this model is as shown below, which has proven through various experiments to be 'the best' for this current task:

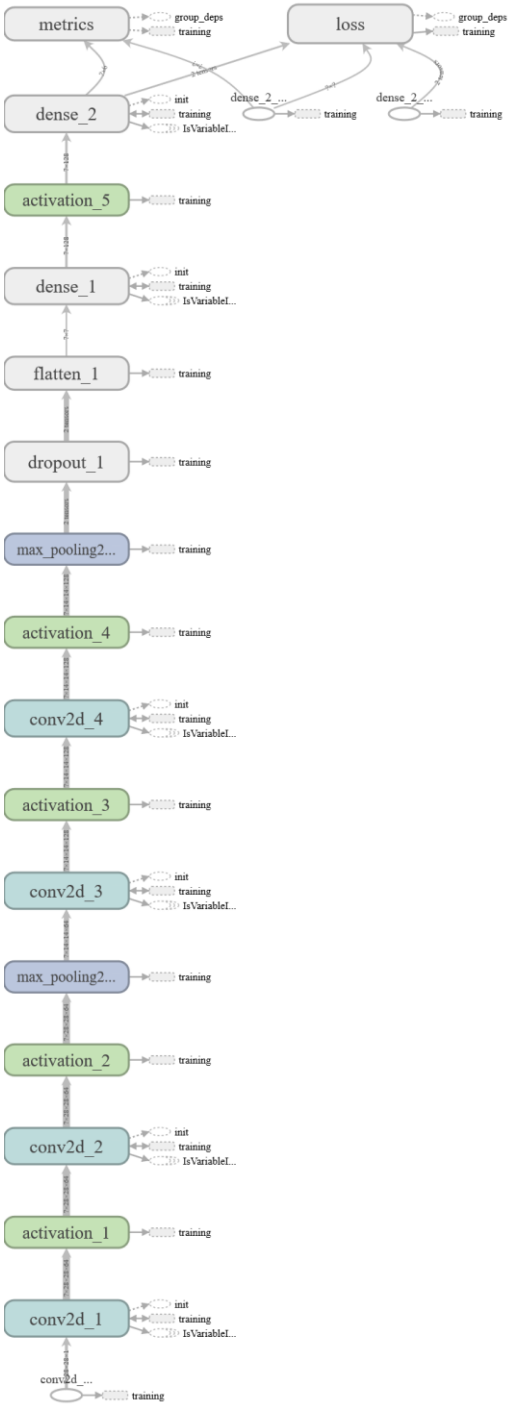
2 repeated stacks of convolutional layer with 64 nodes and window/kernel size of 5 by 5 and a relu activation function has been added, which is then followed by a single max pooling layer of pool/window size of 2 by 2.

This is once again repeated, but now the convolutional layer has 128 nodes each and a window/kernel size of 3 by 3. Note that all layers have 'SAME' padding.

[29] Probably need to reference

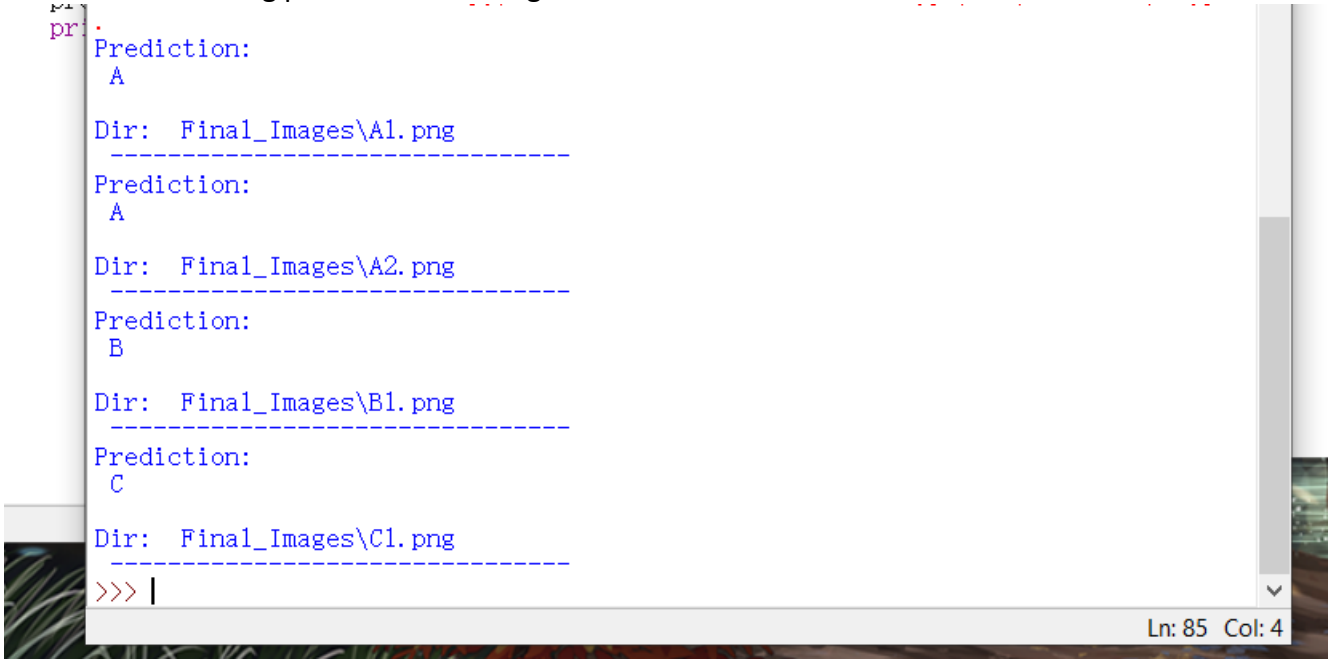
Afterwards, an additional dropout layer has been added which is primarily used to prevent overfitting from occurring. This is achieved by randomly setting some fraction of the input to 0 during training time, and the fraction is specified by the rate parameter which is currently set as 0.2.

The output tensor is then flattened and inputted into the fully connected layer (dense layer) of 128 nodes with a relu activation as well.

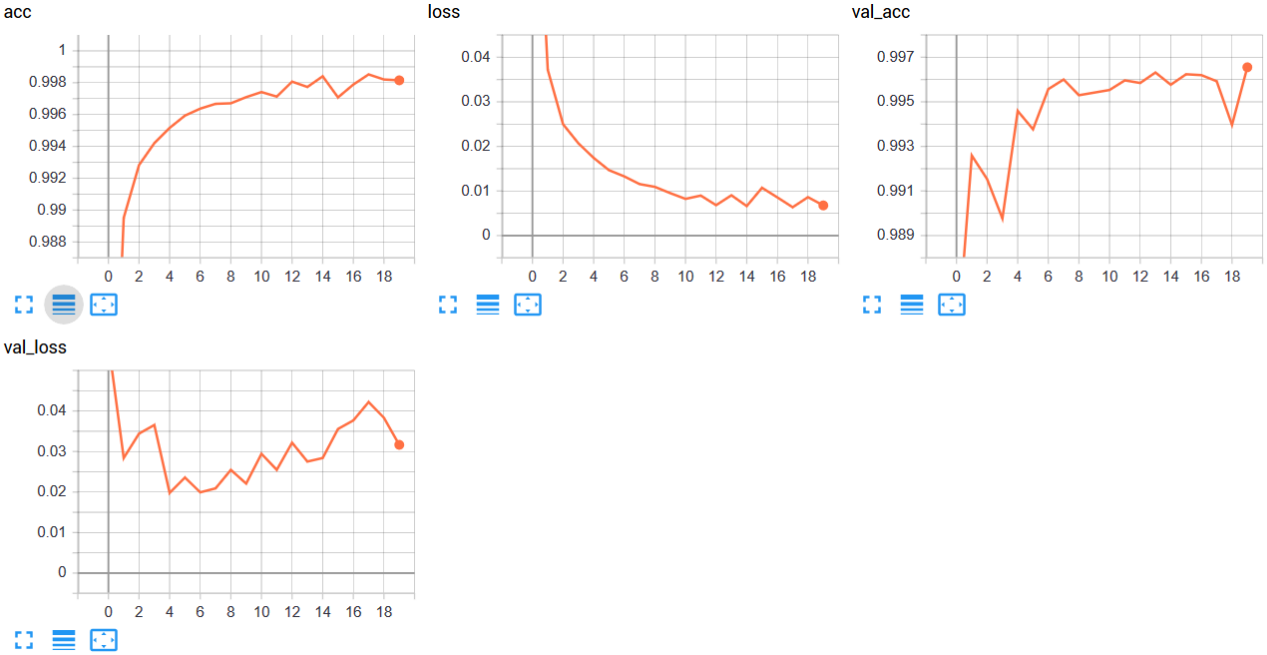


At the end, a fully connected layer of 6 nodes and softmax activation function is added in order to make a final prediction on what class does the input image belong in. The prediction outputted would be a NumPy array of 6 numbers floating between 0 to 1 as softmax function outputs numbers between 0 to 1(probability).

This is then converted into a list with the element of greatest value becoming a 1 and others become zero, which forms a one hot array representing what class the image belong in. Below is an example of this model making predictions on images.



The validation accuracy of this model reached 0.9968 and has a validation loss of 0.031, of which I believe is satisfactory for a classification task of 3 classes. There is a clear sign of overfitting by looking at the graph of validation loss, which would require the model to be trained again using a smaller number of epochs



- 2.6.4 Improving accuracy of final prediction

Below are the images of number of raw images samples that I have obtained through NIST and NMIST datasets:

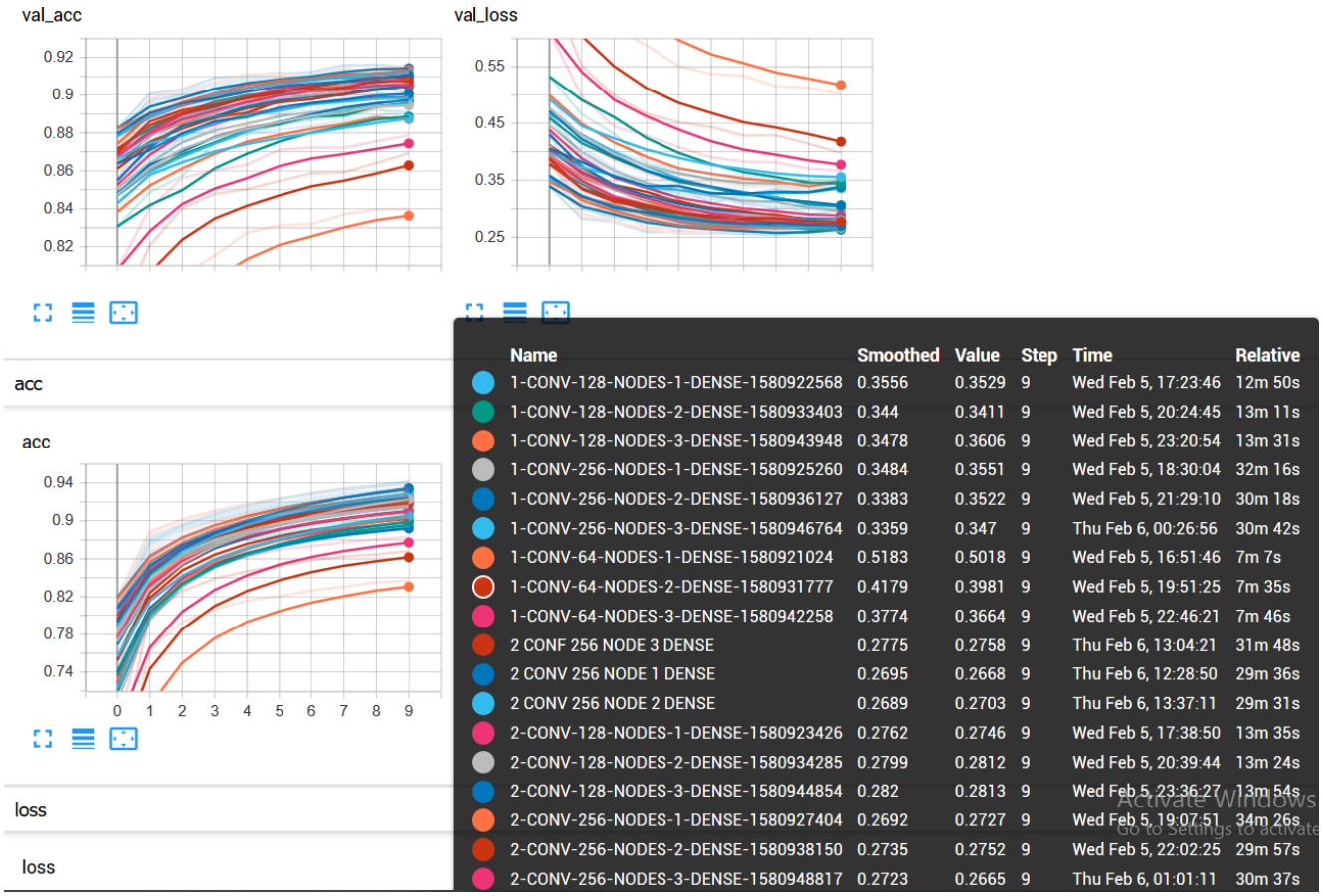
```
A: 13,870 images
A Lower: 14,280 images
B: 13,659 images
B Lower: 11,561 images
C: 12,015 images
C Lower: 6,068 images
D: 11,184 images
D Lower: 11,770 images
E: 11,440 images
E Lower: 14,430 images
F: 10,620 images
F Lower: 5,453 images
G: 11,301 images
G Lower: 8,115 images
H: 12,216 images
H Lower: 12,089 images
I: 10,840 images
I Lower: 5,940 images
J: 12,455 images
J Lower: 4,133 images
K: 10,926 images
K Lower: 5,519 images
L: 11,586 images
L Lower: 12,632
M: 12,336 images
M Lower: 5,743 images
N: 12,063 images
N Lower: 12,856 images
O: 12,115 images
O Lower: 5,976 images
```

```
P:  12,172 images
P Lower: 5,217 images
Q: 11,396 images
Q Lower: 6,614 images
R: 11,566 images
R Lower: 12,190 images
S: 12,285 images
S Lower: 11,529 images
T: 12,030 images
T Lower: 12,155 images
U: 12,267 images
U Lower: 12,094 images
V: 11,914 images
V Lower: 6,232 images
W: 10,784 images
W Lower: 10,889 images
X: 12,206 images
X Lower: 6,112 images
Y: 10,859 images
Y Lower: 5,105 images
Z: 11,939 images
Z Lower: 5,902 images
```

This introduces a big problem as the smallest number of images I have in the alphabet classes is 4,133 images and the greatest is 14,280. CNN model trained using this dataset would be extremely biased as it is more exposed to some classes than another.

Compromising is needed, so the number of sample images in each class has to be smaller. 5000 images per class have been decided to be a good amount. Using relatively small dataset means that the model wouldn't be able to achieve high accuracy scores (>95%) without overfitting from happening, hence a method of improving the accuracy of final prediction is needed

The concept of majority voting is used while developing this method. Multiple CNN models would be trained, these models would have different number of convolutional layers, number of nodes per layer and number of fully connected layers. Below is an image of graphs produced using Tensorboard of the validation accuracy and validation loss scores of the variety of CNN models that I have trained:



Name	Smoothed	Value	Step	Time	Relative
1-CONV-128-NODES-1-DENSE-1580922568	0.3556	0.3529	9	Wed Feb 5, 17:23:46	12m 50s
1-CONV-128-NODES-2-DENSE-1580933403	0.344	0.3411	9	Wed Feb 5, 20:24:45	13m 11s
1-CONV-128-NODES-3-DENSE-1580943948	0.3478	0.3606	9	Wed Feb 5, 23:20:54	13m 31s
1-CONV-256-NODES-1-DENSE-1580925260	0.3484	0.3551	9	Wed Feb 5, 18:30:04	32m 16s
1-CONV-256-NODES-2-DENSE-1580936127	0.3383	0.3522	9	Wed Feb 5, 21:29:10	30m 18s
1-CONV-256-NODES-3-DENSE-1580946764	0.3359	0.347	9	Thu Feb 6, 00:26:56	30m 42s
1-CONV-64-NODES-1-DENSE-1580921024	0.5183	0.5018	9	Wed Feb 5, 16:51:46	7m 7s
1-CONV-64-NODES-2-DENSE-1580931777	0.4179	0.3981	9	Wed Feb 5, 19:51:25	7m 35s
1-CONV-64-NODES-3-DENSE-1580942258	0.3774	0.3664	9	Wed Feb 5, 22:46:21	7m 46s
2 CONF 256 NODE 3 DENSE	0.2775	0.2758	9	Thu Feb 6, 13:04:21	31m 48s
2 CONV 256 NODE 1 DENSE	0.2695	0.2668	9	Thu Feb 6, 12:28:50	29m 36s
2 CONV 256 NODE 2 DENSE	0.2689	0.2703	9	Thu Feb 6, 13:37:11	29m 31s
2-CONV-128-NODES-1-DENSE-1580923426	0.2762	0.2746	9	Wed Feb 5, 17:38:50	13m 35s
2-CONV-128-NODES-2-DENSE-1580934285	0.2799	0.2812	9	Wed Feb 5, 20:39:44	13m 24s
2-CONV-128-NODES-3-DENSE-1580944854	0.282	0.2813	9	Wed Feb 5, 23:36:27	13m 54s
2-CONV-256-NODES-1-DENSE-1580927404	0.2692	0.2727	9	Wed Feb 5, 19:07:51	34m 26s
2-CONV-256-NODES-2-DENSE-1580938150	0.2735	0.2752	9	Wed Feb 5, 22:02:25	29m 57s
2-CONV-256-NODES-3-DENSE-1580948817	0.2723	0.2665	9	Thu Feb 6, 01:01:11	30m 37s

Within all these trained models, I have selected 5 different models on top of the model described in section 2.6.3, as they produced the lowest validation loss scores:

3 CONV 128 NODE 1 DENSE	0.264
3 CONV 256 NODE 1 DENSE	0.2636
2 CONV 256 NODE 1 DENSE	0.2692
2 CONF 256 NODE 3 DENSE	0.2723
2 CONV 256 NODE 2 DENSE	0.2735

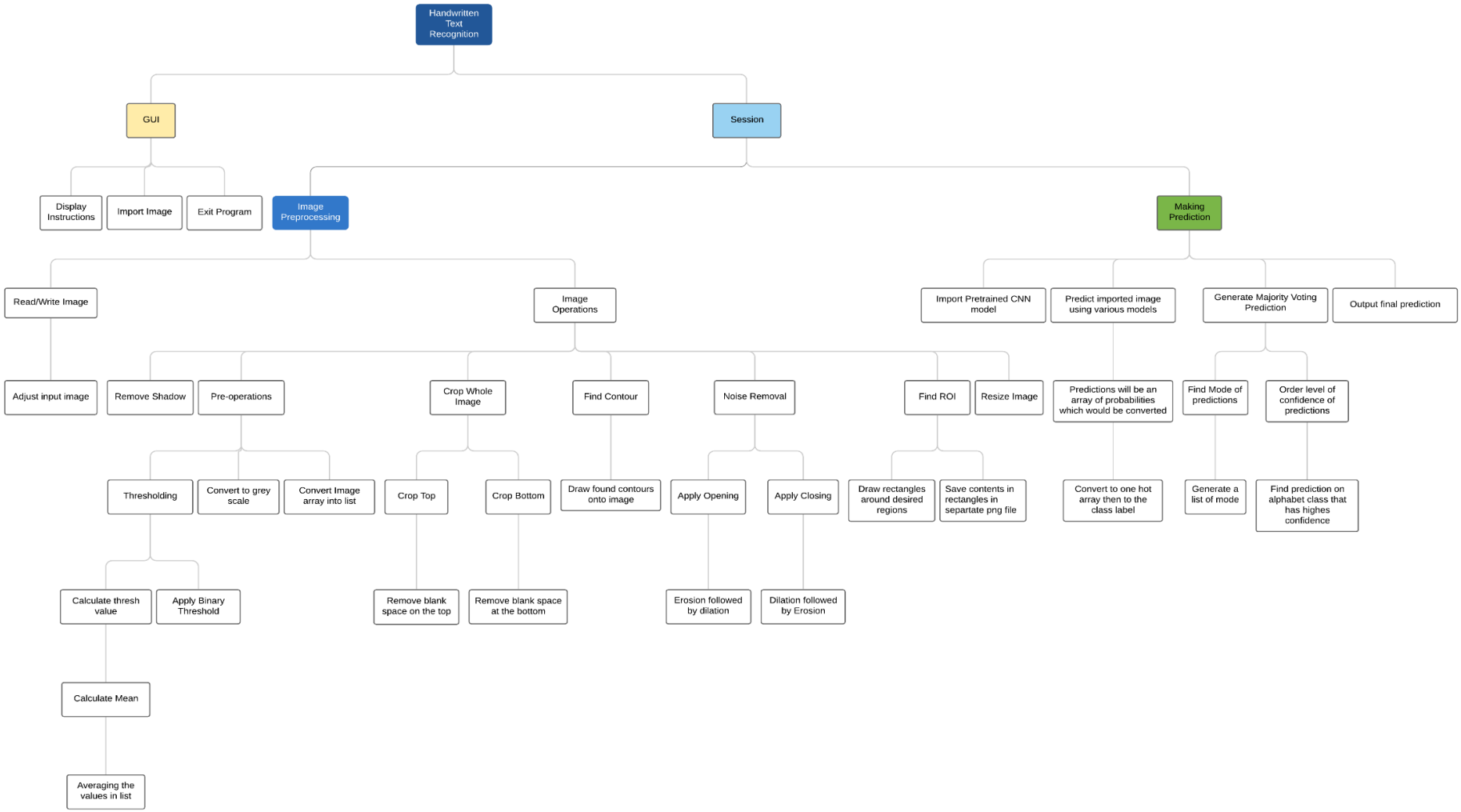
For each prediction on what alphabet class does the input image belong in, 6 predictions generated by the 6 different models will be produced and saved along with their confidence level on whether their prediction is right or not. The confidence levels would be a float number between 0 and 1 and will be saved and sorted from lowest confidence to highest.

If all models generate same alphabet prediction, then this prediction would obviously be the final prediction. If the predictions aren't all the same and all predictions have the same confidence level, then the mode of the alphabet predictions (most frequently appearing element) would be found and this mode would become the final prediction. In case of having more than one mode, hence multiple class appear at same amount of times, then one of the modes would be randomly selected to become the final prediction.

If the confidence level of the predictions isn't the same, then the most confident prediction (highest confidence level) on alphabet class would become the final prediction, this would be achieved with the use of a stack. Final prediction would then be either outputted directly if only recognizing a single letter, or would be concatenated into a string of text before outputting onto console, if recognizing entire text.

Continue on next page

2.7 Modular Structure of the System



3. TECHNICAL SOLUTION

3.0. Complex Code Reference

3.1. GUI class

- ***Initialisation***

```
1. from tkinter import *
2. from tkinter.filedialog import askopenfilename
3. from tkinter.messagebox import showerror
4. from tkinter import messagebox
5.
6. class GUI(Frame):
7.     def __init__(self, window_pos, master=None):
8.         Frame.__init__(self, master)
9.         self.operation = ""
10.        self.master = master
11.        self.master.filepath = ""
12.        self.window_pos = window_pos
13.        self.init_window()
```

Line 6: GUI class inherits from Frame class which is a built-in class within Tkinter library
Line 8: Calls constructor of the Frame parent class to initialise all inherited attributes, the root widget Tk() is passed as parameter into the Frame .
Line 13: Creates the main window, would be mentioned later in the section

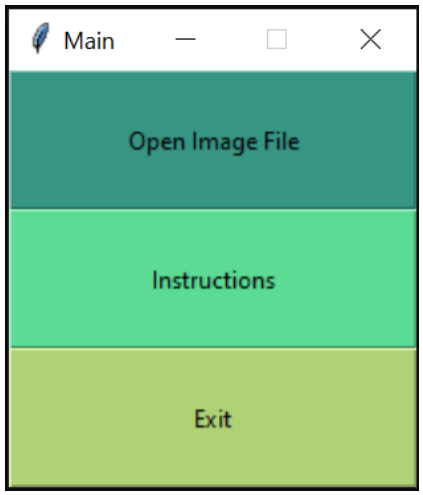
- ***Table of attributes***

Identifier	Purpose

Self.operation	String variable that stores the operation choice that the user selected on the Operation option window
Self.master	Master is a widget within the Frame, parent class and is None by default. However, to utilise the frames, a root widget Tk() will be passed as and saved within this attribute
Self.master.filepath	String variable that stores the full file directory of the imported image
Self.window_pos	List variable that stores a list of [x,y] coordinates, in order for windows to be positioned at the centre of the screen

- ***init_window method***

Purpose: Used to create the main window with three buttons on it: "Open_Image", "Instructions" and "Exit" button

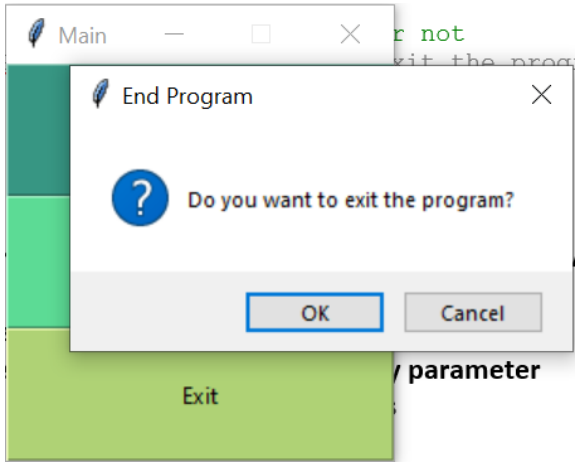



```
def init_window(self):
        self.master.title("Main") #Name of the frame
```

```
        self.pack(fill=BOTH, expand=1) #Widget fill empty space
        #Objective 1.a
        input_button = Button(self,text = "Open Image File",width = 28,
                              height = 4, bg = "#379683", fg ="Black",
                              command = self.Open_Image) #Open_Image button
        instruction_button =Button(self,text = "Instructions",width = 28,
                                   height = 4,bg = "#5CDB95",fg ="Black",
                                   command = self.display_instruction)
                                    #Instructions Button
        exit_button = Button(self,text = "Exit", width = 28, height = 4,
                             bg = "#AFD275",fg ="Black",
                             command = self.quitting)#Objective 1.h
        input_button.place(x=0,y=0)
        instruction_button.place(x=0,y=70)
        exit_button.place(x=0,y=140)
        #Placing of buttons with specified coordinates and place them on frame

        #Set coordinates of buttons and place them on the frame

        #Calls self.quitting method before exiting the main window
        self.master.protocol("WM_DELETE_WINDOW",self.quitting)
```

- ***Quitting method***

Purpose: Only used by the main window, when user decided to close the main window by pressing the Exit button or by pressing the cross on top right corner, a message is displayed asking whether user would like to exit or not. If true then main window is closed and program terminates




```
def quitting(self):
        #Display message asking whether user would like to quit or not
        if messagebox.askokcancel("End Program","Do you want to exit the program?"):
            self.master.destroy()
```

- ***Format_window method***

Purpose: Method called when creating the Instruction window and Operation option window (later on in this section) and does the following –

1. **Formats the windows so that they're positioned at the centre of the screen**
2. **Ensure that they're not resizable and are named with string specified by parameter**
3. **Ensure that the window is shown on top of all other displayed windows**
4. **Hides the main window**
5. **Apply protocol that redisplays main window when the current window is closed**

```
def format_window(self,window,name):
        #Format window so that they're displayed at the centre of the screen and aren't resizable
        window.geometry('250x210')
```

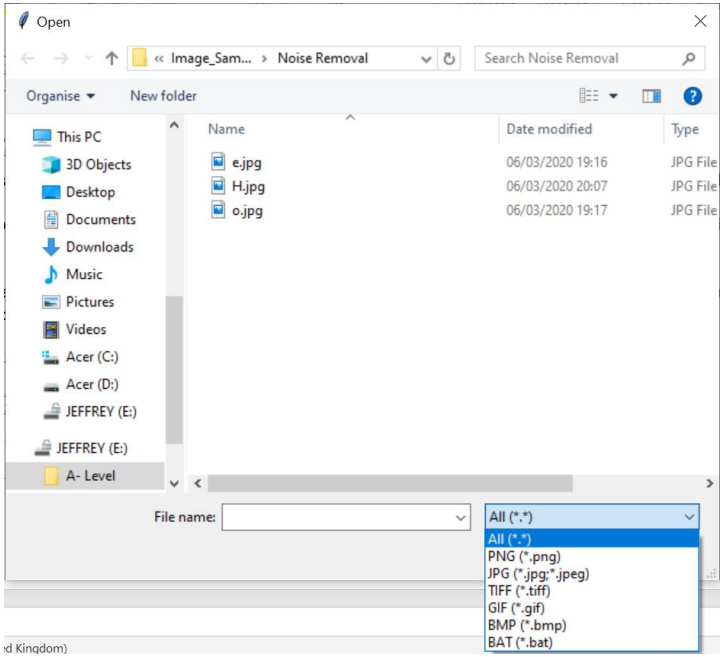
```
        window.geometry("+{}+{}".format(self.window_pos[0],self.window_pos[1]))
        window.resizable(False,False)
        window.title(name)
        window.attributes('-topmost','true')
        self.master.withdraw() #Hide main Frame
        window.protocol("WM_DELETE_WINDOW",lambda:[window.destroy(),self.master.deiconify()])
        #Redisplay main window if the window is closed
```

- ***Open_Image method***

Purpose: Opens file selecting window to allow user to import a single file. If the file selecting window is closed, main window is redisplayed.

If they file imported has a file extension that is not in the extensionsToCheck local list constant, then an error is shown and main window is redisplayed

If the file extension is in the list, then the directory to that file is saved within the self.master.filepath attribute and the operation_option method is called to display Operation option window



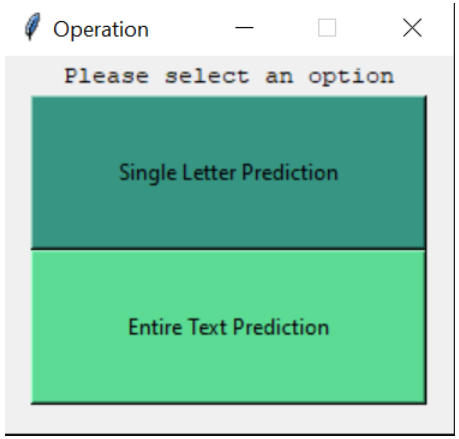

```
def Open_Image(self):
        #Objective 1.b
        extensionsToCheck = [".png",".jpg",".jpeg",".tiff",".gif",".bmp",".bat"]
        self.master.withdraw() #Hides main window
        #File extensions of image files, may need to add more
        file = askopenfilename(filetypes=(("All", "*.*"),
                                          ("PNG", "*.png"),
                                          ("JPG", "*.jpg;*.jpeg"),
                                          ("TIFF", "*.tiff"),
                                          ("GIF", "*.gif"),
                                          ("BMP", "*.bmp"),
                                          ("BAT", "*.bat")))
        #Ask user to input file in order to obtain file directory, may need to add more
        if file.lower() == "":
            #Redisplay main window if file selecting window is closed
            self.master.deiconify()
```

```
        elif not any(ext in file.lower() for ext in extensionsToCheck):
            #Objective 1.c
            #Ensures file inputted is an image file
            showerror(title = "Error",message = "Unknown filetype inputted")
            #Raise error, display error window
            label = Label(self,text = ("Please input an image file"))
            label.config(font=("Courier", 10))
            label.place(x=250,y=100)
            #Objective 1.f
            self.master.deiconify() #Redisplay main window
        else:
            self.master.filepath = file #File directory
            self.operation_option()
```

- ***Operation_option method***

Purpose: Creates a window that has 2 buttons: "Single Letter Prediction" and "Entire Text Prediction". Allow the user to select the desired operation (Single letter / Entire text), by clicking the appropriate button.

Button clicked would call its corresponding method, either single_letter or entire_text method.



```
def operation_option(self):
        #Objective 1.d
        window = Toplevel() #Creating a frame on top of the root frame
        self.format_window(window,'Operation')
        label = Label(window,text = ("Please select an option"),width = 40)
        label.config(font=("Courier", 10))
        label.pack()
        #Specify the text to display on button, width/height of button, back/foreground of button and the command
        single_letter_button = Button(window,text = "Single Letter Prediction", width = 30,height = 5,
                                      bg = "#379683", fg ="Black",command = self.single_letter).pack()
        entire_text_button = Button(window,text = "Entire Text Prediction",
                                    width = 30,height = 5, bg = "#5CDB95",fg ="Black",
                                    command = self.entire_text).pack() #Buttons
```

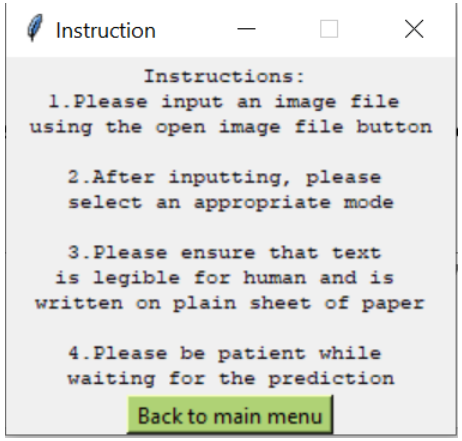
- ***Single_letter and entire_text method***

Purpose: Stores the corresponding string value of operation type (“Single” or “Entire”) into the self.operation attribute and closes the all windows displayed. However, the program doesn’t terminate just yet.

```
def single_letter(self):
        #Objective 1.e
        self.operation = ("Single")
        self.master.destroy()

def entire_text(self):
        #Objective 1.e
        self.operation =("Entire")
        self.master.destroy()
```

- ***Display_instruction method***

Purpose: Creates the Instruction window with instructions displayed as text. This method is called when the “Instruction” button is pressed on the main window




```
def display_instruction(self):
        #Objective 1.g
        window = Toplevel() #Create another frame
        self.format_window(window,'Instruction')
        #Instructions to be displayed
        label = Label(window,text = ("Instructions: \n1.Please input an image file \nusing the open image file
button\n\n2.After inputting, please \nselect an appropriate mode\n\n3.Please ensure that text \nis legible for human
and is \nwritten on plain sheet of paper\n\n4.Please be patient while \nwaiting for the prediction" ),width = 40)
        label.config(font=("Courier", 8))
        label.pack()
        return_menu_button = Button(window,text = "Back to main menu",
                                    width = 15, bg = "#AFD275", fg ="Black",
                                    command = lambda:[window.destroy(),self.master.deiconify()]).pack()
        #Destroy instruction window and re-display main Frame window
```

- ***Return_image_directory and return_operation_type methods***

Purpose: Returns the value stored within self.master.filepath attribute or self.operation attribute, used within Session class (see section *3.7*) to retrieve these values.

```
    def return_image_directory(self): #Return extracted image directory
        return self.master.filepath
```

```
    def return_operation_type(self):  #Return selected operation type
        return self.operation
```

- ***Full Code***

```
from tkinter import *
from tkinter.filedialog import askopenfilename
from tkinter.messagebox import showerror
from tkinter import messagebox

class GUI(Frame):
    def __init__(self, window_pos, master=None):
        Frame.__init__(self, master)
        self.operation = ""
        self.master = master
        self.master.filepath = ""
        self.window_pos = window_pos
        self.init_window()

    def init_window(self):
        self.master.title("Main") #Name of the frame

        self.pack(fill=BOTH, expand=1) #Widget fill empty space
        #Objective 1.a
        input_button = Button(self,text = "Open Image File",width = 28,
                              height = 4, bg = "#379683", fg ="Black",
                              command = self.Open_Image) #Open_Image button
        instruction_button =Button(self,text = "Instructions",width = 28,
                                   height = 4,bg = "#5CDB95",fg ="Black",
                                   command = self.display_instruction)
                                    #Instructions Button
        exit_button = Button(self,text = "Exit", width = 28, height = 4,
                             bg = "#AFD275",fg ="Black",
                             command = self.quitting)#Objective 1.h
        input_button.place(x=0,y=0)
        instruction_button.place(x=0,y=70)
        exit_button.place(x=0,y=140)
```

```
        self.master.protocol("WM_DELETE_WINDOW",self.quitting)
        #Placing of buttons with specified coordinates and place them on frame

        #Set coordinates of buttons and place them on the frame

    def quitting(self):
        #Display message asking whether user would like to quit or not
        if messagebox.askokcancel("End Program","Do you want to exit the program?"):
            self.master.destroy()

    def format_window(self,window,name):
        #Format window so that they're displayed at the centre of the screen and aren't resizable
        window.geometry('250x210')
        window.geometry("+{}+{}".format(self.window_pos[0],self.window_pos[1]))
        window.resizable(False,False)
        window.title(name)
        window.attributes('-topmost','true')
        self.master.withdraw() #Hide main Frame
        window.protocol("WM_DELETE_WINDOW",lambda:[window.destroy(),self.master.deiconify()])
        #Redisplay main window if the window is closed
    def Open_Image(self):
        #Objective 1.b
        extensionsToCheck = [".png",".jpg",".jpeg",".tiff",".gif",".bmp",".bat"]
        self.master.withdraw() #Hides main window
        #File extensions of image files, may need to add more
        file = askopenfilename(filetypes=(("All", "*.*"),
                                          ("PNG", "*.png"),
                                          ("JPG", "*.jpg;*.jpeg"),
                                          ("TIFF", "*.tiff"),
                                          ("GIF", "*.gif"),
                                          ("BMP", "*.bmp"),
                                          ("BAT", "*.bat")))
        #Ask user to input file in order to obtain file directory, may need to add more
        if file.lower() == "":
            self.master.deiconify()
        elif not any(ext in file.lower() for ext in extensionsToCheck):
            #Objective 1.c
            #Ensures file inputted is an image file
            showerror(title = "Error",message = "Unknown filetype inputted")
```

```
            #Raise error, display error window
            label = Label(self,text = ("Please input an image file"))
            label.config(font=("Courier", 10))
            label.place(x=250,y=100)
            #Objective 1.f
            self.master.deiconify() #Redisplay main window
        else:
            self.master.filepath = file #File directory
            self.operation_option()

    def operation_option(self):
        #Objective 1.d
        window = Toplevel() #Creating a frame on top of the root frame
        self.format_window(window,'Operation')
        label = Label(window,text = ("Please select an option"),width = 40)
        label.config(font=("Courier", 10))
        label.pack()
        single_letter_button = Button(window,text = "Single Letter Prediction",
                                      width = 30,height = 5, bg = "#379683", fg ="Black",
                                      command = self.single_letter).pack()
        entire_text_button = Button(window,text = "Entire Text Prediction",
                                    width = 30,height = 5, bg = "#5CDB95",fg ="Black",
                                    command = self.entire_text).pack() #Buttons

    def single_letter(self):
        #Objective 1.e
        self.operation = ("Single")
        self.master.destroy()

    def entire_text(self):
        #Objective 1.e
        self.operation =("Entire")
        self.master.destroy()

    def display_instruction(self):
        #Objective 1.g
        window = Toplevel() #Create another frame
        self.format_window(window,'Instruction')
        label = Label(window,text = ("Instructions: \n1.Please input an image file \nusing the open image file
```

```
button\n\n2.After inputting, please \nselect an appropriate mode\n\n3.Please ensure that text \nis legible for human
and is \nwritten on plain sheet of paper\n\n4.Please be patient while \nwaiting for the prediction" ),width = 40)
        label.config(font=("Courier", 8))
        label.pack()
        return_menu_button = Button(window,text = "Back to main menu",
                                    width = 15, bg = "#AFD275", fg ="Black",
                                    command = lambda:[window.destroy(),self.master.deiconify()]).pack()
        #Destroy instruction window and re-display main Frame window

    def return_image_directory(self): #Return extracted image directory
        return self.master.filepath

    def return_operation_type(self):  #Return selected operation type
        return self.operation
```

3.2. *Image_preprocess class*

- ***Initialisation***

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import os
import time

class Image_Preprocess():
    def __init__(self):
        self.DATADIR = ''
        self.img = None
        self.image_array = []
        self.thresh_value = 0
        self.count = 1
```

- ***Table of attributes***

Identifier	Purpose
Self.DATADIR	Stores the directory to the imported image

Self.img	Stores the image that is currently being pre-processed in NumPy array form to enable processing using opencv
Self.image_array	Stores the image that is currently being pre-processed in list form, to enable cropping and finding threshold value of the image
Self.thresh_value	Stores the threshold value of current image, used for binary thresholding
Self.count	Integer variable that is used for naming of the processed image files in ascending order. Please refer to CNN_prediction method of Session class for why this is necessary

- ***Read_image method***

Purpose: stores the image in NumPy array form specified by the DATADIR parameter into self.img attribute and store the value of DATADIR parameter into self.DATADIR attribute

```
def read_image(self,DATADIR):
        self.DATADIR = DATADIR
        self.img = cv2.imread(self.DATADIR)
```

- ***Remove_Shadow method***

Purpose: Removes shadow within the image, code taken from the forum:
https://stackoverflow.com/questions/44752240/how-to-remove-shadow-from-scanned-images-using-opencv

```
#Objective 2.d
    def Remove_Shadow(self, image):
        #https://stackoverflow.com/questions/44752240/how-to-remove-shadow-from-scanned-images-using-opencv
        rgb_planes = cv2.split(image)
        result_planes = []
        result_norm_planes = []
        for plane in rgb_planes:
            dilated_img = cv2.dilate(plane, np.ones((7,7), np.uint8))
            bg_img = cv2.medianBlur(dilated_img, 21)
            diff_img = 255 - cv2.absdiff(plane, bg_img)
```

```
            norm_img = cv2.normalize(diff_img,None, alpha=0, beta=255, norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_8UC1)
            result_planes.append(diff_img)
            result_norm_planes.append(norm_img)
        return cv2.merge(result_norm_planes)
```

- ***Pre_operations method***

Purpose: Convert image to grey scale, find its threshold value and apply binary thresholding to convert image into binary form. Then the contour line within image is found in order to thicken the letters and noise removal is applied to the image. The resulting image is returned.

```
def Pre_operations(self,image):
        #Apply gaussian blur, simple noise removal
        blurr_image = cv2.GaussianBlur(image,(5,5),cv2.BORDER_DEFAULT)
        #Objective 2.a
        grey_image = cv2.cvtColor(blurr_image,cv2.COLOR_RGB2GRAY)
        self.image_array = grey_image.tolist()
        self.thresh_value = self.calculate_thresh(self.image_array)
        #Objective 2.c
        ret, thresholded = cv2.threshold(grey_image,self.thresh_value,255,cv2.THRESH_BINARY + cv2.THRESH_OTSU)
        thresholded = cv2.bitwise_not(thresholded)
        contour,hier = cv2.findContours(thresholded,cv2.RETR_CCOMP,cv2.CHAIN_APPROX_SIMPLE)
        for cnt in contour:
            cv2.drawContours(thresholded,[cnt],0,255,24)
        thresholded = cv2.bitwise_not(thresholded)
        thresholded = self.noise_removal(thresholded,30)
        thresholded = cv2.bitwise_not(thresholded)
        self.img = thresholded
        return thresholded
```

- ***Crop_top_image and crop _bottom_image methods***

Purpose: Count the number of empty arrays (only contains pixel value of 255, white pixel) from top to bottom within the image array passed as parameter, line by line until a not empty array (meaning the array contains a black pixel) is met. Then remove that number of arrays while reserving ¼ of space. Return the resulting image array.

Image is already in binary form before applying these methods, or else it wouldn't work.

Both methods work in the exact same way, where crop_bottom_image does it in reverse (from bottom up rather than to bottom)

```
#Objective 2.g
def crop_top_image(self,image_array_input):
        image_array = image_array_input
        empty_space_count = 0
        length = len(image_array)
        for x in range(length):#FOR EACH ARRAY WITHIN THE 2D ARRAY
            if min(image_array[x]) == 255: #IF ALL ELEMENTS = 255, ADD 1 TO EMPTY SPACE COUNT
                empty_space_count += 1
            else:
                break #IF 1 ELEMENT WAS FOUND TO NOT BE 255 THEN MOVE ONTO THE NEXT ARRAY
        print("Top: ",empty_space_count)
        for y in range(int(empty_space_count * (3/4))):#RESERVE TO PIXELS TO NOT BE DELETED
            image_array.remove(image_array[0]) #FOR THE NUMBER OF COUNTS, REMOVE THE WHITE SPACES
        return image_array

    def crop_bottom_image(self,image_array_input): #SIMILAR PRINCIPLE AS CROP_TOP_IMAGE() FUNCTION
        image_array = image_array_input
        empty_space_count = 0
        for x in range ((len(image_array)-1),-1,-1): #ITERATE IN REVERSE
            if min(image_array[x]) == 255:
                empty_space_count += 1
            else:
                break
        print("Bottom: ",empty_space_count)
        for y in range(int(empty_space_count * (3/4))):
            del image_array[-1]
        return image_array
```

- ***Crop_whole_image method***

Purpose: Applies crop_top_image and crop_bottom_image methods to the image array, then rotate it by 270 degrees and do the same procedure again, in order to crop the sides of the image as well.

At the end rotate image back to original and returns the numpy array version of the image array.

```
#Objective 2.g
    def crop_whole_image(self,image_array_input):
        image_array = image_array_input
        image_array = self.crop_top_image(image_array)
        image_array = self.crop_bottom_image(image_array)#CROP TOP AND BOTTOM
        rotate_image = np.rot90(image_array,3) #ROTATE BY 270 DEGREES, EASIER TO CROP LEFT/RIGHT
        image_array = rotate_image.tolist() #CONVERT NUMPY ARRAY TO LIST
        image_array = self.crop_top_image(image_array)
        image_array = self.crop_bottom_image(image_array) #CROP LEFT AND RIGHT
        fully_cropped_image = np.rot90(image_array)#ROTATE IMAGE BACK TO ORIGINAL
        return np.asarray(fully_cropped_image, dtype = np.float32)
        #CONVERT LIST BACK TO NUMPY ARRAY IN ORDER TO BE DISPLAYED BY MATPLOTLIB
```

- ***Find_contour method***

Purpose: Finds contour lines within the image and draw them onto the original image using thicker contour lines. Return the resulting image with contour lines drawn.

```
    #Objective 2.f
    def find_contour(self,image):
        cv2.imwrite("tmp.png",image)
        image = cv2.imread("tmp.png")
        image = cv2.cvtColor(image,cv2.COLOR_RGB2GRAY)
        #Need to be in binary form before finding contour lines
        ret,thresh = cv2.threshold(image,self.thresh_value,255,cv2.THRESH_BINARY)
        #Find the contour lines within the image
        contours, hierachy = cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_NONE)
        for x in range(1,len(contours)):
            cnt = contours[x]
            cv2.drawContours(thresh, [cnt], 0, (0,0,0),3) #Drawn with black colour and thickened
        os.remove("tmp.png")
        return thresh
```

- ***Image_resize method***

Purpose: Resize the image specified by parameter with the size specified by parameter (dimension of size * size), and return the resulting image

```
#Objective 2.b
    def image_resize(self, image,SIZE):
        resize_img = cv2.resize(image,(SIZE,SIZE))
        ret,final_image = cv2.threshold(resize_img,self.thresh_value,255,cv2.THRESH_BINARY)
        return final_image
```

- ***Calculate_thresh and calculate_mean methods***

Purpose: Take in the image array as parameter (in list form) and calculate the mean value of the entire array by calculating mean for each line of array within the image array and calculate the mean of those means.

```
#Objective 2.c
    def calculate_thresh(self,image_array):
        mean_values = []
        for x in range(len(image_array)):
            mean_values.append(self.calculate_mean(image_array[x]))
        return self.calculate_mean(mean_values)

    def calculate_mean(self,list_value):
        total = 0
        for element in (list_value):
            total += element
        mean = round(total / len(list_value))
        return mean
```

- ***Find_ROI method***

Purpose: Splits the image specified by parameter into its constituent parts, used to split the image of text into individual images of letters and store them into the ROI folder of current session.

Also used to achieve noise removal, in order to remove big ink spots within the image as only ROI images that has dimension that's bigger than 40 * 40 will be saved

The following forum has given me inspiration on how to create this method:
https://cvisiondemy.com/extract-roi-from-image-with-python-and-opencv/

```
#Objective 2.e
    def find_ROI(self,image,DATADIR):
        #https://cvisiondemy.com/extract-roi-from-image-with-python-and-opencv/
        ret, thresh = cv2.threshold(image, self.thresh_value, 255, cv2.THRESH_BINARY_INV)
        kernel = np.ones((5,5),np.uint8)
        img_dilation = cv2.dilate(thresh, kernel, iterations=1)
        contours, hierachy = cv2.findContours(img_dilation.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        # sort contours
        sorted_contours = sorted(contours, key=lambda ctr: cv2.boundingRect(ctr)[0])

        for i, ctr in enumerate(sorted_contours):
            # Get bounding box
            x, y, w, h = cv2.boundingRect(ctr)

            # Getting ROI
            roi = image[y:y + h, x:x + w]
            #Drawing rectangles of white colour (invisible) onto the image
            cv2.rectangle(image, (x, y), (x + w, y + h), (255,255,255), 2)
            if w > 40 and h > 40:
                line_paper = False
                if line_paper:
                    image_array = np.asarray(roi)
                    cropped = self.crop_whole_image(image_array)
                    height,width = np.shape(cropped)
                    dimension = [width,height]
                    if np.max(dimension) > np.min(dimension) * 3:
                        print(dimension,"discard")
                    else:
                        save_dir = os.path.join(DATADIR, 'ROI' + str(i) + "." + 'png')
                        cv2.imwrite(save_dir, roi)
                else:
```

```
save_dir = os.path.join(DATADIR, 'ROI' + str(i) + "." + 'png')
cv2.imwrite(save_dir, roi)
```

- ***Noise_removal method***

Purpose: Apply opening operation[30] to the image with kernel of (3 ,3) and return resulting image

```
#Objective 2.d
    def noise_removal(self,image,kernel = 3):
        ret,thresh = cv2.threshold(image,self.thresh_value,255,cv2.THRESH_BINARY_INV)
        kernel = np.ones((kernel,kernel),np.uint8)
        opening = cv2.morphologyEx(thresh,cv2.MORPH_OPEN, kernel)
        final = opening
        return final
```

- ***Padding method***

Purpose: Used to pad blank space on the image specified by the parameter to stop parts of the letter from touching the boarder of the image, the resulting padded image is returned. The final padded dimension is specified by the 'size' parameter (final dimension should be size * size)

The following forum has given me inspiration on how to create this method:
https://jdhao.github.io/2017/11/06/resize-image-to-square-with-padding/

```
def padding(self,image,size):
        #https://jdhao.github.io/2017/11/06/resize-image-to-square-with-padding/
        WHITE = [255,255,255]
        #Pad some predetermined white space around the image before doing actual padding
        small_pad = cv2.copyMakeBorder(image, 30, 30, 30, 30, cv2.BORDER_CONSTANT,value=WHITE)
        shape = [np.shape(small_pad)[0],np.shape(small_pad)[1]]
        factor = float(size / max(shape))
        enlargement = []
```

[30] See section 1.3.4

```
        for element in shape:
            enlargement.append(int(element * factor))
        small_pad = cv2.resize(small_pad,(enlargement[1],enlargement[0]))
        width_change = (size - enlargement[1]) // 2
        height_change = (size - enlargement[0]) // 2
        full_pad = cv2.copyMakeBorder(small_pad, height_change, height_change, width_change, width_change,
cv2.BORDER_CONSTANT,value=WHITE)
        return full_pad
```

- ***Preprocess_img***

Purpose: Final pre-processing of the image, in which the image passed through parameter is padded, find its contour and drawn onto itself, remove any additional noises, cropped and resize to 28 * 28 pixels.

The image will be saved into the Final_image folder of current session and named in the order that they are processed in (if there are multiple images) using the self.count attribute.

```
def preprocess_img(self,SAVE_DATADIR,NOISE_DIR,image):
        #Padding
        padded_img = self.padding(image,300)
        found_contour = self.find_contour(padded_img) #Find contour within image
        self.find_ROI(found_contour,NOISE_DIR) #Do additional noise removal, see section 4.3.4
        NR_path = NOISE_DIR + "\ROI0.png"
        NR_image = cv2.imread(NR_path)
        pad_NR = self.padding(NR_image,300) #Pad noise removed image again
        self.image_array = pad_NR.tolist()
        cropped = self.crop_whole_image(self.image_array) #Crop image
        final_image = self.image_resize(cropped,28) #Resize image to 28*28 dimension
        #Save image to Final_image folder of current session, named in chronological order
        cv2.imwrite(os.path.join(SAVE_DATADIR,'Final_image{number}.png'.format(number = self.count)),final_image)
        self.count += 1
        os.remove(NR_path)
```

- ***Full Code***

```
import cv2
```

```
import numpy as np
import matplotlib.pyplot as plt
import os
import time

class Image_Preprocess():
    def __init__(self):
        self.DATADIR = ''
        self.img = None
        self.image_array = []
        self.thresh_value = 0
        self.count = 1
        self.image_datas = []

    def read_image(self,DATADIR):
        self.DATADIR = DATADIR
        self.img = cv2.imread(self.DATADIR)

    #Objective 2.d
    def Remove_Shadow(self, image):
        #https://stackoverflow.com/questions/44752240/how-to-remove-shadow-from-scanned-images-using-opencv
        rgb_planes = cv2.split(image)
        result_planes = []
        result_norm_planes = []
        for plane in rgb_planes:
            dilated_img = cv2.dilate(plane, np.ones((7,7), np.uint8))
            bg_img = cv2.medianBlur(dilated_img, 21)
            diff_img = 255 - cv2.absdiff(plane, bg_img)
            norm_img = cv2.normalize(diff_img,None, alpha=0, beta=255, norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_8UC1)
            result_planes.append(diff_img)
            result_norm_planes.append(norm_img)
        result = cv2.merge(result_planes)
        return cv2.merge(result_norm_planes)

    def Pre_operations(self,image):
        blurr_image = cv2.GaussianBlur(image,(5,5),cv2.BORDER_DEFAULT)
        #Objective 2.a
        grey_image = cv2.cvtColor(blurr_image,cv2.COLOR_RGB2GRAY)
        self.image_array = grey_image.tolist()
```

```
        self.thresh_value = self.calculate_thresh(self.image_array)
        #Objective 2.c
        ret, thresholded = cv2.threshold(grey_image,self.thresh_value,255,cv2.THRESH_BINARY + cv2.THRESH_OTSU)
        thresholded = cv2.bitwise_not(thresholded)
        contour,hier = cv2.findContours(thresholded,cv2.RETR_CCOMP,cv2.CHAIN_APPROX_SIMPLE)
        for cnt in contour:
            cv2.drawContours(thresholded,[cnt],0,255,24)
        thresholded = cv2.bitwise_not(thresholded)
        thresholded = self.noise_removal(thresholded,30)
        thresholded = cv2.bitwise_not(thresholded)
        self.img = thresholded
        return thresholded

    #Objective 2.g
    def crop_top_image(self,image_array_input):
        image_array = image_array_input
        empty_space_count = 0
        length = len(image_array)
        for x in range(length):#FOR EACH ARRAY WITHIN THE 2D ARRAY
            if min(image_array[x]) == 255: #IF ALL ELEMENTS = 255, ADD 1 TO EMPTY SPACE COUNT
                empty_space_count += 1
            else:
                break #IF 1 ELEMENT WAS FOUND TO NOT BE 255 THEN MOVE ONTO THE NEXT ARRAY
        for y in range(int(empty_space_count * (3/7))):#RESERVE TO PIXELS TO NOT BE DELETED
            image_array.remove(image_array[0]) #FOR THE NUMBER OF COUNTS, REMOVE THE WHITE SPACES
        return image_array

    def crop_bottom_image(self,image_array_input): #SIMILAR PRINCIPLE AS CROP_TOP_IMAGE() FUNCTION
        image_array = image_array_input
        empty_space_count = 0
        for x in range ((len(image_array)-1),-1,-1): #ITERATE IN REVERSE
            if min(image_array[x]) == 255:
                empty_space_count += 1
            else:
                break
        for y in range(int(empty_space_count * (3/7))):
            del image_array[-1]
        return image_array
```

```
    #Objective 2.g
    def crop_whole_image(self,image_array_input):
        image_array = image_array_input
        image_array = self.crop_top_image(image_array)
        image_array = self.crop_bottom_image(image_array)#CROP TOP AND BOTTOM
        rotate_image = np.rot90(image_array,3) #ROTATE BY 270 DEGREES, EASIER TO CROP LEFT/RIGHT
        image_array = rotate_image.tolist() #CONVERT NUMPY ARRAY TO LIST
        image_array = self.crop_top_image(image_array)
        image_array = self.crop_bottom_image(image_array) #CROP LEFT AND RIGHT
        fully_cropped_image = np.rot90(image_array)#ROTATE IMAGE BACK TO ORIGINAL
        return np.asarray(fully_cropped_image, dtype = np.float32)
        #CONVERT LIST BACK TO NUMPY ARRAY IN ORDER TO BE DISPLAYED BY MATPLOTLIB

    #Objective 2.f
    def find_contour(self,image):
        cv2.imwrite("tmp.png",image)
        image = cv2.imread("tmp.png")
        image = cv2.cvtColor(image,cv2.COLOR_RGB2GRAY)
        ret,thresh = cv2.threshold(image,self.thresh_value,255,cv2.THRESH_BINARY)
        contours, hierachy = cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_NONE)
        for x in range(1,len(contours)):
            cnt = contours[x]
            cv2.drawContours(thresh, [cnt], 0, (0,0,0),3)
        os.remove("tmp.png")
        return thresh

    #Objective 2.b
    def image_resize(self, image,SIZE):
        resize_img = cv2.resize(image,(SIZE,SIZE))
        ret,final_image = cv2.threshold(resize_img,self.thresh_value,255,cv2.THRESH_BINARY)
        return final_image

    #Objective 2.c
    def calculate_thresh(self,image_array):
        mean_values = []
        for x in range(len(image_array)):
            mean_values.append(self.calculate_mean(image_array[x]))
        return self.calculate_mean(mean_values)
```

```
    def calculate_mean(self,list_value):
        total = 0
        for element in (list_value):
            total += element
        mean = round(total / len(list_value))
        return mean

    #Objective 2.e
    def find_ROI(self,image,DATADIR):
        #https://cvisiondemy.com/extract-roi-from-image-with-python-and-opencv/
        ret, thresh = cv2.threshold(image, self.thresh_value, 255, cv2.THRESH_BINARY_INV)
        kernel = np.ones((5,5),np.uint8)
        img_dilation = cv2.dilate(thresh, kernel, iterations=1)
        contours, hierachy = cv2.findContours(img_dilation.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        # sort contours
        sorted_contours = sorted(contours, key=lambda ctr: cv2.boundingRect(ctr)[0])

        for i, ctr in enumerate(sorted_contours):
            # Get bounding box
            x, y, w, h = cv2.boundingRect(ctr)

            # Getting ROI
            roi = image[y:y + h, x:x + w]
            #Drawing rectangles of white colour (invisible) onto the image
            cv2.rectangle(image, (x, y), (x + w, y + h), (255,255,255), 2)
            if w > 40 and h > 40:
                line_paper = False
                if line_paper:
                    image_array = np.asarray(roi)
                    cropped = self.crop_whole_image(image_array)
                    height,width = np.shape(cropped)
                    dimension = [width,height]
                    if np.max(dimension) > np.min(dimension) * 3:
                        print(dimension,"discard")
                    else:
                        save_dir = os.path.join(DATADIR, 'ROI' + str(i) + "." + 'png')
                        cv2.imwrite(save_dir, roi)
                else:
                    save_dir = os.path.join(DATADIR, 'ROI' + str(i) + "." + 'png')
```

```
                    cv2.imwrite(save_dir, roi)

    #Objective 2.d
    def noise_removal(self,image,kernel = 3):
        ret,thresh = cv2.threshold(image,self.thresh_value,255,cv2.THRESH_BINARY_INV)
        kernel = np.ones((kernel,kernel),np.uint8)
        opening = cv2.morphologyEx(thresh,cv2.MORPH_OPEN, kernel)
        #closing = cv2.morphologyEx(thresh,cv2.MORPH_CLOSE, kernel)
        final = opening
        return final

    def padding(self,image,size):
        #https://jdhao.github.io/2017/11/06/resize-image-to-square-with-padding/
        WHITE = [255,255,255]
        small_pad = cv2.copyMakeBorder(image, 30, 30, 30, 30, cv2.BORDER_CONSTANT,value=WHITE)
        shape = [np.shape(small_pad)[0],np.shape(small_pad)[1]]
        factor = float(size / max(shape))
        enlargement = []
        for element in shape:
            enlargement.append(int(element * factor))
        small_pad = cv2.resize(small_pad,(enlargement[1],enlargement[0]))
        width_change = (size - enlargement[1]) // 2
        height_change = (size - enlargement[0]) // 2
        full_pad = cv2.copyMakeBorder(small_pad, height_change, height_change, width_change, width_change,
cv2.BORDER_CONSTANT,value=WHITE)
        return full_pad

    def preprocess_img(self,SAVE_DATADIR,NOISE_DIR,image):
        #Do padding first, find contour and after resizing, do cropping
        padded_img = self.padding(image,300)
        found_contour = self.find_contour(padded_img)
        self.find_ROI(found_contour,NOISE_DIR)
        NR_path = NOISE_DIR + "\ROI0.png"
        NR_image = cv2.imread(NR_path)
        pad_NR = self.padding(NR_image,300)
        self.image_array = pad_NR.tolist()
        cropped = self.crop_whole_image(self.image_array)
        final_image = self.image_resize(cropped,28)
        cv2.imwrite(os.path.join(SAVE_DATADIR,'Final_image{number}.png'.format(number = self.count)),final_image)
```

```
        self.count += 1
        os.remove(NR_path)
```

3.3. Utilities

- ***Stack***

```
class Stack():
    def __init__(self,size =3 ):
        self.size = size
        self.stack = []
        self.top = -1
    def alter_size(self,new_size):
        self.size = new_size
    def push(self,item):
        if len(self.stack) == self.size:
            print("Stack is full")
        else:
            self.stack.append(item)
            self.top += 1
    def peak(self):
        if self.top == -1:
            print("Stack is empty")
        else:
            print(self.stack[self.top])
    def pop(self):
        if self.top == -1:
            print("Stack is already empty")
        else:
            return self.stack[self.top]
            del self.stack[self.top]
            self.top -= 1
```

- ***Merge Sort***

```
def merge_sort(list_to_sort,front,rear):
    if front < rear:
```

```
        middle = ((front + rear) // 2)
        List_Left = merge_sort(list_to_sort,front,middle)
        List_Right = merge_sort(list_to_sort, middle + 1, rear)
        return merge(List_Left, List_Right)
    else:
        return [list_to_sort[front]]

def merge(List_Left,List_Right):
    l = []
    while len(List_Left) > 0 and len(List_Right) > 0:
        if List_Left[0] > List_Right[0]:
            l.append(List_Right[0])
            del List_Right[0]
            #remove first item from List Right
        else:
            l.append(List_Left[0])
            del List_Left[0]
            #remove first itme from List Left
    while len(List_Left) > 0:
        l.append(List_Left[0])
        del List_Left[0]
        #remove first itme from List Left
    while len(List_Right) > 0:
        l.append(List_Right[0])
        del List_Right[0]
        #remove first item from List Right
    return l
```

- ***Queue***

```
class Queue():
    def __init__(self,size = 3):
        self.size = size
        self.front = 0
        self.rear = -1
        self.queue = []
    def alter_size(self,new_size):
        self.size = new_size
```

```
    def enqueue(self,item):
        if len(self.queue) == self.size:
            print("The queue is full")
        else:
            self.queue.append(item)
            self.rear += 1
    def dequeue(self):
        if self.front > self.rear:
            print("End of queue")
        else:
            return_item = self.queue[self.front]
            self.queue[self.front] = "DELETED"
            self.front += 1
            return return_item
```

3.4. Loading_Data module

- ***Table of variables***

Identifier	Purpose
DATADIR	Data directory to the dataset used for training process
Training_data	2-D list that stores lists that contains image after pre-processing as first element and its corresponding label as the second element
IMG_SIZE	Constant that stores the standard size that all images should be in (28 * 28)
CATEGORIES	List that contains all possible labels for the alphabet classes
Category_length	Identify the amount of categories stored within the CATEGORIES list, used in one_hot function in order to create one hot encoded array (through enumeration)
Img_pps	An object of Image_Preprocess class used to do image pre-processing
files	Used to identify the name of the current file that is being processed, used to monitor the image pre-processing process

- ***Full Code***

```
import numpy as np
import matplotlib.pyplot as plt
import os
import cv2
import random
import pickle
import sys
sys.path.append('../')
from Picture_Preprocess.Image_preprocess_class import Image_Preprocess

DATADIR = "C:/Users/Jeffrey/Desktop/Dataset/Alphabet/Test Sets/Test_Set_For_Smaller_Set"
training_data = []
IMG_SIZE = 28 #Requires set dimensions for all images
              #Training data are all 28 * 28 pixels
CATEGORIES = ["A","A1","B","B1","C","C1","D","D1","E","E1","F","F1","G","G1",
              "H","H1","I","I1","J","J1","K","K1","L","L1","M","M1","N","N1",
              "O","O1","P","P1","Q","Q1","R","R1","S","S1","T","T1",
              "U","U1","V","V1","W","W1","X","X1","Y","Y1","Z","Z1"] #Labels for classes Each containing around 9000
images
category_length = len(CATEGORIES)
img_pps = Image_Preprocess()
files = []
files = [f for f in sorted(os.listdir(DATADIR))]

def create_training_data():
    counter = 0
    for category in CATEGORIES:
        path = os.path.join(DATADIR,files[counter]) #path to Handwritten alphabet images
        class_label = CATEGORIES.index(category) #Objective 4.c
        image_count = 0
        for img in os.listdir(path): #Iterate through image stored within the folder of current category/alphabet
            try:
                #Objective 4.b
                img_pps.img = cv2.imread(path + '/'+img,cv2.IMREAD_GRAYSCALE)
                image_array = np.asarray(img_pps.img)
                mean = np.mean(image_array)
                if mean < 100: #Do Thresholding when the image has black background
```

```
                    ret,thresh = cv2.threshold(img_pps.img,125,255,cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)
                    img_pps.img = thresh #All images should have white background and black foreground
                img_pps.image_array = img_pps.img.tolist()
                #Simple pre-processing of image
                final = img_pps.image_resize(img_pps.crop_whole_image(img_pps.image_array),28)
                #Resulting image after pre-processing
                training_data.append([final,class_label])
                image_count += 1
            except Exception as e:
                print("Error")
        if len(category) < 2:
            category_name = category+ " "
            file_name = files[counter]+ " "
        else:
            category_name = category
            file_name = files[counter]
        print(category_name, "completed","|    Path: ",file_name,"|    Class Label: ",str(class_label),"|    Number
of Images: ",str(image_count)) #Output relevant info to make it easier to monitor the process
        counter += 1


def one_hot(li,length): #Return the one hot encoded array of the list passed as parameter
    dim = length
    #Create a list of zeros with dimension (length of li parameter, category_length variable
    results = np.zeros((len(li),dim))
    for i, label in enumerate(li):
        results[i,label] = 1
        #Make corresponding index within the results list 1
    return results

create_training_data()
random.shuffle(training_data) #Randomizing training data

x = [] #Features
y = [] #Labels

for features, label in training_data:
    x.append(features)
    y.append(label)
```

```
x = np.array(x).reshape(-1,IMG_SIZE,IMG_SIZE, 1) #Creating tensor of shape (IMG_SIZE,IMG_SIZE) grayscale
y = one_hot(y,category_length) #Objective 4.c, Turning Labels into one hot array
pickle_out = open("x.pickle","wb") #Objective 4.d
pickle.dump(x,pickle_out) #Save the pickle file locally
pickle_out.close()

pickle_out = open("y.pickle","wb") #Objective 4.d
pickle.dump(y,pickle_out) #Save the pickle file locally
pickle_out.close()

wait = input()
```

3.5. CNN_Training module

Purpose: Creates the basic CNN structure that contains convolutional layers, max-pooling layers, fully connected layers and the final output layer. Import the pickle files produced by the Loading_Data module and use it to train the model.

Please refer to sections 1.3.1, 1.3.2 and 2.6 for the purpose of each layer and the parameters chosen

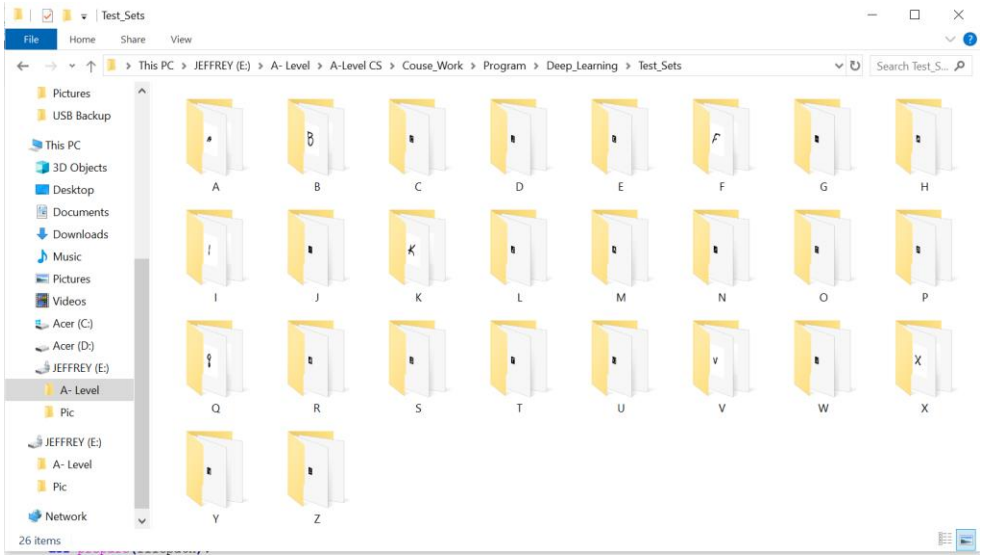
- ***Full Code***

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.callbacks import TensorBoard
from keras_tqdm import TQDMCallback
import pickle
import time

pickle_in = open("x.pickle","rb") #Objective 4.e Import processed image files
image = pickle.load(pickle_in)

pickle_in = open("y.pickle","rb") #Objective 4.e Import corresponding one-hot encoded labels
label = pickle.load(pickle_in)
```

```
print("Name:")
NAME =str(input())

model = Sequential()#Creating a Sequential model structure
#Objective 3.c, see activation parameter (RELU activation)
model.add(Conv2D(64, (5, 5), input_shape=image.shape[1:], padding = "SAME", activation = 'relu'))#Objective 3.a
model.add(Conv2D(64, (5, 5), input_shape=image.shape[1:], padding = "SAME", activation = 'relu')) #Objective 3.b
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, (5, 5), input_shape=image.shape[1:], padding = "SAME", activation = 'relu'))#Objective 3.a
model.add(Conv2D(128, (5, 5), input_shape=image.shape[1:], padding = "SAME", activation = 'relu')) #Objective 3.b
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Dropout(0.2))

model.add(Flatten()) #Objective 3.d
model.add(Dense(128, activation = 'relu')) #Dense layers are equivalent to Fully connected layers

model.add(Dense(52,activation = 'softmax')) #Objective 3.e

tensorboard = TensorBoard(log_dir="logs/{}".format(NAME)) #Using tensorboard to monitor training process

model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'],
              )

model.fit(image, label,
          batch_size=64,
          epochs= 11,
          validation_split=0.5,
          verbose = 0,
          callbacks=[tensorboard,TQDMCallback()])

model.save('Models/{}.model'.format(NAME))  #Save the trained model
```

3.6. *Model_Accuracy_Test module*

Purpose: Test accuracy of models stored within CNN_Models folder[31] by using each model to make predictions on the images stored within the Test_Sets folder (shown below, approx. 20 images per folder). The result is recorded into a text file named using the same name as the model.



- ***Full Code***

```
import cv2
import tensorflow as tf
import os
import numpy as np
import sys
sys.path.append('../')
from Picture_Preprocess.Image_preprocess_class import Image_Preprocess
```

[31] Please see section 2.2

```
CATEGORIES = ["A","A1","B","B1","C","C1","D","D1","E","E1","F","F1","G","G1",
              "H","H1","I","I1","J","J1","K","K1","L","L1","M","M1","N","N1",
              "O","O1","P","P1","Q","Q1","R","R1","S","S1","T","T1",
              "U","U1","V","V1","W","W1","X","X1","Y","Y1","Z","Z1"]
IMG_SIZE = 28
path = "Test_Sets\\" #Path to the small data set used for accuracy testing purposes
showinfo = False
img_pps = Image_Preprocess()
model_dir = "CNN_Models\\"
#Same function as the one in Loading_Data module, pre-process the image
def prepare(filepath):
    img_pps.img = cv2.imread(filepath,cv2.IMREAD_GRAYSCALE)
    image_array = np.asarray(img_pps.img)
    mean = np.mean(image_array)
    if mean < 100: #Do Thresholding when it's black background
        ret,thresh = cv2.threshold(img_pps.img,125,255,cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)
        img_pps.img = thresh
    img_pps.image_array = img_pps.img.tolist()
    cropped = img_pps.crop_whole_image(img_pps.image_array)
    final = img_pps.image_resize(cropped,IMG_SIZE)
    return final.reshape(-1, IMG_SIZE, IMG_SIZE, 1)

for model in os.listdir(model_dir): #Iterates through the trained models stored in the CNN_Models folder
    model_path = model_dir + model
    #Creates a text file that will record result of the predictions made by current model
    file = open("Results\\{}_Results.txt".format(model),"w")
    model = tf.keras.models.load_model(model_path) #Load the current model specified by model_path
    for alphabet in os.listdir(path): #Iterate through each alphabet folder stored within the Test_Sets folder
        filepath = path + alphabet
        correct = 0 #Count for number of correct predictions
        count = 0 #Count for number for images within the current alphabet folder
        for image in os.listdir(filepath): #Iterate through each image stored within current alphabet folder
            imagepath = filepath + "\\" + image

            prediction = model.predict([prepare(imagepath)]) #Make prediction
            orig_prediction = prediction.tolist()

            prediction = [round(x) for x in orig_prediction[0]]
```

```
            if showinfo:
                print(imagepath)
                print(orig_prediction)
                print("Prediction:\n", CATEGORIES[(prediction.index(1))], "\n\nDir: ", filepath, '\n ---------------
-----------------')
            if alphabet in CATEGORIES[(prediction.index(1))]:
                correct += 1 #Add one if prediction is correct
            count += 1
        writeline = "Category: " + alphabet + " Result: " + str(correct) + "/" + str(count)+ "\n"
        file.write(writeline) #Store all the information into the text file

    file.close()
```

3.7. Making_Prediction class

- ***Initialisation***

```
import cv2
import tensorflow as tf
import os
import numpy as np
import random
from random import seed
from random import randint
import sys
sys.path.append('../')
from Picture_Preprocess.Image_preprocess_class import Image_Preprocess
from Utilities.Stack import Stack
from Utilities.Linear_Queue import Queue
from Utilities.Merge_Sort import *

class Making_Prediction():
    def __init__(self):
        self.CATEGORIES = ["A","A1","B","B1","C","C1","D","D1","E","E1","F","F1","G","G1",
              "H","H1","I","I1","J","J1","K","K1","L","L1","M","M1","N","N1",
              "O","O1","P","P1","Q","Q1","R","R1","S","S1","T","T1",
              "U","U1","V","V1","W","W1","X","X1","Y","Y1","Z","Z1"]
```

```
self.IMG_SIZE = 28
self.img_pps = Image_Preprocess()
self.conf_stack = Stack()
self.Queue_operator = Queue()
self.predictions_confidence = []
self.predictions_confidence_list = []
self.model_num = 0
self.prepared = None
```

- ***Table of attributes***

Identifier	Purpose
self.CATEGORIES	List constant that stores the list of alphabet classes that the letter could be in, identical to the CATEGORIES variable in the Loading_Data module
self.IMG_SIZE	Integer constant that stores the interger value 28, which is the dimension that all images need to be resized into (28 * 28) before making predictions on it
self.img_pps	An object of Image_Preprocess class used to apply preprocess operations on images before making predictions on the image
self.conf_stack	An object of Stack class, which is a stack used to store the confidence levels of predictions made by different models, with top pointer pointing at the highest confidence
self.Queue_operator	An object of Queue class, a linear queue used to store the confidence levels of predictions made by different models
self.predictions_confidence	A 2-dimensional list with each element being a list variable that stores 2 elements, the first element is the prediction of alphabet class (string/char) made by a model and the second element is a list of confidence levels
self.predictions_confidence_list	A 2-dimensional list with each element being a list variable that stores 2 elements, the first element is the prediction of alphabet

	class (string/char) made by a model and the second element is the confidence level of that prediction
self.model_num	Integer variable that stores the number of models that are used to generate the predictions, in order to adjust the size[32] of self.conf_stack and self.Queue_operator
self.prepared	Variabel used to store the pre-operated image after applying image processing methods within Image_Preprocess class

To make it easier for the reader to distinguish the difference between self.predictions_confidence and self.predictions_confidence_list attributes, I have taken the screenshot below:

```
Predictions_confidence:
['A', [[1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0
.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]]]

['Q1', [[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]]]

['G1', [[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]]]

['E1', [[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]]]

predictions_confidence_list:  [['A', 1.0], ['Q1', 1.0], ['G1', 1.0], ['E1', 1.0]]
```

The first part shows what is stored within self.predictions_confidence attribute, where the index of where the highest confidence level is within the list of confidence levels represents the index of the alphabet within self.CATEGORIES attribute.

Second part shows what is stored within self.predictions_confidence_list attribute, where the second element (the integer value, not always 1.0) represents the highest confidence level within the list of confidence levels.

[32] Please see section 3.3

- ***Prepare method***

Purpose: Does some basic image pre-processing (grayscale, thresholding, cropping, resize) on the image file specified by the filepath parameter, make sure that all images have white background and black foreground. Returns the resulting image file. Used within gen_predictions_confidence method

```
def prepare(self,filepath):
        self.img_pps.img = cv2.imread(filepath,cv2.IMREAD_GRAYSCALE)
        image_array = np.asarray(self.img_pps.img)
        mean = np.mean(image_array)
        if mean < 100: #Do Thresholding when it's black background
            ret,thresh = cv2.threshold(self.img_pps.img,125,255,cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)
            self.img_pps.img = thresh #Apply thresholding
        self.img_pps.image_array = self.img_pps.img.tolist()
        #Convert numpy array to list
        cropped = self.img_pps.crop_whole_image(self.img_pps.image_array)
        #Crop the image
        final = self.img_pps.image_resize(cropped,self.IMG_SIZE)
        #Resize image to 28* 28 resolution
        return final.reshape(-1, self.IMG_SIZE, self.IMG_SIZE, 1)
```

- ***Make_prediction method***

Purpose: Loads the trained CNN model specified by the model_dir parameter and generates a prediction using that model.

The prediction of the alphabet class is specified by the index of which the value 1 is within the list of confidence level, which corresponds to the index of alphabet within the self.CATEGORIES attribute

Returns a list with first element being the prediction of alphabet class that the image belongs in and the second element is a list of confidence levels of that prediction

```
def make_prediction(self,model_dir):
        #Path is the path to image/images
        model = tf.keras.models.load_model(model_dir)
        prediction = model.predict([self.prepared])
```

```
        confidence = prediction.tolist()
        rounded_prediction = [round(x) for x in confidence[0]]
        alphabet_prediction = self.CATEGORIES[(rounded_prediction.index(1))]
        return [alphabet_prediction[0],confidence]
```

- ***Gen_predictions_confidence method***

Purpose: Iterates through the models stored within the CNN_Models folder, and for each model, make_prediction() method is called with the full directory of that model passed as parameter to make prediction using that model.

Self.model_num is incremented in order to count the number of models used/the number of models stored within the folder.

Finally, the return value from the make_prediction method (refer to previous sub-section) is appended into the self.predictions_confidence attribute and is returned as well.

```
def gen_predictions_confidence(self,path):
        #Path is the path to image/images
        current_path = os.path.dirname(__file__)
        model_folder_path = ("Deep_Learning\\CNN_Models")
        self.prepared = self.prepare(path)
        for model in os.listdir(model_folder_path):
            #Generate predictions using different models
            self.model_num += 1
            model_path = model_folder_path + "\\" + model #Path to each model
            print("Using ",model," model\n")
            #Objective 4.f
            prediction_confidence = self.make_prediction(model_path)
            self.predictions_confidence.append(prediction_confidence)
            #Each prediction of letter saved in a list attribute
            #Confidence of these predictions saved in another list of corresponding indices
        return self.predictions_confidence
```

- ***Create_confidence_stack method***

**Value stored within self.predictions_confidence is passed as List_of_prediction_accuracy parameter into this method from Session class*

Purpose: Each list within list_of_prediction_accuracy (where the second element is changed from a list of floating point numbers to the maximum number within that list), is appended into the self.predictions_confidence_list attribute.

a stack of confidence levels is generated with the top pointer pointing at the highest confidence.

```
def create_confidence_stack(self,list_of_prediction_accuracy):
        #Each element within list should contain the confidence/accuracy of prediction
        self.Queue_operator.alter_size(self.model_num)
        self.conf_stack.alter_size(self.model_num)
        for pair in list_of_prediction_accuracy:
            max_conf = max(pair[1][0]) #Maximum number within the list of confidence levels
            self.predictions_confidence_list.append([pair[0],max_conf])
            self.Queue_operator.enqueue(max_conf)
            #Enqueue all confidence scores
        print("predictions_confidence_list: ",self.predictions_confidence_list,"\n")
        self.Queue_operator.queue = merge_sort(self.Queue_operator.queue,0,self.Queue_operator.size - 1)
        #Sort the scores from low to high
        print("Queue: ",self.Queue_operator.queue,"\n")
        for element in self.Queue_operator.queue:
            self.conf_stack.push(self.Queue_operator.dequeue())
            #Push into stack with the top being highest confidence
```

- ***Final_prediction method***

Please refer to the Table of attributes and Create_confidence_stack method for what is stored within the self.predictions_confidence_list attribute. Purpose of this method is to generate a final prediction base on majority voting methods.

```
1. def final_prediction(self):
2.         list_len = len(self.predictions_confidence_list)
3.         letter_predictions = []
4.         prediction = "Prediction"
```

```
5.          for item in self.predictions_confidence_list:
6.              letter_predictions.append(item[0])
7.          print("Letter_predicions: ",letter_predictions)
8.          mode = self.find_mode(letter_predictions)
9.          print("Mode: ",mode)
10.         print("Stack: ",self.conf_stack.stack)
11.         if len(mode) == 1:#Majority Voting
12.             prediction = mode[0]
13.         elif len(set(self.conf_stack.stack)) == 1:
14.             #All predictions have same confidence
15.             seed(1)
16.             rnd = randint(0,len(mode) -1)
17.             list_of_mode = [letter for letter in mode]
18.             for x in range(3):
19.                 random.shuffle(list_of_mode)
20.                 #Prediction would be a random letter within the list
21.             print("Random integer: ",str(rnd))
22.             print("Shuffled List: ",list_of_mode)
23.             prediction = list_of_mode[rnd]
24.         else:
25.             greatest_confidence = self.conf_stack.pop()
26.             #Otherwise final prediction would be the one of highest confidence
27.             print("Greatest confidence: ",greatest_confidence)
28.             for pairs in self.predictions_confidence_list:
29.                 if pairs[1] == greatest_confidence:
30.                     prediction = pairs[0]
31.                     break
32.         return prediction
```

Line 3 – 6: All predictions on the image made by different models are stored within the letter_predictions variable
Line 9 – 12: Mode of the letter_prediction list is found, if there is only a single mode, then this mode is assigned to the prediction variable
Line 13 – 23: If there is more than 1 mode, and all predictions have the same confidence (same integer value within the stack), then a list of mode is created and a random mode is selected and assigned to the prediction variable
Line 24 – 31: If there is more than 1 mode and the predictions have different confidence levels, then the top of the stack is popped and the prediction that has this confidence level is assigned to the prediction variable
Line 32: The value of prediction variable is returned

- ***Find_mode method***

Purpose: Find the mode within the list that is passed as parameter. In the current case, it is used to find the mode of the list of predictions on which alphabet does the letter belong in.

```
def find_mode(self,list_of_items):
        (values,counts) = np.unique(list_of_items,return_counts = True)
        mode = values[counts == counts.max()]
        return mode
```

- ***Full Code***

```
import cv2
import tensorflow as tf
import os
import numpy as np
import random
from random import seed
from random import randint
import sys
sys.path.append('../')
from Picture_Preprocess.Image_preprocess_class import Image_Preprocess
from Utilities.Stack import Stack
from Utilities.Linear_Queue import Queue
from Utilities.Merge_Sort import *

class Making_Prediction():
    def __init__(self):
        self.CATEGORIES = ["A","A1","B","B1","C","C1","D","D1","E","E1","F","F1","G","G1",
              "H","H1","I","I1","J","J1","K","K1","L","L1","M","M1","N","N1",
              "O","O1","P","P1","Q","Q1","R","R1","S","S1","T","T1",
              "U","U1","V","V1","W","W1","X","X1","Y","Y1","Z","Z1"]
        self.IMG_SIZE = 28
        self.img_pps = Image_Preprocess()
```

```
        self.conf_stack = Stack()
        self.Queue_operator = Queue()
        self.predictions_confidence = []
        self.predictions_confidence_list = []
        self.model_num = 0
        self.prepared = None

    def prepare(self,filepath):
        self.img_pps.img = cv2.imread(filepath,cv2.IMREAD_GRAYSCALE)
        image_array = np.asarray(self.img_pps.img)
        mean = np.mean(image_array)
        if mean < 100: #Do Thresholding when it's black background
            ret,thresh = cv2.threshold(self.img_pps.img,125,255,cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)
            self.img_pps.img = thresh #Apply thresholding
        self.img_pps.image_array = self.img_pps.img.tolist()
        #Convert numpy array to list
        cropped = self.img_pps.crop_whole_image(self.img_pps.image_array)
        #Crop the image
        final = self.img_pps.image_resize(cropped,self.IMG_SIZE)
        #Resize image to 28* 28 resolution
        return final.reshape(-1, self.IMG_SIZE, self.IMG_SIZE, 1)

    def make_prediction(self,model_dir):
        #Path is the path to image/images
        model = tf.keras.models.load_model(model_dir)
        prediction = model.predict([self.prepared])
        confidence = prediction.tolist()
        rounded_prediction = [round(x) for x in confidence[0]]
        alphabet_prediction = self.CATEGORIES[(rounded_prediction.index(1))]
        #print("Prediction:\n",alphabet_prediction, "\n\nDir: ", filepath, '\n --------------------------------')
        return [alphabet_prediction[0],confidence]

    def gen_predictions_confidence(self,path):
        #Path is the path to image/images
        current_path = os.path.dirname(__file__)
        #model_folder_path = os.path.relpath('CNN_Models',current_path)
        model_folder_path = ("Deep_Learning\\CNN_Models")
        self.prepared = self.prepare(path)
        for model in os.listdir(model_folder_path):
```

```
            #Generate predictions using different models
            self.model_num += 1
            model_path = model_folder_path + "\\" + model #Path to each model
            print("Using ",model," model\n")
            #Objective 4.f
            prediction_confidence = self.make_prediction(model_path)
            self.predictions_confidence.append(prediction_confidence)
            #Each prediction of letter saved in a list attribute
            #Confidence of these predictions saved in another list of corresponding index
        return self.predictions_confidence

    def create_confidence_stack(self,list_of_prediction_accuracy):
        #Each element within list should contain the confidence/accuracy of prediction
        self.Queue_operator.alter_size(self.model_num)
        self.conf_stack.alter_size(self.model_num)
        for pair in list_of_prediction_accuracy:
            max_conf = max(pair[1][0])
            self.predictions_confidence_list.append([pair[0],max_conf])
            self.Queue_operator.enqueue(max_conf)
            #Enqueue all confidence scores
        print("predictions_confidence_list: ",self.predictions_confidence_list,"\n")
        self.Queue_operator.queue = merge_sort(self.Queue_operator.queue,0,self.Queue_operator.size - 1) #Sort the
scores from low to high
        print("Queue: ",self.Queue_operator.queue,"\n")
        for element in self.Queue_operator.queue:
            self.conf_stack.push(self.Queue_operator.dequeue())
            #Push into stack with the top being highes confidence

    def final_prediction(self):
        list_len = len(self.predictions_confidence_list)
        letter_predictions = []
        prediction = "Prediction"
        for item in self.predictions_confidence_list:
            letter_predictions.append(item[0])
        print("Letter_predicions: ",letter_predictions)
        mode = self.find_mode(letter_predictions)
        print("Mode: ",mode)
        print("Stack: ",self.conf_stack.stack)
        if len(mode) == 1:#Majority Voting
```

```
            prediction = mode[0]
        elif len(set(self.conf_stack.stack)) == 1:
            #All predictions have same confidence
            seed(1)
            rnd = randint(0,len(mode) -1)
            list_of_mode = [letter for letter in mode]
            for x in range(3):
                random.shuffle(list_of_mode)
                #Prediction would be a random letter within the list
            print("Random integer: ",str(rnd))
            print("Shuffled List: ",list_of_mode)
            prediction = list_of_mode[rnd]
        else:
            greatest_confidence = self.conf_stack.pop()
            #Otherwise final prediction would be the one of highest confidence
            print("Greatest confidence: ",greatest_confidence)
            for pairs in self.predictions_confidence_list:
                if pairs[1] == greatest_confidence:
                    prediction = pairs[0]
                    break
        return prediction

    def find_mode(self,list_of_items):
        (values,counts) = np.unique(list_of_items,return_counts = True)
        mode = values[counts == counts.max()]
        return mode
```

3.8. Sessions

I have used this forum to give me inspiration in creating the methods Determine_window_size and Position_window:
https://stackoverflow.com/questions/3352918/how-to-center-a-window-on-the-screen-in-tkinter

- ***Initialisation***

```
from GUI.User_Interface import GUI
from Picture_Preprocess.Image_preprocess_class import Image_Preprocess
```

```
from Deep_Learning.Making_Prediction_Class import Making_Prediction
from tkinter import *
import time
import datetime
import cv2
import os
import shutil
import numpy as np
import matplotlib.pyplot as plt

class Session():
    def __init__(self):
        self.image_path = ''
        self.operation_type = ''
        self.img_pps = Image_Preprocess()
        self.now = datetime.datetime.now()
        self.datapaths = {'SESSION':'Sessions\\',
                          'ROI_PATH':'ROI_Images',
                          'FINAL_PATH':'Final_Images',
                          'CURSESSION':'' #Directory for current session
                          }
        self.Firsttime = True
```

- ***Table of attributes***

Identifier	Purpose
self.image_path	String variable that stores the full directory of the imported image file
self.operation_type	String variable that stores the operation type that the user selected using the GUI
self.img_pps	An object of Image_Preprocess class used to do image pre-processing
self.now	Stores the current date/time
self.datapaths	Dictionary used to create the basic Session folder structures
self.Firsttime	Boolean variable used to indicate whether the current iteration is the first one within the same session

- ***Startup_window method***

Purpose: Object of GUI class instantiated which generates the main window that has dimension 205 * 210, positioned at the centre of the screen and not resizable.

retrieve the value of operation type and self.master.filepath and self.operation attribute of GUI class and store them into self.image_path and self.operation_type attribute of current class

```
def startup_window(self):
        root = Tk() #
        sizes = self.determine_Window_size(root)
        window_pos = self.position_window(sizes)
        root.geometry("205x210") #Decides frame dimensions
        root.geometry("+{}+{}".format(window_pos[0],window_pos[1]))
        root.resizable(False,False)
        app = GUI(window_pos,root)
        root.mainloop()
        #GUI closed as the mainloop is exited
        self.image_path = app.return_image_directory() #Filepath of image file
        self.operation_type = app.return_operation_type()
        time.sleep(1)
```

- ***Determine_window_size and Position_window methods***

Purpose: Returns list of x and y coordinate in order for the window to be positioned at the very centre of the screen.

```
def determine_Window_size(self,root):
        win_width = root.winfo_reqwidth()
        win_height = root.winfo_reqheight()
        screen_width = root.winfo_screenwidth()
        screen_height = root.winfo_screenheight()
        return_list = [[win_width,screen_width],[win_height,screen_height]]
        return return_list

def position_window(self,sizes):
```

```
        info = sizes
        x_pos = int(info[0][1]/2 - info[0][0]/2)
        y_pos = int(info[1][1]/2 - info[1][0]/2)
        return [x_pos,y_pos]
```

- ***Update_time method***

Purpose: Update the datetime value stored within the self.now attribute to the current datetime value, and update the value identified by the 'CURSESSION' key.

```
def update_time(self):
        self.now = datetime.datetime.now()
        self.datapaths['CURSESSION'] = 'Session-{now}\\'.format(now = self.now.strftime('%Y-%m-%d_%H-%M-%S'))
```

- ***Create_folder_structure method***

Purpose: If a session main folder doesn't exist, then it is created, if there are more than 10 folders then the session main folder is emptied. The session folder for current session is created, named using the format 'Session-DATE-TIME' and contains a Final_Images, a ROI_Image and a Noise_Remove[33] folder.

The directory to the ROI_Image, Final_Images folder and Noise_Remove folder is returned as a list at the end.

```
def create_folder_structure(self):
        session = self.datapaths['SESSION']
        if not os.path.exists(session):
            os.mkdir(session)
        if len([name for name in os.listdir(session)
                if os.path.isdir(os.path.join(session,name))]) > 10:
            noerror = False
            while noerror == False:
                try:
                    shutil.rmtree(session)#Removes all session if there are more than 5 sessions
```

[33] Please see section 4.3.4

```
                os.mkdir(session)
                print('Trying')
                noerror = True
            except:
                pass
        self.update_time()
        current_session = os.path.join(session,self.datapaths['CURSESSION'])
        roi_path =os.path.join(current_session,self.datapaths['ROI_PATH'])
        final_path = os.path.join(current_session,self.datapaths['FINAL_PATH'])
        noise_remove_path = os.path.join(current_session,self.datapaths['NOISE_REMOVE'])
        if not os.path.exists(current_session):
            os.mkdir(current_session)
            os.mkdir(roi_path)
            os.mkdir(final_path)
            os.mkdir(noise_remove_path)
        return [roi_path,final_path,noise_remove_path]
```

- ***Preprocess_image method***

Purpose: Only apply shadow removal for the first iteration of the current session, apply Pre_operation method (of img_pps object of Image_preprocess class) to the image.

If the operation chosen by the user is "Single", then apply preprocess_img method (of img_pps object) to the image, and save the final image to the Final_Image folder whose directory (roi_final[1]) is passed as parameter.

If the operation chose by the user is "Entire", then the image is separated into constituent images of letters and saved into the ROI folder of current session whose directory (roi_final[0]) is passed as parameter into the find_ROI method (of img_pps object). And for each image saved within the ROI folder of current session, they processed recursively using "Single" operation.

For extra information, please refer to section 2.5.9

```
def preprocess_image(self,image_path,folder_structure):
        roi_final = folder_structure #index 0 to ROI folder, index 1 to Final folder
        self.img_pps.read_image(image_path)
        if self.Firsttime:
```

```
            print("First Time")
            self.img_pps.img = self.img_pps.Remove_Shadow(self.img_pps.img)
            self.Firsttime = False
        operated_image = self.img_pps.Pre_operations(self.img_pps.img)
        if self.operation_type == 'Single':
            self.img_pps.preprocess_img(roi_final[1],roi_final[2],operated_image)
        elif self.operation_type == 'Entire':
            self.img_pps.find_ROI(operated_image,roi_final[0])
            self.operation_type = 'Single'
            for element in os.listdir(roi_final[0]):
                path_to_element = roi_final[0] + ('\\' + element)
                self.preprocess_image(path_to_element,roi_final) #Recursion
        else:
            print('An error has occured, restarting program')
```

- ***CNN_prediction method***

Purpose: For the number of images stored within the Final_Images folder, call the gen_predictions_confidence method of prediction_makers (object of Make_Prediction class), generate a confidence stack (please refer to create_confidence_stack method of Make_Prediction class) and make a final prediction. The predictions are made in ascending order of how the image within the folder is named, e.g. Final_image1 will be predicted first, then Final_image2 ….

```
def CNN_prediction(self,folder_structure):
        final_path = folder_structure[1]
        predict_list = ""
        for image_path in os.listdir(final_path):
            prediction_maker = Making_Prediction()
            final_image_path = final_path + "\\" + image_path
            alph_conf_list = prediction_maker.gen_predictions_confidence(final_image_path)
            print("Predictions_confidence: ")
            for a in alph_conf_list:
                print(a,"\n")
            prediction_maker.create_confidence_stack(alph_conf_list)
            print("Image_path: ",final_image_path)
            prediction = prediction_maker.final_prediction()
            predict_list += str((prediction[0].upper()))
            image = cv2.imread(final_image_path)
```

```
            cv2.imshow('img',image)
            cv2.waitKey(0)
        print("\nPress enter for prediction")
        enter=  input()
        os.system('cls')
        print(predict_list)
        print("\nPress enter to continue")
        enter2 = input()
        os.system('cls')
```

- ***Full Code***

```
from GUI.User_Interface import GUI
from Picture_Preprocess.Image_preprocess_class import Image_Preprocess
from Deep_Learning.Making_Prediction_Class import Making_Prediction
from tkinter import *
import time
import datetime
import cv2
import os
import shutil
import numpy as np
import matplotlib.pyplot as plt

class Session():
    def __init__(self):
        self.image_path = ''
        self.operation_type = ''
        self.img_pps = Image_Preprocess()
        self.now = datetime.datetime.now()
        self.datapaths = {'SESSION':'Sessions\\',
                          'ROI_PATH':'ROI_Images',
                          'FINAL_PATH':'Final_Images',
                          'NOISE_REMOVE': 'Noise_Remove',
                          'CURSESSION':'' #Directory for current session
                          }
```

```
        self.Firsttime = True
    def startup_window(self):
        root = Tk()
        sizes = self.determine_Window_size(root)
        window_pos = self.position_window(sizes)
        root.geometry("205x210") #Decides frame dimensions
        root.geometry("+{}+{}".format(window_pos[0],window_pos[1]))
        root.resizable(False,False)
        app = GUI(window_pos,root)
        root.mainloop()
        self.image_path = app.return_image_directory() #Filepath of image file
        self.operation_type = app.return_operation_type()
        time.sleep(1)
    def determine_Window_size(self,root):
        win_width = root.winfo_reqwidth()
        win_height = root.winfo_reqheight()
        screen_width = root.winfo_screenwidth()
        screen_height = root.winfo_screenheight()
        return_list = [[win_width,screen_width],[win_height,screen_height]]
        return return_list
    def position_window(self,sizes):
        info = sizes
        x_pos = int(info[0][1]/2 - info[0][0]/2)
        y_pos = int(info[1][1]/2 - info[1][0]/2)
        return [x_pos,y_pos]
    def update_time(self):
        self.now = datetime.datetime.now()
        self.datapaths['CURSESSION'] = 'Session-{now}\\'.format(now = self.now.strftime('%Y-%m-%d_%H-%M-%S'))
    def create_folder_structure(self):
        session = self.datapaths['SESSION']
        if not os.path.exists(session):
            os.mkdir(session)
        if len([name for name in os.listdir(session)
                if os.path.isdir(os.path.join(session,name))]) > 10:
            noerror = False
            while noerror == False:
                try:
                    shutil.rmtree(session)#Removes all session if there are more than 5 sessions
                    os.mkdir(session)
```

```
                print('Trying')
                noerror = True
            except:
                pass
        self.update_time()
        current_session = os.path.join(session,self.datapaths['CURSESSION'])
        roi_path =os.path.join(current_session,self.datapaths['ROI_PATH'])
        final_path = os.path.join(current_session,self.datapaths['FINAL_PATH'])
        noise_remove_path = os.path.join(current_session,self.datapaths['NOISE_REMOVE'])
        if not os.path.exists(current_session):
            os.mkdir(current_session)
            os.mkdir(roi_path)
            os.mkdir(final_path)
            os.mkdir(noise_remove_path)
        return [roi_path,final_path,noise_remove_path]

    def preprocess_image(self,image_path,folder_structure):
        roi_final = folder_structure #index 0 to ROI folder, index 1 to Final folder
        self.img_pps.read_image(image_path)
        if self.Firsttime:
            print("First Time")
            self.img_pps.img = self.img_pps.Remove_Shadow(self.img_pps.img)
            self.Firsttime = False
        operated_image = self.img_pps.Pre_operations(self.img_pps.img)
        if self.operation_type == 'Single':
            self.img_pps.preprocess_img(roi_final[1],roi_final[2],operated_image)
        elif self.operation_type == 'Entire':
            self.img_pps.find_ROI(operated_image,roi_final[0])
            self.operation_type = 'Single'
            for element in os.listdir(roi_final[0]):
                path_to_element = roi_final[0] + ('\\' + element)
                self.preprocess_image(path_to_element,roi_final) #Recursion
        else:
            print('An error has occured, restarting program')
            #self.operation_type = 'Error'

    def CNN_prediction(self,folder_structure):
        final_path = folder_structure[1]
        predict_list = ""
```

```
        for image_path in os.listdir(final_path):
            prediction_maker = Making_Prediction()
            final_image_path = final_path + "\\" + image_path
            alph_conf_list = prediction_maker.gen_predictions_confidence(final_image_path)
            print("Predictions_confidence: ")
            for a in alph_conf_list:
                print(a,"\n")
            prediction_maker.create_confidence_stack(alph_conf_list)
            print("Image_path: ",final_image_path)
            prediction = prediction_maker.final_prediction()
            predict_list += str((prediction[0].upper()))
            image = cv2.imread(final_image_path)
            cv2.imshow('img',image)
            cv2.waitKey(0)
        print("\nPress enter for prediction")
        enter=  input()
        os.system('cls')
        print(predict_list)
        print("\nPress enter to continue")
        enter2 = input()
        os.system('cls')
```

3.9. Main module

- ***Full Code***

Purpose: Launches the program, ensures that the program doesn't exit after producing a CNN prediction unless user clicked the exit button

```
from Session import Session
import os
import sys

print("Hello")
main = Session() #Initiate new session
main.update_time()
main.startup_window()
if not main.operation_type == '':
```

```
    folder_structure = main.create_folder_structure()
    main.preprocess_image(main.image_path,folder_structure)
    main.CNN_prediction(folder_structure)
    os.system("Main.py") #Objective 1.f
else:
    print("Exiting")
```

4. TESTING

4.1. Test plan

Tests would be split into four sub sections, which would involve testing of the GUI, the Image pre-process operations, Convolutional neural network, Session. There would be subtitles within the testing video indicating the Test number of current tests.

- 4.1.1 GUI

Test No.	Objective No.	Purpose	Test Data	Expected Outcome	Actual Outcome
1.1	1.b	Navigation	Press 'Open Image File' Button on main window	Closes the main window and display a file selecting window	SUCCESS
1.2	1.g		Press 'Instructions' Button on main window	Closes the main window and displays a new window with text instructions displayed	SUCCESS
1.3	1.h		Press 'Exit' Button on main window	Closes all existing windows	SUCCESS
1.4	1.c	To ensure that only image files can be imported	Select non-image file such as text files, executable files…	Closes file selecting window and display Error Message asking user to import file of correct file type. Redisplay main window	SUCCESS

1.5	1.d	To ensure that any type of image file can be imported	Importing an image file at file selecting window such as JPG, PNG, GIF…. files	Directory of image file saved to 'filepath' attribute and display Operation window	SUCCESS
1.6	1.e	Ensure image pre-processor is set to correct operation type	Press 'Single Letter Prediction' button on Operation window	Operation type "Single" should be saved in 'operation' attribute	SUCCESS
1.7			Press 'Entire Letter Prediction' button on Operation window	Operation type "Entire" should be saved in 'operation' attribute	SUCCESS
1.8	1.f	Ensure program returns back to main window when user accidentally closed any window	Press 'Open Image File' button then close the file selecting window	Closes file selecting window and redisplays main window	SUCCESS
1.9			Press 'Open Image File' Button and select an image file on file selecting window. Then close the Operation window	Closes Operation window and redisplays main window	SUCCESS
1.10			Press 'Instruction' button then close the Instruction window	Close Instruction window and redisplays main window	SUCCESS
1.11		Ensure program doesn't exit after producing a prediction, unless the exit button is pressed	Press 'Single Letter Prediction' button on Operation window after importing an image that contains a single handwritten letter	Output prediction onto console, then return Back to and redisplay main window	SUCCESS
1.12			Press 'Entire Letter Prediction' button on Operation window after importing an image that contains a line of handwritten text		SUCCESS
1.13	1.a	To ensure that the main window is created properly	/	Present a window with three buttons on it, "Import Image File", "Instructions" and "Exit"	SUCCESS

- 4.1.2 Image Pre-process[34]

Within this section, objectives number 2.15 and 2.16 are testing whether all objectives stated before are functioning properly as a whole, to provide a correct result.

Test No.	Objective No.	Purpose	Test Data	Expected Outcome	Actual Outcome
2.1	2.a	Ensure that imported images are converted to greyscale	Series of image files	Image file displayed should be in grey scale, image should have a colour channel of 1	SUCCESS
2.2	2.b	Ensure that images are converted to set dimensions	Series of image files	Image file should be resized to 28 * 28 resolution	SUCCESS
2.3		Finding suitable threshold value for images	Series of greyscale images, with each having a	A suitable threshold value that will be used for binary thresholding is calculated, where the value should allow needed foreground (the handwritten letter) to become black pixels and unneeded background to become white pixels	SUCCESS

[34] Explanation of terms used:
'Series of image files': Images that will be used would be shown either in the video or annotated screen shots
'Binary form image': Image with white background and black foreground

2.4	2.c	Apply binary thresholding to imported image	handwritten letter written on different coloured paper	All Image file displayed should have white background and clear legible black letter as foreground	SUCCESS
2.5	2.d	Removing noises within the image	Series of binary form image files of different handwritten letter with various noises (ink spots, empty spaces within written letters)	Ink spots and empty spaces within the image should be removed where the letter remains legible. Necessary empty spaces such as empty spaces within the letter 'O', the letter 'e' and so on shouldn't be removed	Failed, please see section 4.3.4
2.6		Removing shadows within the image	Series of binary form image file of a handwritten letter with light amount of shadow	Shadow within the imported image should be removed, where the handwritten letter still remains legible	SUCCESS
2.7	2.e	Separating image of entire text into individual images of letters	Binary form image files that contains a line of text written on plain coloured paper, wide range of coloured paper (red, green, blue, white….)	The text should be separated into individual letters and each letter will be saved as an image file into a folder, with each image file named in the order that they are processed in	SUCCESS
2.8	2.f	Drawing contour lines onto an image of a padded single handwritten letter	A binary form padded image file that contains a handwritten letter	Contour lines within the image are found, thickened and drawn on top of original letter.	SUCCESS
			A binary form padded image after applying shadow removal		
2.9		Resizing image of a single handwritten letter with contour lines drawn	Image files with contour lines drawn and noise removed already	After resizing to 28*28 resolution, drawn contour lines should be thick enough to ensure that letter is still legible	SUCCESS but please see section 4.3.2
2.10		Cropping top areas of the image		Resulting image should have some unnecessary white background on top of the letter removed, with some white background reserved	SUCCESS

2.11	2.g	Cropping bottom areas of the image	Series of binary form image files that contains a handwritten letter with contour lines drawn	Resulting image should have some unnecessary white background at the bottom of the letter removed, with some white background reserved	SUCCESS
2.12		Cropping image to keep necessary areas only		The resulting image should have some unnecessary white background around the letter removed, with some white background reserved	SUCCESS
2.13	/	Ensure that padding of white spaces around the image of a handwritten letter is done	Series of binary form image files that contains a handwritten letter	The resulting image should be padded so that it has dimension of 300 * 300 pixels, constituent parts of the letter shouldn't be touching the boarder of the image	SUCCESS
2.14	/	Ensure that the Pre_operations method is functioning properly	Series of images with either a single handwritten letter or an entire text written on a plain sheet of paper	Resulting image should be in binary form with the letter within the image being thickened	SUCCESS
2.15	/	Ensure that the preprocess_img method is functioning properly	Series of images with a single handwritten letter written on a plain sheet of paper that has Pre-operations[35] applied	An image of 28 * 28 pixels with white background and black foreground should be written to the Final_Images folder of current Session folder and named in the format 'Final_image{}.png' with the value within {} being the current iteration number. Letter within the image should be clearly legible by human	SUCCESS

[35] See section 4.1.2 test number 2.14

- 4.1.3 CNN

Test No.	Objective No.	Purpose	Test Data	Expected Outcome	Actual Outcome	Evidence
3.1	3.a	Creating CNN Structure	Log file of training loop	The CNN model should have some convolutional layers	SUCCESS	Video
3.2	3.b			The CNN model should some max pooling layers	SUCCESS	
3.3	3.c			The CNN model should use RELU as activation function for all layers (except for final layer)	SUCCESS	
3.4	3.d			Output at the end of convolutional and max pooling layer stacks should be flattened into one dimension	SUCCESS	
3.5	3.e			The CNN model should have some fully connected layers with the final fully connected layer having softmax as activation function	SUCCESS	
3.6	4.a	Creating a dataset of handwritten letter images	/	The dataset should be split into 52 folders with 26 uppercase letter folders and 26 lowercase letter folders, each containing 5000 images	SUCCESS	See section 4.2.1.1
3.7	4.b	Pre-processing of images used as training data	Small dataset of images of handwritten letters	All training data fed into the model should have white background, black foreground, cropped and resized to 28*28 resolution	SUCCESS	Video
3.8	4.c	Assigning labels to each training data	A list of 52 elements containing the upper and lowercase alphabet characters	Labels assigned to each training data should be the index of the corresponding letter within the list	SUCCESS	
3.9		Converting integer labels into one hot encoded array	List that contains pairs of training data and label	Each label should be converted into one hot array, with the original integer value being the index of integer value 1 within the array and other elements should be integer value 0	SUCCESS	

3.10	4.d	Saving training data	List of training data and list of labels	Both lists should be written to separate pickle files, exported and saved in locally	SUCCESS	See section 4.2.1.2
3.11	4.e	Optimizing training	Pickle files of labels and training data	The training process shouldn't overfit nor underfit the model, meaning the graph of validation loss score shouldn't be rising at the end of training nor dropping dramatically	SUCCESS	Video
3.12	4.f	Making Prediction of alphabet class	A trained CNN model and an imported image file that contains a single handwritten letter	Should return a list of prediction of alphabet class that the letter within the image belong in along with the confidence level of the prediction	SUCCESS	
3.13		Making prediction using several trained CNN models	Folder of trained CNN models and imported image files that contains a single handwritten letter	Should return 2-Dimensional list with each element being a list of alphabet prediction produced by each trained model within the model folder and its confidence level	SUCCESS	
3.14	/	Creating stack of confidence levels of predictions	2-Dimensional list with each element being a list of alphabet prediction and its confidence	A stack of confidence levels should be created with the top being highest confidence and the lowest being lowest confidence	SUCCESS	
3.15	/	Producing final prediction base on		The mode of alphabet predictions within the 2D list should be found	SUCCESS	
3.16				If there is only one mode, and this mode would be the final prediction	SUCCESS	

3.17		majority voting and level of confidence	Stack of confidence levels of predictions and a 2-Dimensional list with each element being a list of alphabet prediction and its confidence	If there are more than one mode and all confidence levels within the stack are the same, then the modes should be shuffled and one of the modes would be randomly selected to be the final prediction	SUCCESS	Video
3.18				If there are than one mode and the confidence levels within the stack are different, then the top of the stack is popped and the alphabet prediction who has this confidence level become the final prediction	SUCCESS	
3.19	/	Testing of accuracies of models	Models of trained CNN, a small dataset of images of handwritten letters	Accuracy of the predictions made by each model should be around 80%	Failed, see section 4.3.5	

- 4.1.4 Session

There are no objectives associated with this class, as the purpose of Session class is to act as an interface for the communication of other classes such as GUI class, Image Pre-process class and so on. There would be some test that are seemingly the same, but in fact are testing different parts of the program. Please refer to section 2.2 if unsure about some of the terms used.

Test No.	Purpose	Test Data	Expected Outcome	Actual Outcome
4.1	Ensure that the coordinates required to position main window at the centre is calculated	/	List that contains the coordinates that allows main window to be positioned at the centre	SUCCESS
4.2	Ensure main window creation methods within GUI class can be called	List of coordinates retrieved from previous objective	Main window is created and positioned at the centre of the screen; user shouldn't be able to change dimension of any window	SUCCESS

4.3	Ensure image directory is correctly extracted and passed	Importing an image using the GUI[36]	Path to the image is correctly passed from GUI class and saved within image_path attribute	SUCCESS
4.4	Ensure operation type selected by the user is correctly passed from the GUI class	Press 'Single Letter Prediction' button on Operation window	Operation type of either "Single" or "Entire" correctly passed from GUI class and saved within operation_type attribute	SUCCESS
		Press 'Entire Letter Prediction' button on Operation window		
4.5	Creating folder structure	A folder without the Session main folder	Only a single Session main folder should exist	SUCCESS
4.6		A folder that contains the Session main folder		SUCCESS
4.7	Ensure that the Session main folder doesn't store more than 10 Session sub-folders	Session main folder that contains more than 10 Session sub-folders	If there are more than 10 sub folders, then the sub-folders should be deleted before creating a new sub-folder, otherwise create a new sub-folder as usual	SUCCESS
4.8	Ensure Session sub folder for a new session is created correctly	/	All Session sub folder should be created whenever a new session is initiated and should be named in the format and should contain a Final_Images folder and a ROI_Images folder. The directory of the two folders should be returned as a list	SUCCESS
4.9	Shadow removal for first iteration only	Series of imported image files that contains either a single handwritten letter or entire text with light amount of shadow	Shadow removal should only be applied to the image file once at the start of every session[37], further iterations within the same session (e.g. when pre-processing an entire	SUCCESS

[36] See section 4.1.1 test number 1.1 and 1.5

[37] See section 2.2

			text) shouldn't have shadow removal applied	
4.10	Pre-Process[38] image using Image_Preprocess class depending on operation type	Operation type 'S' with an Image that contains single handwritten letter imported and with shadow removed	Image should be pre-processed and resulting image stored in Final_Images folder of current Session sub folder	SUCCESS
4.11		Operation type 'E' with an Image that contains handwritten text imported and with shadow removed	Image should be separated into images of individual letters within the text and stored in ROI_Images folder of current Session sub folder and each would be pre-processed and stored in Final_Images folder	SUCCESS
4.12	Making CNN prediction	Pre-processed image/images that are stored within Final_Images folder of current Session sub folder	For the number of images stored within Final_Images folder, output predictions of the alphabet class that the letter images belong in onto console	SUCCESS

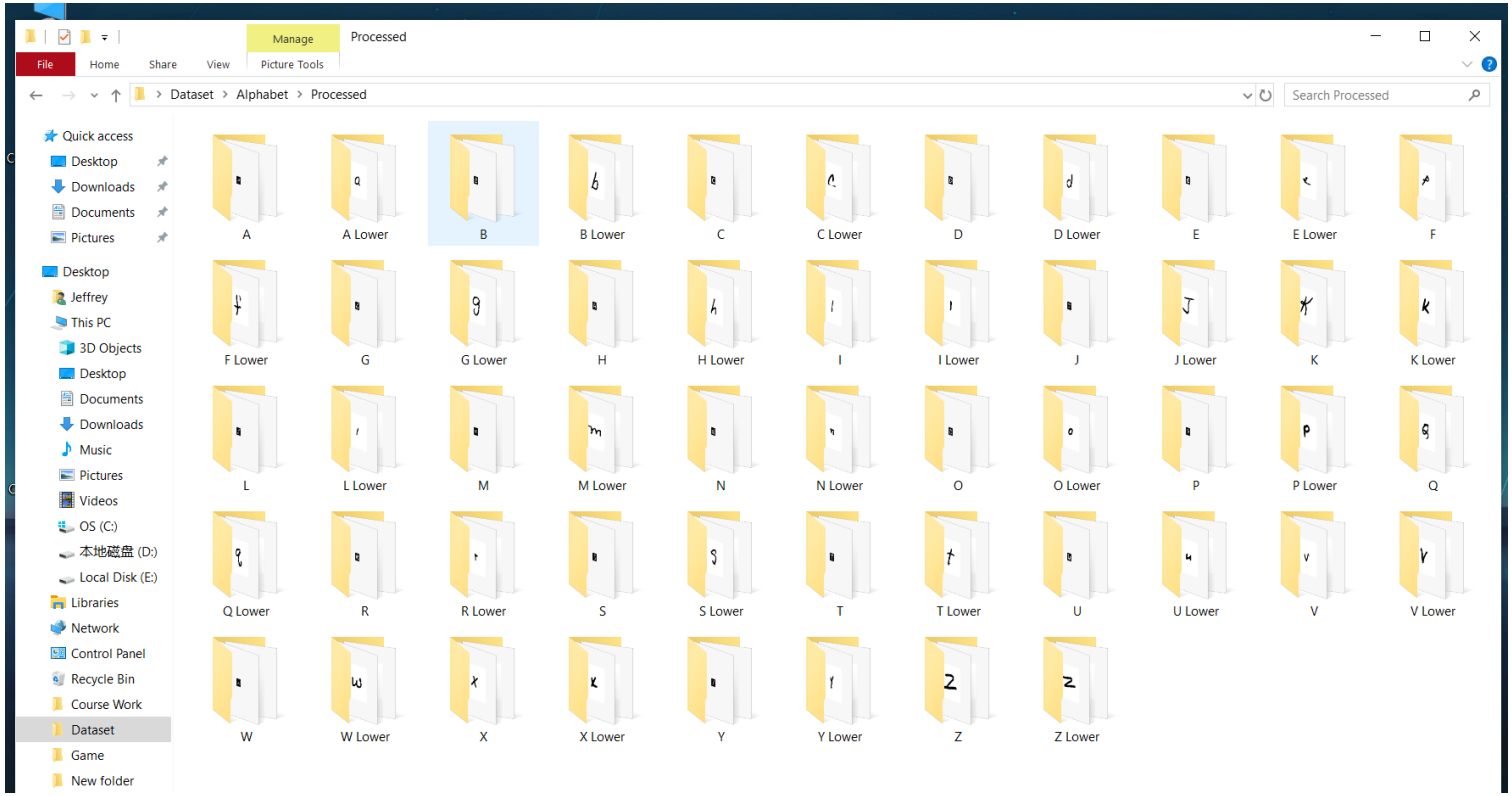
4.2. Testing Evidence

- 4.2.1 Screenshot Evidence

4.2.1.1 Dataset of Training data

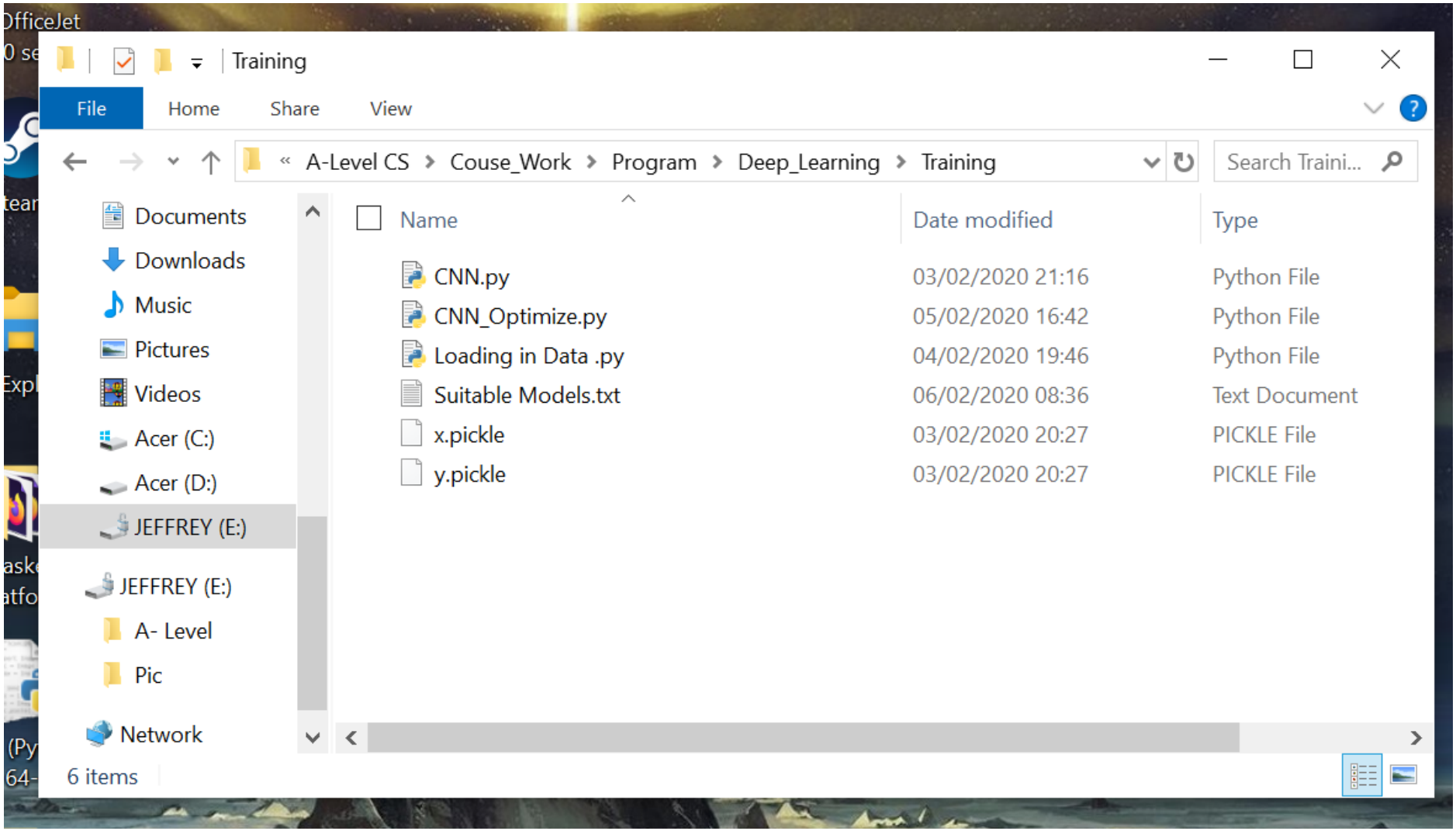
Image of the dataset used:

[38] See section 4.1.2 test number 2.16



Dataset created manually by selecting appropriate images from the NIST and NMIST datasets, with 5000 images in each alphabet folder, total of 52 folders

4.2.1.2 Saved Pickle files



x.pickle saves all the image files, y.pickle stores all the corresponding one hot encoded labels.

- 4.2.2 Video Evidence

GUI	Image Pre-process	CNN	Session
https://youtu.be/sWWID_K-09U	https://youtu.be/DgUkns8Q2lI	https://youtu.be/yBb8MxBRMEg	https://youtu.be/TH40jesnC34

4.3. Errors and solution

Sections 4.3.1 to 4.3.3 are the errors that I have already fixed before the time of recording the testing videos. Remaining sections are errors that I have encountered during testing.

- 4.3.1 Holes within letters (Objective No. 2.d)

The error that I have encountered is as shown below:

The images show the ROI images of letters separated from an original text of 'ABC', there are many holes within the image although the original image contains perfect written letters of 'ABC'. After some experiments and analysis, I have found out that this error is cause by applying shadow removal method twice to each individual letter.

The algorithm that I have developed back then also uses recursion to achieve image pre-processing as explained in section 2.5.9, which led to the shadow removal method being applied to the original text image once, and applied again to each letter while processing them individually.

As a result of that, some parts of the letter become extremely faint, hence while doing binary thresholding after converting the image into grey scale, the faint areas would be recognized as unwanted area, and therefore converted to white, which created those white spots.

This error has been fixed by adding a restriction to the Session class, which only allows shadow removal to be applied one time only for each session:

```
if self.Firsttime:
        print("First Time Processing, doing shadow removal")
        self.img_pps.img = self.img_pps.Remove_Shadow(self.img_pps.img)
        self.Firsttime = False
```

Please refer to section 3.7 Preprocess_image method for extra detail

And also, contour lines are drawn onto the text straight after applying shadow removal it in order to thicken the letters within the text and make them less faint. Noise removal is also applied to the original text after drawing contour lines onto the shadow removed text, before it being split up into individual letters through ROI.

- ## 4.3.2 Drawing contour lines (Objective No. 2.f)

The same text is used for image pre-processing as the previous section, and the image above shows the final 28 * 28 pixels image after all the pre-processing, of which to me seems pretty difficult to recognize.

This error is cause by drawing contour lines onto the letter with some parts of the letter touching the boundaries of the image, which is cause by cropping the image with no reserved white spaces around the letter.

This error has been fixed by using padding of white spaces around the letter before drawing contour lines. All images would first be lightly padded (e.g. pad 30 pixels on top/bottom/left/right of the image), and then they are padded to 300 * 300 pixels in size. This ensures that the letter does not touch the boundary of the image.

```
def padding(self,image,size):
        #https://jdhao.github.io/2017/11/06/resize-image-to-square-with-padding/
        WHITE = [255,255,255]
        #Pad some predetermined white space around the image before doing actual padding
        small_pad = cv2.copyMakeBorder(image, 30, 30, 30, 30, cv2.BORDER_CONSTANT,value=WHITE)
        shape = [np.shape(small_pad)[0],np.shape(small_pad)[1]]
        factor = float(size / max(shape))
        enlargement = []
        for element in shape:
            enlargement.append(int(element * factor))
        small_pad = cv2.resize(small_pad,(enlargement[1],enlargement[0]))
        width_change = (size - enlargement[1]) // 2
        height_change = (size - enlargement[0]) // 2
        full_pad = cv2.copyMakeBorder(small_pad, height_change, height_change, width_change,
width_change, cv2.BORDER_CONSTANT,value=WHITE)
        return full_pad
```

Please refer to section 3.2 Padding method for extra detail

In addition, I have changed my cropping algorithm so that it reserves ¼ of empty pixel arrays for each orientation (top, bottom, left right) around the letter:

```
def crop_top_image(self,image_array_input):
        image_array = image_array_input
        empty_space_count = 0
```

```
        length = len(image_array)
        for x in range(length):#FOR EACH ARRAY WITHIN THE 2D ARRAY
            if min(image_array[x]) == 255: #IF ALL ELEMENTS = 255, ADD 1 TO EMPTY SPACE COUNT
                empty_space_count += 1
            else:
                break #IF 1 ELEMENT WAS FOUND TO NOT BE 255 THEN MOVE ONTO THE NEXT ARRAY
        print("Top: ",empty_space_count)
        for y in range(int(empty_space_count * (3/4))):#RESERVE TO PIXELS TO NOT BE DELETED
            image_array.remove(image_array[0]) #FOR THE NUMBER OF COUNTS, REMOVE THE WHITE SPACES
        return image_array
```

Please refer to section 3.2 Crop_top_image and crop_bottom_image methods for extra detail

- 4.3.3 CNN prediction error (Objective No. 4.f)

IMAGE OF TERRIBLE CNN RESULTS:

```
Test_Sets\D\hsf_0_00000.png
(128, 128)
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Prediction:
 W

Dir:  Test_Sets\D
 --------------------------------
Test_Sets\D\hsf_0_00001.png
(128, 128)
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Prediction:
 W

Dir:  Test_Sets\D
 --------------------------------
Test_Sets\D\hsf_0_00002.png
(128, 128)
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Prediction:
 W

Dir:  Test_Sets\D
 --------------------------------
Test_Sets\D\hsf_0_00003.png
(128, 128)
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Prediction:
 W

Dir:  Test_Sets\D
```

```
Test_Sets\H\H-4.png
(28, 28)
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Prediction:
 W

Dir:  Test_Sets\H
 --------------------------------
Test_Sets\H\H-5.png
(28, 28)
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Prediction:
 W

Dir:  Test_Sets\H
 --------------------------------
Test_Sets\H\H-6.png
(28, 28)
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Prediction:
 W

Dir:  Test Sets\H
```

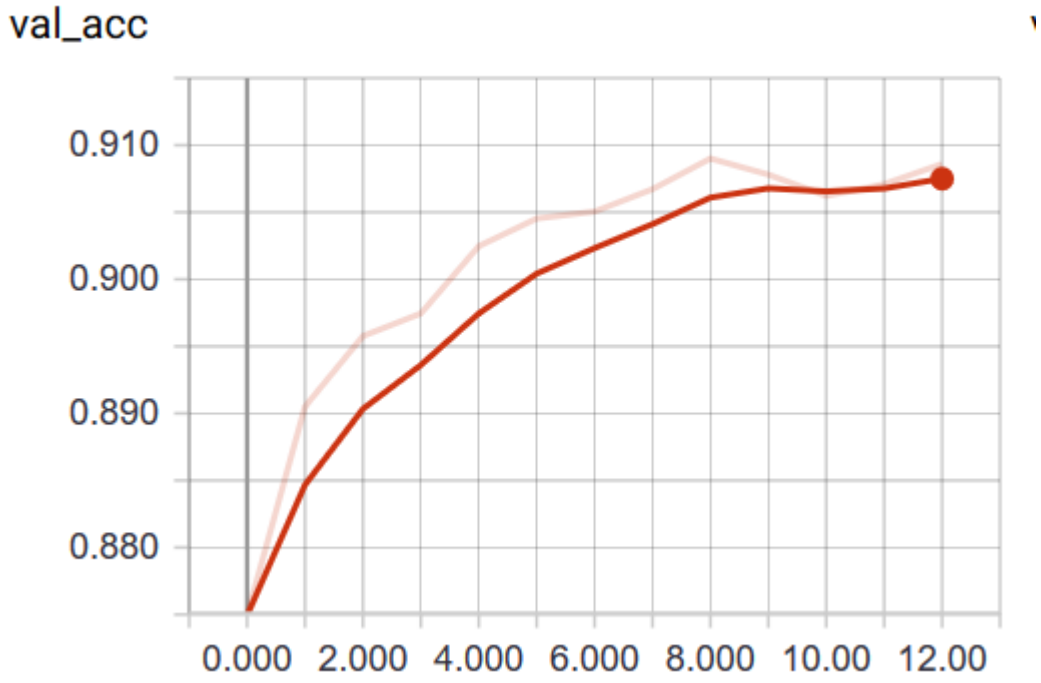
Results.txt - Notepad

File Edit Format View Help

```
Category: A Result: 0/22
Category: B Result: 0/7
Category: C Result: 0/21
Category: D Result: 0/35
Category: E Result: 0/20
Category: F Result: 0/10
Category: G Result: 0/25
Category: H Result: 0/20
Category: I Result: 0/25
Category: J Result: 0/20
Category: K Result: 0/20
Category: L Result: 0/20
Category: M Result: 0/20
Category: N Result: 0/20
Category: O Result: 0/20
Category: P Result: 0/20
Category: Q Result: 0/20
Category: R Result: 0/20
Category: S Result: 0/20
Category: T Result: 0/20
Category: U Result: 0/20
Category: V Result: 0/20
Category: W Result: 20/20
Category: X Result: 0/20
Category: Y Result: 0/16
Category: Z Result: 0/19
```

Ln 1, Col 1 | 100% | Windows (CRLF) | UTF-8

IMAGE OF TENSORBOARD ANALYSIS:



As shown in the image shown above, the CNN model that I have trained in the early stages of development were terrible at recognizing the test dataset, and has a tendency of recognizing everything as “W”.

However, this contradicts with the high validating accuracy score (which is a good thing) that I have obtained with Tensor Board. Bizarre situation!

Thankfully, after dozens of days of trial and error, I have figured out that this error is caused by a VERY silly mistake while creating the list of alphabet class labels. I have forgotten to add some of the letters within the alphabet into the list, which messed up the entire model.

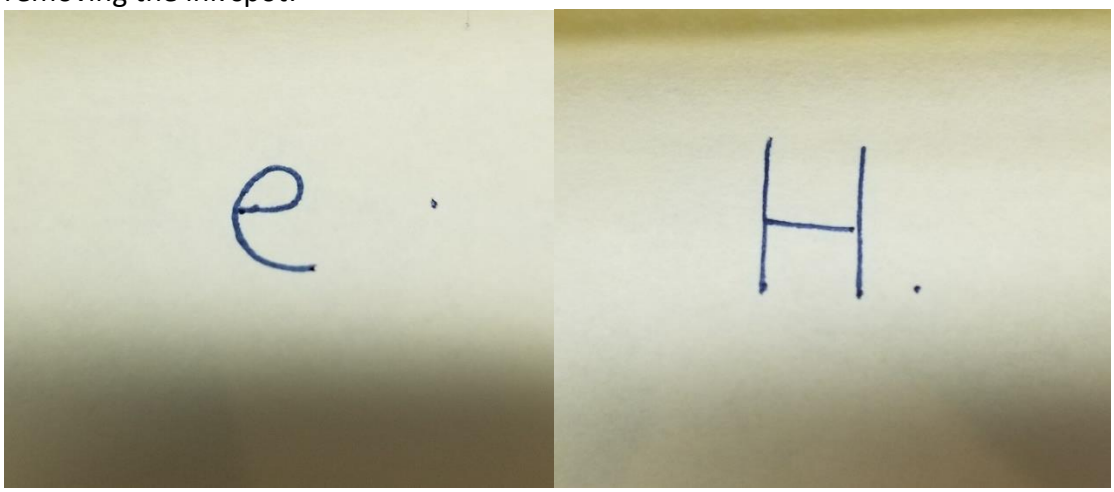
In addition, another mistake that I have made is that I did not use the same image pre-processing methods for the training process and the prediction process, where for the training process I have used a more simplified version, which would lead to the model being unable to recognized specific edges, as they seem different in its point of view.

As a result, I have fixed this error by applying the same image pre-processing methods for both the training and the prediction process, and adapted the original list of alphabet class labels so that it contains the whole alphabet.

Detail of the code can be found in section 3.4 and 3.5

- ## 4.3.4 Noise removal of relatively large ink spots (Test No. 2.9, Objective No. 2.f)

As shown in the testing video, with images shown below, the program has been unsuccessful in removing the ink spot:



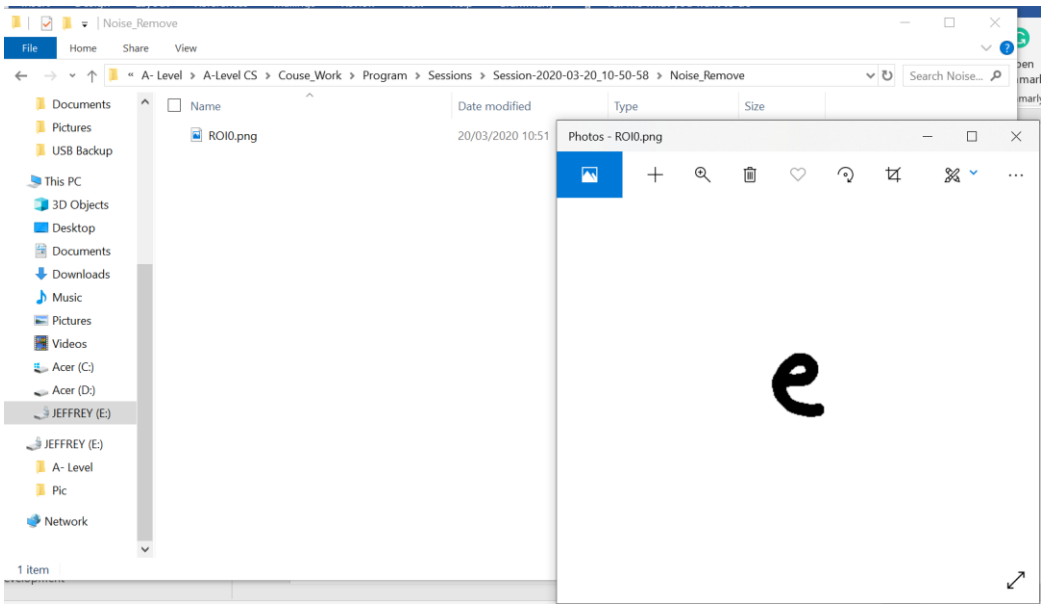
This error has been fixed using ROI within the preprocess_img method along with the original noise removal method of Image_preprocess class as shown:

```
def preprocess_img(self,SAVE_DATADIR,NOISE_DIR,image):
        #Do padding first, find contour and after resizing, do cropping
        padded_img = self.padding(image,300)
        found_contour = self.find_contour(padded_img)
        self.find_ROI(found_contour,NOISE_DIR)
        NR_path = NOISE_DIR + "\ROI0.png"
        NR_image = cv2.imread(NR_path)
        pad_NR = self.padding(NR_image,300)
        self.image_array = pad_NR.tolist()
        cropped = self.crop_whole_image(self.image_array)
        final_image = self.image_resize(cropped,28)
            cv2.imwrite(os.path.join(SAVE_DATADIR,'Final_image{number}.png'.format(number = self.count)),final_image)
        self.count += 1
        os.remove(NR_path)
```

A new folder called Noise_Remove folder is created within the current session, and the directory of that folder is passed into this method along with other parameters that remains unchanged.

ROI of the image that only contains a single letter is found, and if there is a relatively big ink spot within the image, it would be identified as a region of interest and removed (or rather not saved) and the actual letter would be saved into the Noise_Remove folder, and this image would be processed just like before.

As you can see below, the image stored within the Noise_Removed folder is the letter with the ink spot removed, reason why it looks more plump than the original image is because I have drawn thickened contour lines onto the letter already.



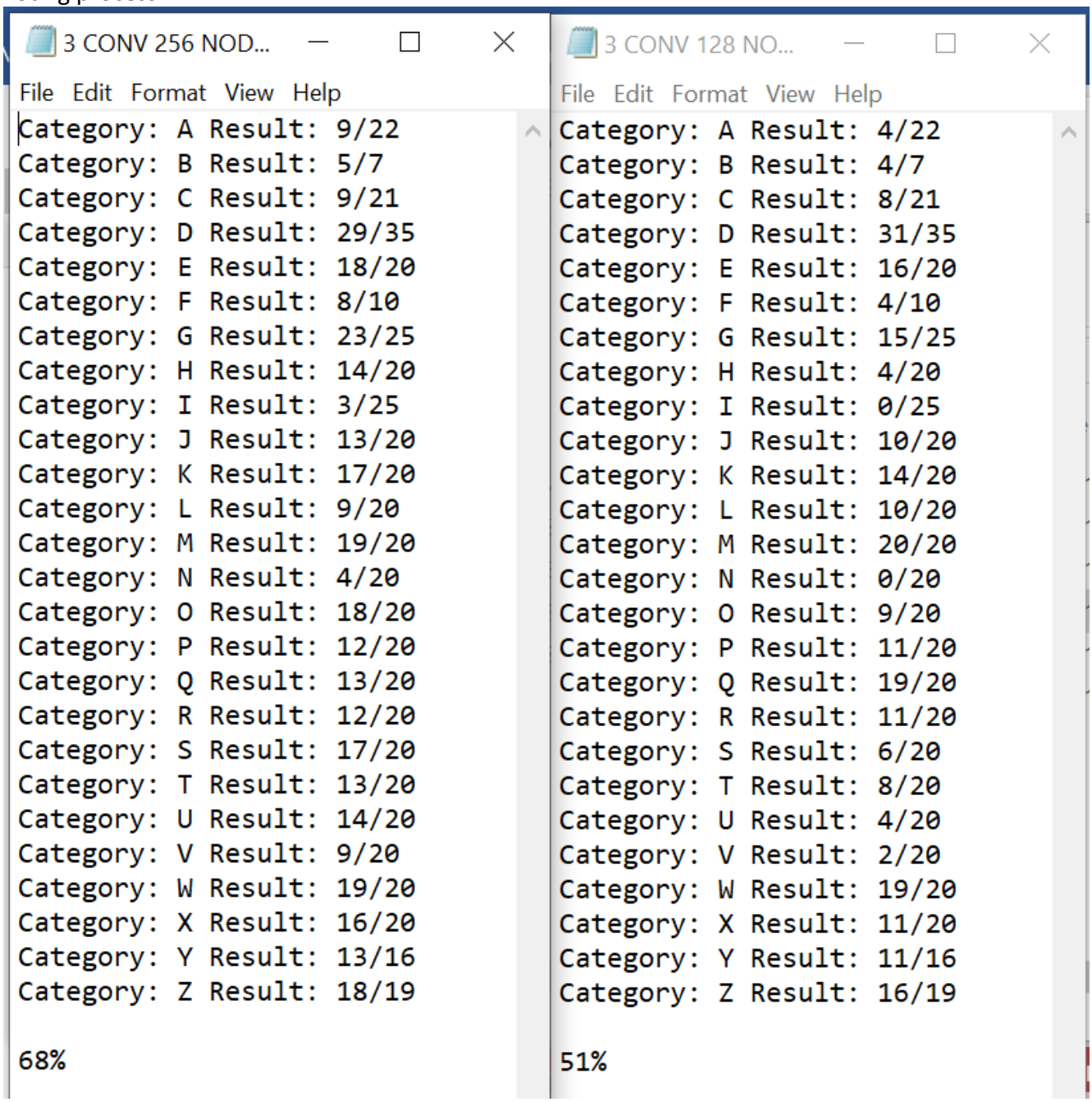
The highlighted part within the code shown below is the selection statement used to filter out the actual letter and ink spots that are relatively big:

```
def find_ROI(self,image,DATADIR):
        ret, thresh = cv2.threshold(image, self.thresh_value, 255, cv2.THRESH_BINARY_INV)
        kernel = np.ones((5,5),np.uint8)
        img_dilation = cv2.dilate(thresh, kernel, iterations=1)
        contours, hierachy = cv2.findContours(img_dilation.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
        # sort contours
        sorted_contours = sorted(contours, key=lambda ctr: cv2.boundingRect(ctr)[0])

        for i, ctr in enumerate(sorted_contours):
            # Get bounding box
            x, y, w, h = cv2.boundingRect(ctr)

            # Getting ROI
            roi = image[y:y + h, x:x + w]
            #Drawing rectangles of white colour (invisible) onto the image
            cv2.rectangle(image, (x, y), (x + w, y + h), (255,255,255), 2)
            if w > 40 and h > 40:
                line_paper = False
                if line_paper:
                    image_array = np.asarray(roi)
                    cropped = self.crop_whole_image(image_array)
                    height,width = np.shape(cropped)
                    dimension = [width,height]
                    if np.max(dimension) > np.min(dimension) * 3:
                        print(dimension,"discard")
                    else:
                        save_dir = os.path.join(DATADIR, 'ROI' + str(i) + "." + 'png')
                        cv2.imwrite(save_dir, roi)
                else:
                    save_dir = os.path.join(DATADIR, 'ROI' + str(i) + "." + 'png')
                    cv2.imwrite(save_dir, roi)
```

This new method is also compatible with processing of entire text as the text is first separated into images of individual letters and then the preprocess_img method is called upon those images one by one.

- 4.3.5 Low accuracy CNN models

Despite having good validation accuracy and validation loss scores as shown in section 2.6.4, the following models has a low and unacceptable accuracy score shown in the video and the images below, and they have been taken out of the CNN_Models folder, and won't be used for the majority voting process:



3 CONV 256 NODE 1 DENSE model

3 CONV 128 NODE 1 DENSE model

5. EVALUATION

5.1. *My evaluation of project performance against specific objectives*

Green colour represents achieved, for details on what each objective represents, please refer to section 1.4.2

Objective	Achieved	Evidence and comment
1.a Have a main window that contains 3 buttons: "Open image file" "Instructions" "Exit"		**Test No. 1.13**, user can do this by launching the program, and the main window would be displayed. **REFER TO init_window method of GUI class**
1.b User clicks the import image file button to import image of hand-written text		**Test No. 1.1,** user can do this by clicking the 'Open Image File' button, and file selecting window is displayed (whilst hiding main window) to import a file **REFER TO Open_Image method of GUI class**
1.c Error detection method		**Test No. 1.4,** error message is displayed when user imported wrong file type and returns back to the main window without exiting the program **REFER TO Open_Image method of GUI class**
1.d After importing the image, Operation window is displayed with more options selected using buttons		**Test No. 1.5,** after user imported an image file successfully, Operation window is displayed. **REFER TO Operation_option method of GUI class**
1.e Specific operation type should be passed into the image pre-processor		**Test No. 1.6, and 1.7,** user can do this by selecting the operation type that they want, and it would be saved and passed into the image pre-processor **REFER TO Single_letter and Entire_letter method of GUI class**
1.f Main window should be redisplayed whenever an error occurs or after making a prediction		**Test No. 1.8 to 1.12,** main window would be redisplayed without the program exiting when user accidentally closed any window, imported file of wrong datatype of after producing a prediction **REFER TO Format_window method of GUI class and Main module**
1.g Instruction window is displayed when the		**Test No. 1.2,** user can do this by clicking the "Instructions" button on the main window

Instruction button is pressed		**REFER TO Display_Instruction method of GUI class**
1.h Exit button when pressed, should terminate program completely		**Test No. 1.3,** user can do this by clicking the "Exit" button on the main window. **REFER TO Quitting method of GUI class**
2.a Convert input image from RGB to grayscale		**Test No. 2.1,** imported image is converted to grayscale with colour channel of 1 **REFER TO Pre_operations method of Image_preprocess class**
2.b Scale image to 28 * 28 resolution		**Test No. 2.2,** imported image is resized to 28 * 28 resolution at the very end of pre-processing, letter remains legible after this operation **REFER TO Image_resize method of Image_preprocess class**
2.c Apply binary thresholding		**Test No. 2.3 and 2.4,** threshold value of the image is calculated and binary thresholding is applied, resulting in image of white background and black foreground **REFER TO Calulate_thres and calculate_mean methods and Pre_operations method of Image_preprocess class**
2.d Apply noise removal through morphological transformation	Error Fixed, see section 4.3.4	**Test No. 2.5, and 2.6,** **REFER TO Noise_removal and Preprocess_img method of Image_preprocess class**
2.e Segment letters out from text		**Test No. 2.7,** image of text is separated into individual images of letters, each of them is named in ascending order and saved within ROI folder of current session **REFER TO Find_ROI method of Image_preprocess class**
2.f Finding contour lines within the image of letter		**Test No. 2.8 and 2.9,** contours within the image is found **REFER TO Find_contour method of Image_preprocess class**
2.g Cropping image		**Test No. 2.10 to 2.12,** unnecessary white space around letter is cropped out, while reserving some white space around it. **REFER TO Crop_top_image and crop_bottom_image methods and Crop_whole_image method of Image_preprocess class**
3.a The CNN model needs to have convolutional layer		**Test No. 3.1,** there are multiple models trained that uses different number of convolutional layers with different parameters. **REFER TO CNN_Training module**

3.b The CNN model needs to have max pooling layer		**Test No. 3.2,** there are multiple models trained that uses different number of max pooling layers **REFER TO CNN_Training module**
3.c The CNN model needs to have RELU as activation except for the final layer		**Test No. 3.3,** RELU activation is used except for the last layer **REFER TO CNN_Training module**
3.d The CNN model needs to have the output tensor flattened		**Test No. 3.4,** output tensor from the stack of convolutional layers and max pooling layers is flattened before being fed into the fully connected layers **REFER TO CNN_Training module**
3.e The CNN model needs to have fully connected layers with sigmoid activation used for the final layer		**Test No. 3.5,** different number of fully connected layers used for the models, last fully connected layer (output layer) has sigmoid as activation **REFER TO CNN_Training module**
4.a A big dataset of handwritten letter images is created		**Test No. 3.6,** dataset contains 52 folders each containing a letter within the alphabet (Upper case or lower case) **REFER TO Section 4.3.1.1**
4.b All images within the training dataset are pre-processed before being fed into the CNN model		**Test No. 3.7,** the images are pre-processed using methods within the Image_Preprocess class **REFER TO Loading_Data module**
4.c Labels are attached to each image and converted to one hot encoded array		**Test No. 3.8 and 3.9,** labels generated base on the index of the alphabet within the list that stores all alphabets, all are converted to on hot encoded array **REFER TO Loading_Data module**
4.d Processed images and corresponding labels are saved within a pickle file		**Test No. 3.10,** pickle file saved locally and are name x (image files) and y (labels) **REFER TO Loading_Data module**
4.e The saved pickle files are imported and fed into the CNN for training		**Test No. 3.11** **REFER TO CNN_Training module**
4.f Trained model is imported and used to create predictions on images of handwritten letter	Error fixed, see section 4.3.5	**Test No. 3.12 and 3.13,** models all have an accuracy of around 80% for getting the right prediction **REFER TO Model_Accuracy_Test module**

5.2. Independent feedback

Ms Biletchi, who is the head of Computer Science department of Epsom college has kindly accepted to give me feedback on my investigation project.

After having an interview where I was able to demonstrate my program using a variety of images, Ms Biletchi has made the following comments which I have summarized below, which can be split into 4 sections:

1. The GUI has been created correctly when the program is launched, and is very user-friendly due to the lack of complicated operations and the user instructions window. All buttons led to the correct operation when clicked. Program doesn't exit when an error occurs and returns back to main menu as specified within the specific objective.

2. Image pre-processing is working excellently, where the image is accurate read with shadow and noises removed. The image is correctly binary thresholded and final image is resized to 28*28 resolution where the letter remains legible. The image is correctly saved into the Final_Image folder of current session folder.

3. The trained CNN models have accuracies of around 80% and are able to make to right predictions. The fact that I have created a separate program[39] used to test the accuracies of trained models on a separate dataset and record the results within a text file is very useful. The majority voting method[40] that I have used to make up for the lack of training data is very smart as well. All predictions made on the images of letters/text (used for demonstration) were correct.

Overall, all objectives have been met and Ms Biletchi is very pleased about the selection of images of letter/text that I have used to demonstrate my program and has made repeated comments about my final product being "very good" which is satisfying.

However, Ms Biletchi has also made suggestions about future development of this program, where I should train the model so that it could be able to recognize cursive text which is more "human-like", and also develop an easier way of importing images of the handwritten text as current I had to take the picture using my phone and then send it to my laptop through email which is very convoluted.

5.3. Analysis of feedback

Ms Biletchi' s comments on my program is very positive which is good to hear, and I am glad to know that my program is working while be used by other users instead of myself.

The GUI is very easy to use with minimal number of buttons that each achieves a simple task, along with a simple user guide, hence user don't need to have any advance knowledge to be able to use this program and produce a prediction, as all the advance processing is done at the background.

[39] See section 3.6
[40] See section 2.6.4

The program doesn't exit unless the exit button on main menu is pressed, which avoids potential mistakes while user is making an input. When a non-image file is imported, error message is displayed and returns back to main screen which once again avoids false inputs.

The result of image pre-processing is satisfactory as the noise and shadows are removed and even after resizing to 28*28 pixels, Miss Biletchi is still able to recognize the letter. All images pre-processed are saved and named properly and has white background and black foreground as required.

The predictions made by the trained models are of satisfactory accuracies, where the correct predictions have been made to all images of letters and text used for demonstration purposes. Overall, all the specific objectives have been achieved.

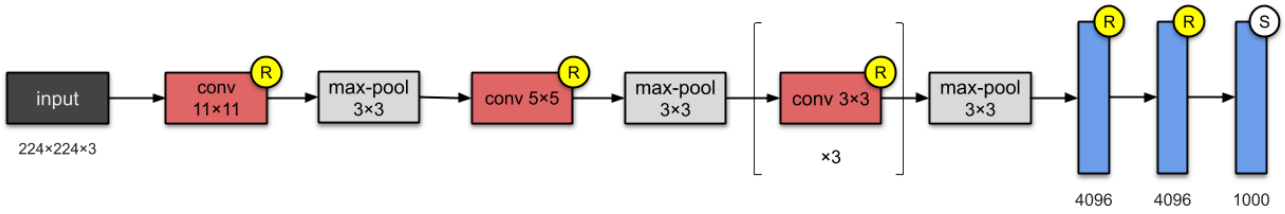
There are 2 suggestions that Ms Biletchi have made to take my program one step further, the first one is developing an easier way of importing image and the second is to recognize cursive handwriting, of which would be mentioned in the next section.

5.4. Future development

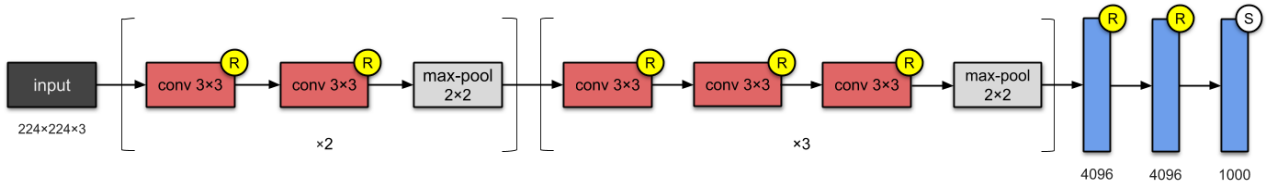
There are a few ideas in my mind about the future development of my program, where the first one is to improve the speed of prediction generation.
Currently, I have used majority voting method where multiple models are used to generate prediction on each letter within the Final_Image folder of current session, which is an extremely slow process.

The reason why I have to use this method is mentioned in section 2.6.4 already. Hence, in order to speed up the prediction process, one way that I could think of is by reducing the number of models used but increase the accuracy of each model, and to do that, I will need a much bigger dataset (rather than 5000 images in each folder) and use that dataset to train my CNN models.

Also, I could use some well-known CNN architectures that have proven to have high accuracy such as AlexNet and VGG-16 with architectures shown below:



AlexNet



VGG-16

Images taken from: https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d#e276

Secondly, another improvement that I could make is that currently the image pre-processing methods is able to distinguish the different between lines and letters, hence is the text/letter is written on line paper, the lines would be recognize as part of the text/letter and will be saved, leading to false predictions.

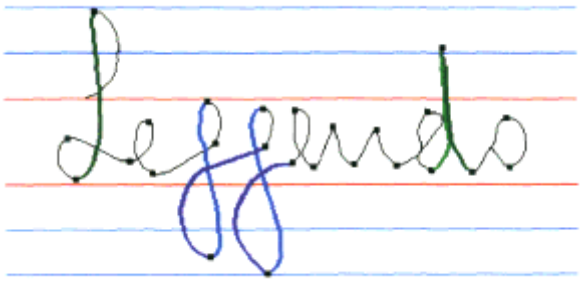
I believe that one way of doing this is by improving my find_ROI method, where I could adapt the selection statement to avoid very distorted ROI from being saved, e.g. if the width is 3 times the height or the other way around, this could potentially be a line that has been recognize as a region of interest and wouldn't be saved.

Another way of doing this is by using the power of CNN, where I could train a model that specifically recognizes images of lines and not lines, and I could implement this model into the adapted find_ROI method mentioned above to stop lines from being saved.

Thirdly, in order to make the image importing process easier, I have thought about using an external touchpad, just like the touchscreen on phones of which could connect to the laptop through a USB cable, and the result of the text/letter written on the touchpad could be fed directly into the program to be pre-processed, then Open Image button option on the main window wouldn't be needed anymore.

At the end, I would also have a deeper investigation in cursive text recognition, of which I believe would be a very difficult but at the same time very interesting topic to investigate about. It is difficult due to the fact that all the letters are connected to each other, hence the ROI method that I am using currently wouldn't work at all.

I have done some brief research into this area, and I have discovered the following. During the pre-processing of images of cursive handwriting, the image would first have noise removal applied as usual. Then the text is 'routed', where the program would trace the writer's ink strokes as shown below, where the line between 2 points would be a single stroke.



After identifying the strokes, as the connection between corresponding strokes have significant change in curvature, by identifying the changes of curvatures, this enables the letters to be ‘cut’ at specific points (the black dots) and separate them into individual letters.

Image and information taken from: http://www.recogniform.net/eng/chr-cursive-handwritten-recognition.html

Investigation on Image Processing Methods and how machines can learn to recognize hand-written English text