

# FIT3081 ASSIGNMENT 3

## GROUP A01

Made by: Tanul Gupta, Tan Pei Sheng, Yap Rui Yi

---

### Table Of Contents

---

[Table Of Contents](#)

[Task Allocation](#)

[Methodology](#)

[Code Explanation](#)

[Pre-Processing Character Image](#)

[Character Segmentation Of Numberplate Images](#)

[Training the neural network](#)

[Test Results](#)

[Training images that failed to meet criteria set for target output](#)

[Test results \(cropped training\)](#)

[Test results \(license plates\)](#)

[Recommendation](#)

[References](#)

EXECUTE THE FOLLOWING FILES IN ORDER TO RUN THE ENTIRE PROGRAM:

1. preprocessing.py
2. optimized\_training.py
3. test\_neural\_network.py
4. CharacterSegment\_Testing.py

## Task Allocation

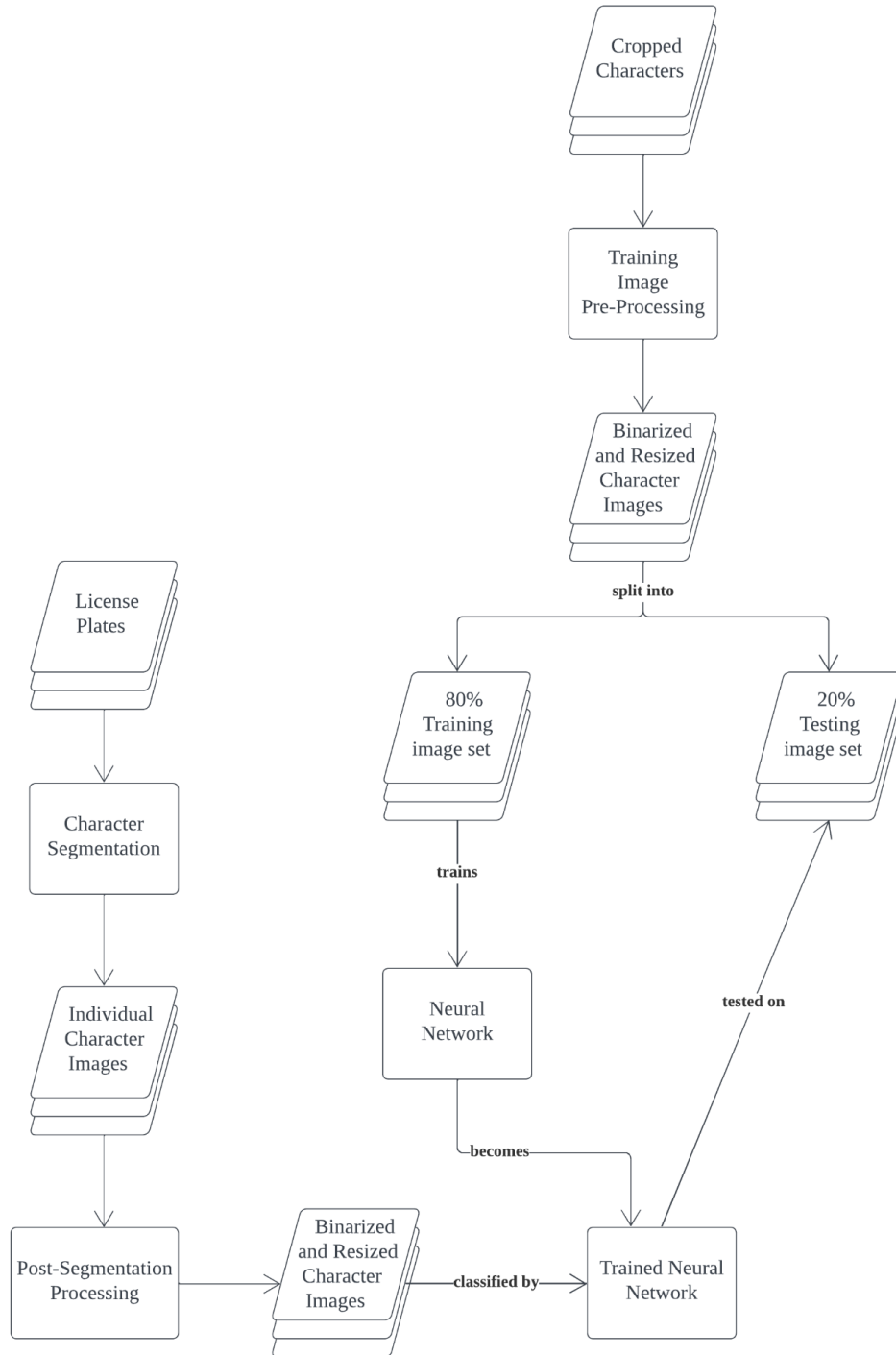
Tanul - Character Segmentation code, optimizing/increasing the speed/efficiency of the neural network, report, collecting the training and testing data.

Pei Sheng - cropping images to acquire training images, research on how accuracy can be improved, testing some of the training configurations, draw a flowchart for the report, writing parts of the report. Help create the preprocessing needed for neural network

Rui Yi - created the neural network, created the preprocessing needed for neural network along with tanul and pei sheng. Also created the code needed to test and verify the accuracy of 20% testing images and segmented images. Help to crop images for training and testing data. Also wrote parts of the report

# Methodology

Our flowchart:



Our methodology consists of two parts, 1) the training of a character recognition neural network that is capable of classifying input images into different categories of characters, 2) the segmentation of car license plate images into multiple images each containing only a single character. A character can be a number ranging from '0' to '9' inclusive, as well as the alphabets 'B', 'F', 'L', 'M', 'P', 'Q', 'T', 'U', 'V', 'W'. The segmented images are then fed into the trained neural network to be classified.

#### Neural Network overview

The objective of the neural network is, given a certain input image, classify it using the weights established during training by traversing through the underlying graph data structure beginning from the input layer and ending at one of the output nodes in the output layer. Once arrived at one of the output layer nodes, the input image is then classified as the category associated with that output node.

To measure the performance of our neural network, we will first test it against training testing data (20% of the previously excluded training data, reserved for testing the trained neural network), then test it again with segmented car license plate images.

To achieve the objective with satisfactory accuracy, we tried several different training configurations. We adjusted the number of epochs performed during training, discovering that the sweet spot is around 1000 without applying learning rate decay, and around 500 if applying learning rate decay. We also tried several different sets of training images, to determine the characteristics of good training images.

#### Car license plate segmentation overview

The objective of segmenting a car license plate image into multiple images each containing only a single character is to produce classifiable images for consumption by our character recognition neural network.

To measure the performance of the segmentation, we first visualize the image regions our algorithm segmented by painting a colored border around the segment and displaying the entire image on screen.

After satisfactory performance has been verified we will write the results of segmentation to disk for later consumption by the neural network.

### Best training images characteristics

We first began training without much consideration for training image characteristics, and as expected the performance of the resulting neural network is extremely inaccurate (accuracy in the range of 35%-45%). We had too many images containing characters that are slightly rotated in different angles which confused our neural network, and at the same time not enough total training images to clarify the important features. To combat this problem we define the following requirements for training images:

1. Contain only a single character
2. The character should be centered
3. The color of the character should be of a higher intensity level than its background
4. The background should be black, but other dark colors are acceptable if high contrast exist between the character and the background
5. Images belonging to the same category should be very similar to each other.
6. Due to the nature of our segmentation method, the border of the training images should only have 1 bit of background pixel between the furthest pixel of the character from the center of the image and the end of the image
7. The images should not be too 'noisy' or blurry
8. Majority of the images should not contain slanted/rotated characters

The following code explanation section will explain our methodology in detail.

## Code Explanation

### Pre-Processing Character Image

This step involves preprocessing the training images and testing images, in the end transforming them into a format that can be easily fed into the neural network. A more detailed description is shown below.

the steps are:

1. Each training character is converted to a fixed size of (20, 40)
2. Then the canny algorithm is applied to binarize the character into 2 colours, with the upper threshold value being 200 and lower threshold value being 150, to filter out as much redundant edges as possible, as well as preserving the structure of the character.
3. Next, the training image will be converted into an numpy array of 1 and 0, by converting all the 255 to 1 via the np.where() function
4. After that, numpy array for the image will be flatten to be used as the input for the neural network, at the same time the target values for the training images will be appended to a list
5. The four above steps are also applied to the testing images

### Character Segmentation Of Numberplate Images

We perform the following:

1. Resizing the numberplate image to a fixed size (357,107)
2. Convert the image to grayscale.
3. Apply gaussian blur to the image to reduce the noise, with a kernel block size of (9,9)
4. Do thresholding to binarize the blurred image. We tried to find the optimal setting such that most of the relevant image (white part of the character remains white, rest black).
5. Perform **connected component analysis** using the cv.connectedComponents(img) function. We are only interested in the labels and not the total number of connected components found in the image.
6. We then loop over to identify the important labels by discarding the background labels which are given the value 0, aka the black pixels.
7. We then create a label mask of zeros the size of the thresholded image which then has the key areas (white pixels) converted to white.
8. We set a lower bound and upperbound of pixels that must be met, to meet the criteria for being identified as a character. If it is met then to a mask these pixels are added..
9. We then make use of the findContours function to identify the contours in the masked image and then use boundingRect function to identify the bounding rectangle of the contours
10. We created a function called compare that sorts the bounding rectangle from left to right based on the X-coordinates first and then top to bottom based on the

Y-coordinates if the X-coordinate is similar. This is done to ensure the stability in the order of the characters in the numberplate meaning, if there is a numberplate VB32 then character V comes before B in the boundingBoxes list.

11. We created an empty list for `average_width`, this will help us in identifying any boundingboxes that are too large or small.
12. We now loop over all the boundingboxes which are basically the characters of the numberplate
13. For each rectangle, the code checks if the `average_width` list is empty. If it is, the width of the rectangle is added to the list, and a cropped image region corresponding to the rectangle is extracted from the original image. The cropped image is then appended to the `img_rectangle` list.
14. If the `average_width` list is not empty, the code compares the width of the rectangle with the mean width calculated from the `average_width` list.
15. If the width is higher than the mean width plus 5 pixels, the width is divided by 2 and added to the `average_width` list. Two cropped image regions are extracted from the original image, corresponding to the left and right halves of the rectangle, and both regions are appended to the `img_rectangle` list.
16. If the width is lower than the mean width minus 5 pixels, the width is multiplied by 1.5 and added to the `average_width` list. A cropped image region is extracted from the original image, corresponding to the extended width of the rectangle, and it is appended to the `img_rectangle` list.
17. If the width is within the acceptable range (close to the mean width), the width is added to the `average_width` list, and a cropped image region corresponding to the rectangle is extracted from the original image. The cropped image is then appended to the `img_rectangle` list.
18. The loop continues until all the bounding rectangles have been processed.
19. Next, to prepare the bounding rectangles to be fed into the neural network for testing, we will apply the same preprocessing as listed above in the [Pre-Processing Character Image](#)
20. Each of the bounded rectangle will be tested using our neural network by executing both the `Forward_Input_Hidden()` function and `Forward_Hidden_Output()`.

## What is connected component analysis?

Basically, it groups and labels connected components, in this case the white pixels. If the pixels are adjacent to each other they are provided the same label. A good visual explanation I could find was from this website: (*Label and Measure Connected Components in a Binary Image - MATLAB & Simulink*, n.d.).

## Why the `average_width +/- 5` is taken?

I noticed that the mean width value for the character in each value ranges in the plus minus 5 range.

From observations from running the program we can observe that there were two cases which were considered erroneous. There was one case where in a numberplate 2 characters 9 and 7 were touching, in this case the rectangle contained both the images so we had to

generate code for higher than the mean, where we divide the box into 2 if the value was greater than 1.5 times the mean value of all the characters of the numberplate seen so far. Similarly for lower than mean where the rectangle generated is cropping part of the character, so we need to make the rectangle slightly larger than current size.

## Training the neural network

**Initialization:** The NeuralNetwork class is defined with attributes such as the number of hidden neurons, number of output neurons, and initial learning rate. The weights, biases, and other variables are initialized.

**Weight Initialization:** The weights and biases between the input and hidden layers (weights\_ji, bias\_j) and between the hidden and output layers (weights\_kj, bias\_k) are initialized randomly within a specific range.

**Training Epochs:** The neural network is then trained in iterations of 'epochs' for each training sample until a predefined number of desired epochs is reached, within each epoch the following methods will be executed:

- **Forward Propagation:** The input is fed forward through the network. The inputs are passed through the hidden layer (Forward\_Input\_Hidden method) (Out j). Then, the outputs of the hidden layer are passed through the output layer (Forward\_Hidden\_Output method)(Out k). The output values are computed using the sigmoid activation function.
- **Error Calculation:** The error between the network's actual output and the desired target output is calculated using the mean squared error formula (Error\_Correction method).
- **Backpropagation:** The errors are then propagated backward through the network to update the weights and biases. The delta values for the output layer (Weight\_Bias\_Correction\_Output method) and hidden layer (Weight\_Bias\_Correction\_Hidden method) are calculated using the chain rule.
- **Weight and Bias Update:** The weights and biases are updated using the delta values calculated in the previous step (Weight\_Bias\_Update method). The learning rate determines the factor of the update.

**Learning Rate Decay:** Learning rate is updated during training (update\_learning\_rate method). In the provided code, the learning rate is decreased over time. This is done to reduce the number of epochs we have to perform without decreasing the neural network's accuracy.

**Save Weights and Biases:** After training, the final weights and biases are saved to separate files (save\_weights\_bias method).

**Accuracy Evaluation:** The accuracy method is called to evaluate the network's performance on the testing data. It calculates the accuracy, number of correct predictions,



and provides additional information such as the accuracy for each character. If carplatechars is provided, it also classifies carplate images based on the output.

**File Reading:** The Read\_Files method is called to read the training and testing data from separate files.

## Why do we use a learning rate decay technique?

Learning rate decay is a technique that can allow better convergence and optimization performance.

**Convergence Speed:** In the early stages of training, when the weights and biases are far from optimal values, a higher learning rate can help the network converge faster. However, as training progresses and the parameters get closer to their optimal values, using a high learning rate can cause overshooting, making it difficult for the network to settle into the optimal solution. Learning rate decay addresses this issue by gradually reducing the learning rate over time, allowing the network to make smaller and more precise updates as it approaches convergence. This can speed up the convergence process overall.

**Stability and Robustness:** Rapid and large updates of weights and biases can lead to instability during training. By reducing the learning rate, you can achieve smoother and more stable weight updates, which can improve the overall robustness of the network.

**Generalization and Avoiding Overfitting:** A high learning rate can cause the network to quickly adapt to the data causing overfitting. This means that it may perform well on one sample data but not on another, and therefore we can say that the model is not for general data. By decreasing (decaying) the learning rate we can sort of regulate the extent to which the model adapts to the data and in turn make the model more general.



We have used the common method from the following: (Haswani, 2020)

## Test Results

### Training images that failed to meet criteria set for target output



None of them failed if we perform the training over many epochs (e.g. 1000 without learning rate decay) or 500 epochs with learning rate decay. We can see many failures when using lower epochs such as 20 or 30. This is due to not having enough rounds of training

### Test results (cropped training)

Segmented Image	Classified By Our Model As
	<b>B</b>  This probably happened due to the fact that the character edges look more similar to the character B instead of 8
	<b>V</b>  This probably happened due to the fact that the character edges look really blurry, and the outer boundary of the M is not fully captured in the picture

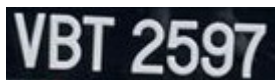
Our character classification performs quite excellently with 95% accuracy as it successfully classifies 38/40 of the characters accurately.

## Test results (license plates)

Segmented Image	Classified By Our Model As
	<b>8</b>  This probably happened due to the nature of the 9, it is slanted and slightly cut-off
	<b>B</b>  This is probably due to the nature of the cropped- image of the 0 there is a lot of glare in the background, extra black space at the top of the image.

Our license plate classification performs quite excellently with 97.1% accuracy as it successfully classifies 67/69 of the characters in the number plates accurately. The only 2 images that it fails are sort of erroneous images from the cropping conducted in the algorithm itself.

Original image for the misclassified as 8:



Thresholded image in the character segmentation:



As you can see 9 and 7 appear to be touching and therefore the program identifies the 9 and 7 to be connected components. We dealt with this part in **point 15 in [Character Segmentation Of Numberplate Images](#)**. The 9 has some parts of its rear cropped which can be attributed to the poor classification.

Original image for the misclassified as B:



Thresholded image in the character segmentation:



As you can see 0 appears to have some noise this can attribute to the misclassification. It appears to be an upside down Q due to the nature of the original image.

## Recommendation

As mentioned in our test results and methodology, adjusting training configurations provided significant improvements in accuracy. Specifically the characteristics of training images have a huge impact on final accuracy, and the number of epochs follow a logarithmic trend where after a certain number (around 500 in our case, after applying learning rate decay) the accuracy gains plateaus.

So in order to further improve the accuracy beyond 97.1%, we must look at training data improvements rather than simply increasing the number of epochs.

To improve training data, we can try increasing the total number of training images for each character. The requirements for the training images will also need to be updated to make better use of an increase in sample size, we can include more varieties of each character (such as rotated/slanted characters or different font) so the neural network can be more generalized while maintaining high accuracy.

---

---

## References

---

---

*Label and Measure Connected Components in a Binary Image - MATLAB & Simulink*. (n.d.). MathWorks. Retrieved June 1, 2023, from

<https://www.mathworks.com/help/images/label-and-measure-objects-in-a-binary-image.html>

Haswani, V. (2020, September 3). *Learning Rate Decay and methods in Deep Learning* | by Vaibhav Haswani | *Analytics Vidhya*. Medium. Retrieved June 2, 2023, from

<https://medium.com/analytics-vidhya/learning-rate-decay-and-methods-in-deep-learning-2ce564f910b>