Riya Roopesh
301360795

CMPT 412

Project1

PART 1: Forward Pass

The following results were obtained by running **test_components.py**

Q 1.1: Inner product layer

Ans: In order to complete the inner_product.py and from a completely connected layer, I implemented the following equation **f (x) = W x + b** within the ***inner_product_forward(input,layer,param)*** function. This is the matrix multiplication of the weights W with the input followed by the addition of the biases, to obtain the desired output. As a result of the well functioning layer we obtain the result as shown in fig1.

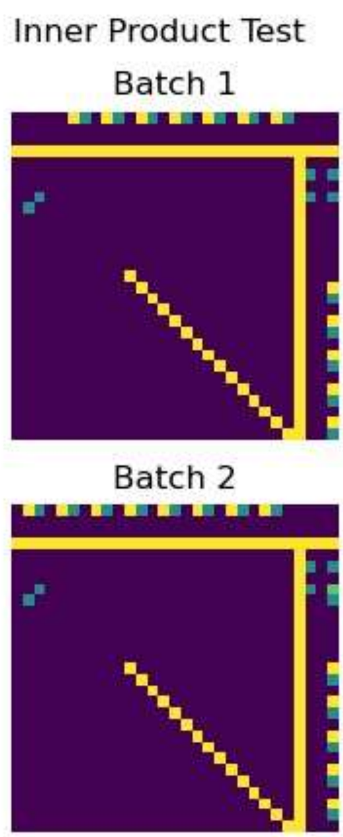


FIG1: INNER PRODUCT LAYER

Riya Roopesh
301360795

Q1.2: Pooling Layer

In this part we fill in the code within **pooling_layer.py**. The basic idea is to reduce the size of the feature maps by replacing a local region of the feature map with aggregated stats, and this was implemented by the use of the following formula $f(X, i, j) = \max_{x \in [i-k/2, i+k/2], y \in [j-k/2, j+k/2]} (X[x, y])$ .The result is shown below(fig2).

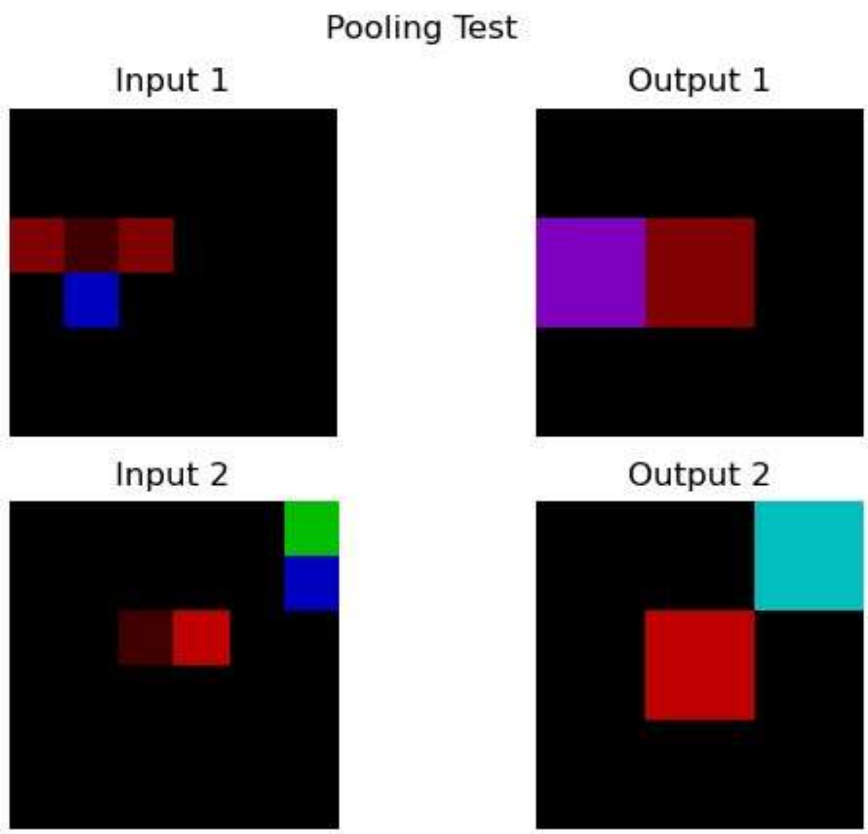


FIG 2: POOLING LAYER

Q1 1.3 Convolution Layer

In this part of the assignment, I filled in code within the **conv_layer.py**. We mainly utilized the function provided to us ***im2col_conv_batch(***) which enabled in making the computation a lot faster. This was then followed by the implementation of the following formula **f (X, W, b) = X ∗ W + b** to obtain the desired output. The results are shown below in fig3 and fig4.

Riya Roopesh
301360795

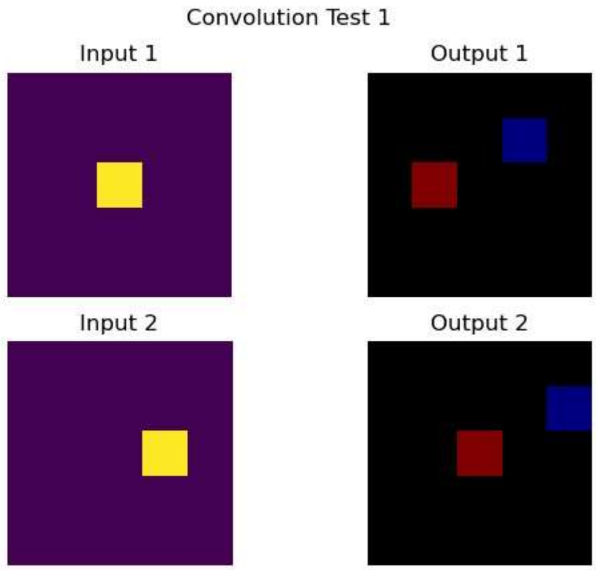


FIG 3 Convolution1

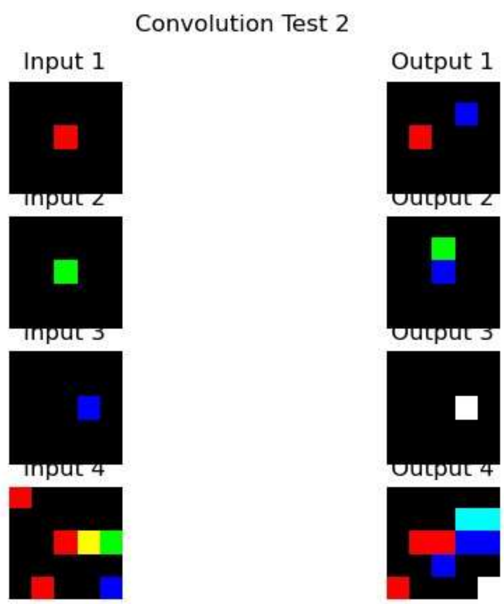


FIG 4: Convolution 2

Q 1.3 ReLu:

For ReLu we used the following formula **f (x) = max(x, 0)** and implemented it within **relu.py**

Riya Roopesh
301360795

PART2: Back Propagation

Q 2.1 Relu:

Implemented within **relu.py** under the ***relu_backward(output, input_data, layer)*** function.

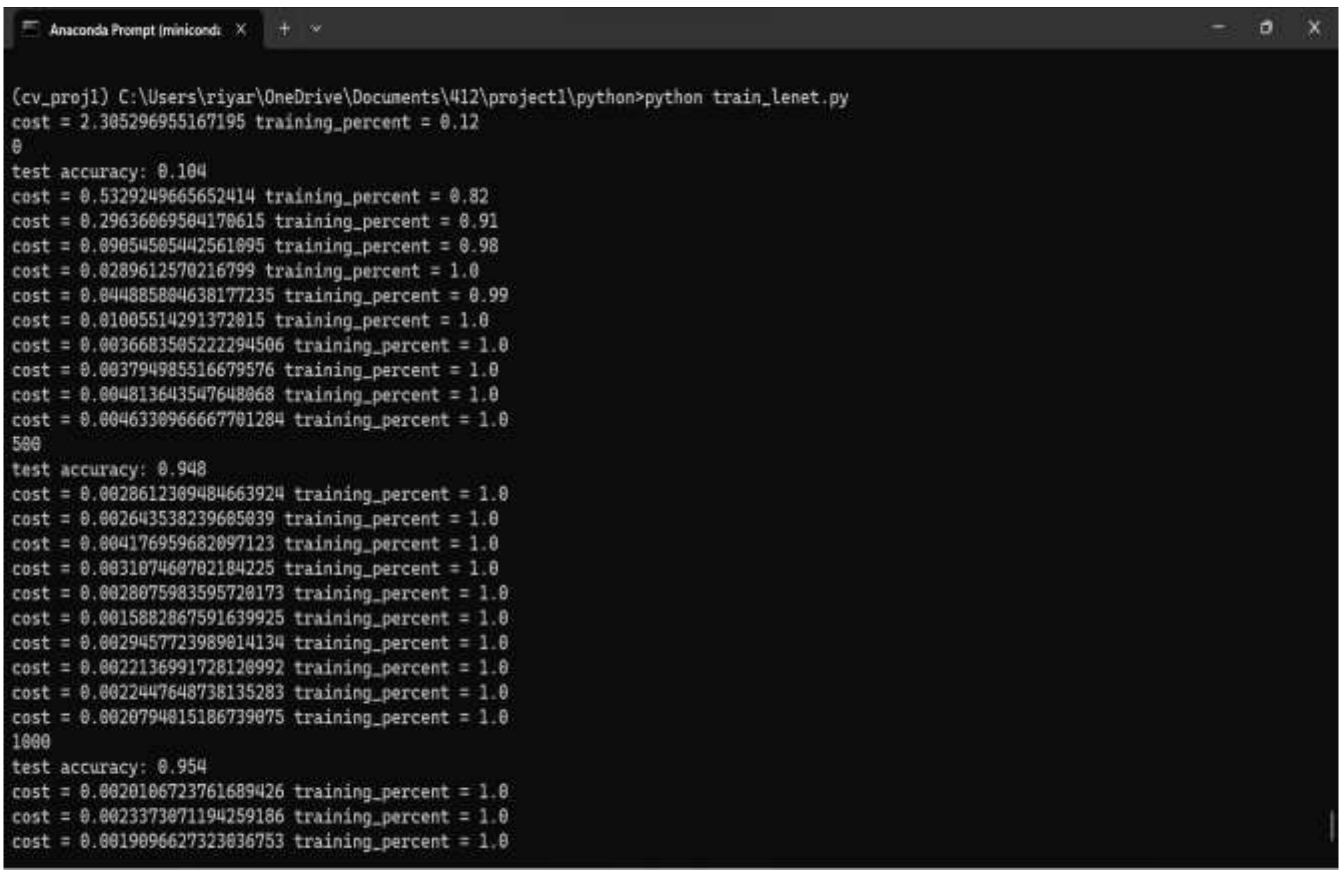
Q 2.2 I**nner Product layer:**

Implemented within **inner_product.py** under the ***inner_product_backward(output, input_data, layer, param).***

PART3: Training

Q 3.1 Training

After running **train_lenet.py** we obtain the following accuracy for **2000** iterations.



```
(cv_proj1) C:\Users\riyar\OneDrive\Documents\412\project1\python>python train_lenet.py
cost = 2.305296955167195 training_percent = 0.12
0
test accuracy: 0.104
cost = 0.5329249665652414 training_percent = 0.82
cost = 0.29636069504170615 training_percent = 0.91
cost = 0.09054505442561095 training_percent = 0.98
cost = 0.0289612570216799 training_percent = 1.0
cost = 0.044885804638177235 training_percent = 0.99
cost = 0.01005514291372015 training_percent = 1.0
cost = 0.0036683505222294506 training_percent = 1.0
cost = 0.003794985516679576 training_percent = 1.0
cost = 0.004813643547648068 training_percent = 1.0
cost = 0.0046330966667701284 training_percent = 1.0
500
test accuracy: 0.948
cost = 0.0028612309484663924 training_percent = 1.0
cost = 0.002643538239605039 training_percent = 1.0
cost = 0.004176959682097123 training_percent = 1.0
cost = 0.003107460702184225 training_percent = 1.0
cost = 0.0028075983595720173 training_percent = 1.0
cost = 0.0015882867591639925 training_percent = 1.0
cost = 0.0029457723989014134 training_percent = 1.0
cost = 0.0022136991728120992 training_percent = 1.0
cost = 0.0022447648738135283 training_percent = 1.0
cost = 0.0020794815186739075 training_percent = 1.0
1000
test accuracy: 0.954
cost = 0.0020106723761689426 training_percent = 1.0
cost = 0.0023373071194259186 training_percent = 1.0
cost = 0.0019096627323036753 training_percent = 1.0
```

Riya Roopesh
301360795

```
Anaconda Prompt (miniconda)
cost = 0.0022447648738135283 training_percent = 1.0
cost = 0.0020794015186739075 training_percent = 1.0
1000
test accuracy: 0.954
cost = 0.0020106723761689426 training_percent = 1.0
cost = 0.0023373071194259186 training_percent = 1.0
cost = 0.0019096627323036753 training_percent = 1.0
cost = 0.0016836184442240007 training_percent = 1.0
cost = 0.002530478919791092 training_percent = 1.0
cost = 0.001424981026587694 training_percent = 1.0
cost = 0.0013955312863448788 training_percent = 1.0
cost = 0.0012548579303198577 training_percent = 1.0
cost = 0.0012531599728476028 training_percent = 1.0
cost = 0.0016220289024063803 training_percent = 1.0
1500
test accuracy: 0.954
cost = 0.0011588890697309224 training_percent = 1.0
cost = 0.0010549078210999387 training_percent = 1.0
cost = 0.0016567510806783558 training_percent = 1.0
cost = 0.0012564284873629792 training_percent = 1.0
cost = 0.0021033447138982236 training_percent = 1.0
cost = 0.0014324693083293347 training_percent = 1.0
cost = 0.0018421925389339633 training_percent = 1.0
cost = 0.0015496281809593599 training_percent = 1.0
cost = 0.0012371622111722045 training_percent = 1.0
cost = 0.0019379486205896679 training_percent = 1.0
2000
test accuracy: 0.956

(cv_proj1) C:\Users\riyar\OneDrive\Documents\412\project1\python>
```

The above two snippet and the copy of the output depict that we obtained a test accuracy of **95%.**

Q 3.2 Test the network:

After modifying the test_network.py I was able to obtain the following confusion matrix.

```
(cv_proj1) C:\Users\riyar\OneDrive\Documents\412\project1\python
>python test_network.py
Confusion Matrix:
[[40  0  0  0  0  0  0  0  0  0]
 [ 0 58  0  0  0  0  0  0  0  0]
 [ 0  0 49  0  0  0  0  0  0  1]
 [ 0  0  0 55  0  1  0  1  1  0]
 [ 0  0  0  0 45  0  0  0  0  3]
 [ 0  0  0  2  0 35  0  1  3  0]
 [ 1  0  0  0  0  1 44  0  0  0]
 [ 0  0  2  0  1  0  0 42  0  1]
 [ 0  0  0  0  0  0  0  0 48  0]
 [ 1  0  0  1  0  1  0  1  1 60]]
```

It is important to note that the confusion matrix may differ slightly every time we run test_network.py The image provided above depicts the output I received when I ran it. In this image we can notice that the most confusing pairs are the following:

8 and 5: It seems as though the network had a hard time differentiating the 5’s from 8. One reason for this is probably because the curve at the bottom of 5 is very similar to 8 and if the rest of the strokes of 5 aren’t written with care it may result in such a confusion.

Riya Roopesh
301360795

9 and 4: It seems that the network identified 4 as 9 multiple times, this is possibly because there are two ways of writing 4, with the two lines connecting and the top and without connecting. I believe that when it was written with the two straight lines connected at the top the network confuses it for a 9 as they may foster a similar look.

Q 3.3 Real world testing:

For this part I created a new python script called **real_world_sample.py** . I stored my samples within the images folder, under another folder called samples. I made sure to resize my images to (28,28) prior to saving them in my folder.

The following are the samples I used in this part:

```
(cv_proj1) C:\Users\riyar\OneDrive\Documents\412\project1\python
>python real_world_sample.py
Number of correct predictions: 8
[1 2 3 4 5 6 7 8 9]
[1 2 3 4 5 6 7 8 4]
```

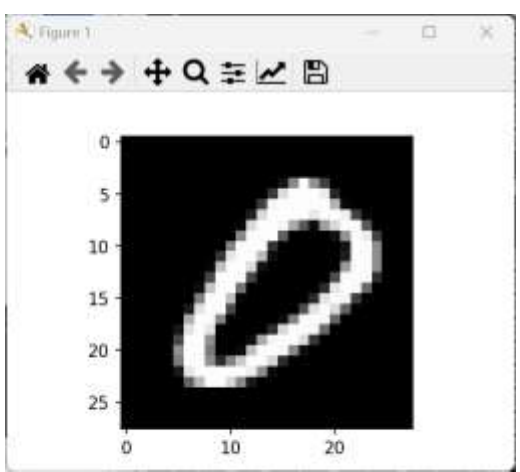
The above snippet is the output I received, the first list is the actual list of elements and the second list depicts the predictions. The output shows that we got 8 out of 9 correct, this shows that the network has a good accuracy. And it confirms the confusion pair shown above that the network may get confused between **9 and 4**.

PART4: Visualization

Q 4.1:

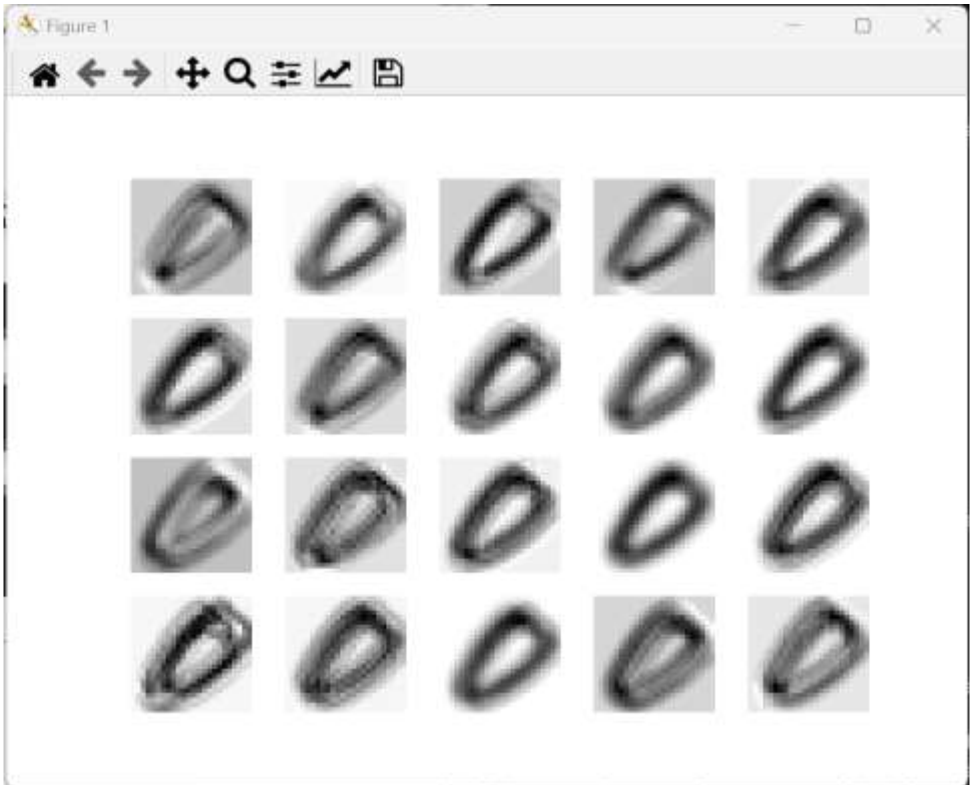
For this part of the assignment we implemented the code within **vis_data.py** and obtained the following outputs.

Input :

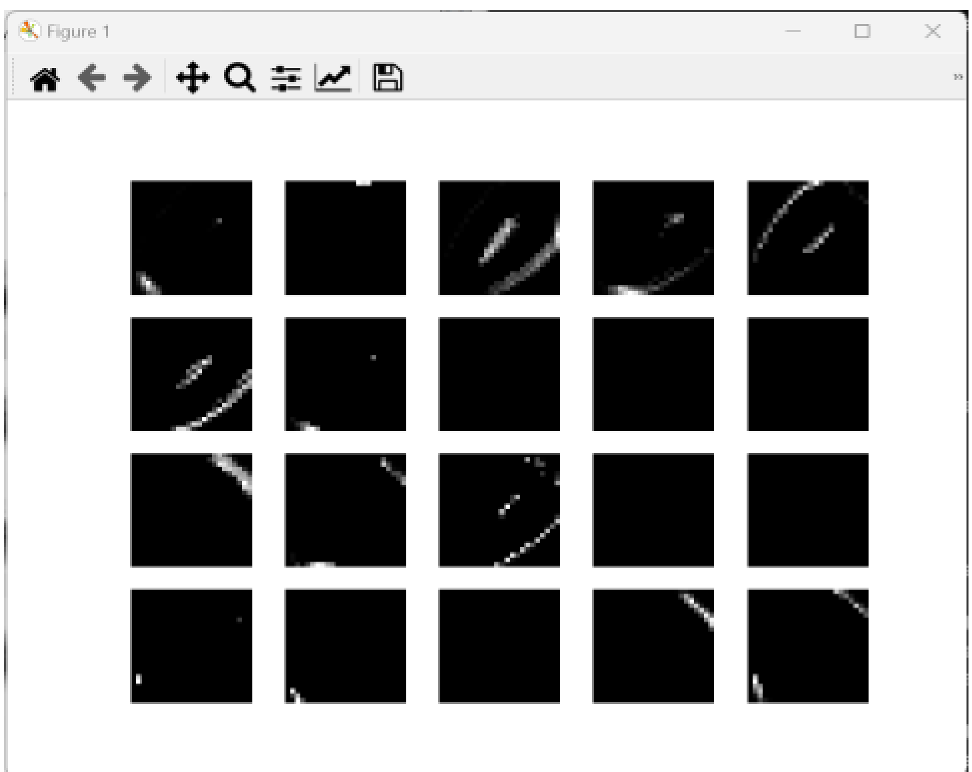


Riya Roopesh
301360795

Convolution layer:



ReLu Layer:



Riya Roopesh
301360795

Q 4.2 Explanation:

In the above images we notice that the convolution layer as compared to the original input image has certain aspects filtered out and blurry but still portrays the digits in a legible manner. We can also notice that grayscale shades seem to be inverted, with the number being darker as opposed to the background, this is probably due to transpose.

On the other hand we notice that in the ReLu layer we notice just a few specs, this is all of the aspects that got filtered out(negative pixels) in the convolution layer. Thus depicting that our network has learnt to identify the essential features such as the curves and the straight lines within the different layers.

<u>PART5: Image Classification</u>

In this part of the assignment I created a new python script called **ec.py** within the python folder. In this script, I mainly utilised the inbuilt **OpenCV** python functions to implement my code. I utilised the inbuilt functions to read the image and convert it to grayscale. I then used **cv2.threshold( )** to binarize the image and lastly, I utilized **cv2.findContours( ),** to find the individual numbers within the single image. For each image I had to tweak the values of the **bounding box** separately, however all the changes were being applied to each bounding box within the image. This did make it slightly troublesome, as each contour found by the function were of different shapes and some did not even capture the entire digit and required tweaking. I also noticed that **cv2.findContours( )** did not find the numbers in order, and so I had to input the individual bounding boxes one by one, to ensure that the prediction was made in order of the contours found.

On running ec.py we get the following images are my results of this part:

image1:

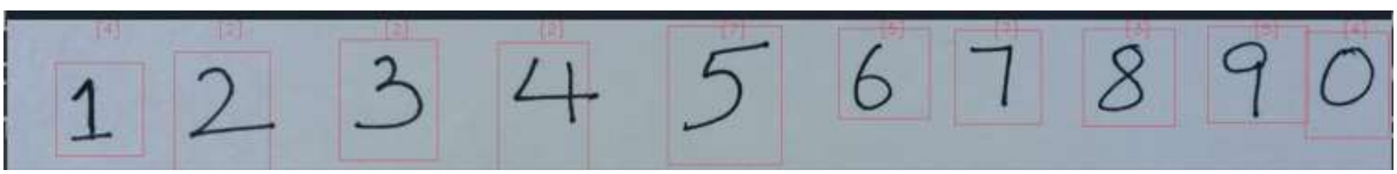


image2:

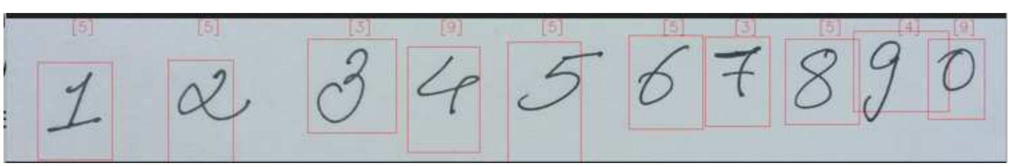
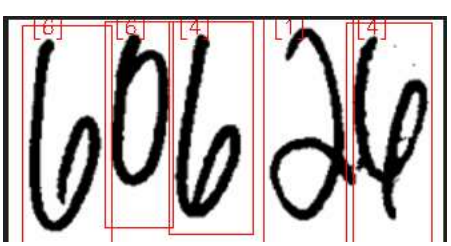


Image3:



Riya Roopesh
301360795

Image4: