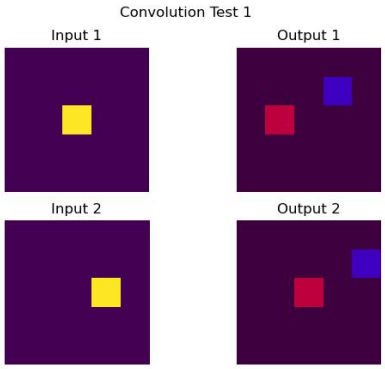
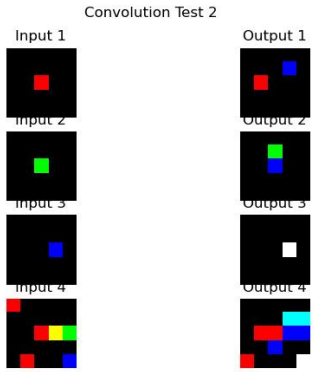
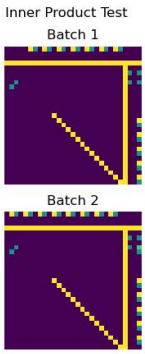
Assignment1 Report
Lihao Qian 301406445

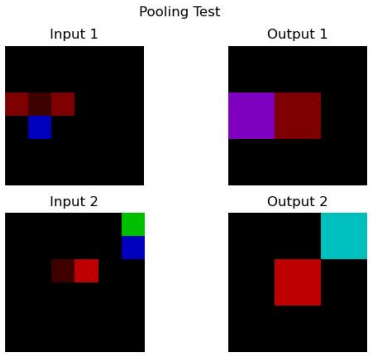
Late for 2 days

For the 4 test layer output, they all exactly match up with the demo example.





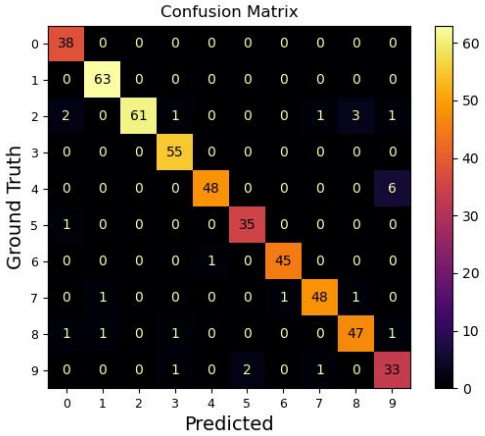




3.1

After 2000 iterations as required, my model stables at
test accuracy: 0.952
cost = 0.0008311737628735044 training_percent = 1.0
It seems a little overfit, feels like the optimal is to terminate trainning at 1500 iterations

3.2

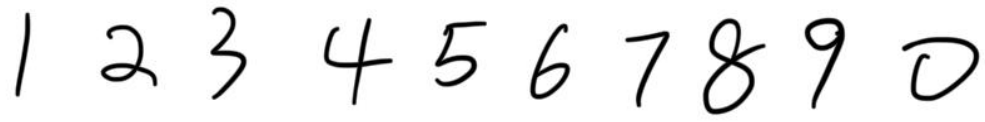


In the confusion matrix for my model, I gave 500 images a batch to the network.

- The most confused pair is 4 and 9, with 6 of them guessed wrong, for this pair I think the main reason is the loop structure on the upper part of number 4 and 9 that makes the network gave the wrong prediction.

- The second confused pair is 2 and 8, with 3 of them guessed wrong, for this pair I think the zigzag structure of 2 and 8 make the network confused when running through the edge extraction layer.

3.3



The number above are used for test the network, I manually separate all the digits and put them under the ../results/hand_draw_img. By running python3 test_hand_draw.py, we get output of below

```
(cv_proj1) ➜  python python3 test_hand_draw.py
Saved processed_handraw images
Ground Truth:
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Prediction:
[0, 1, 2, 3, 9, 5, 6, 7, 8, 2]
```

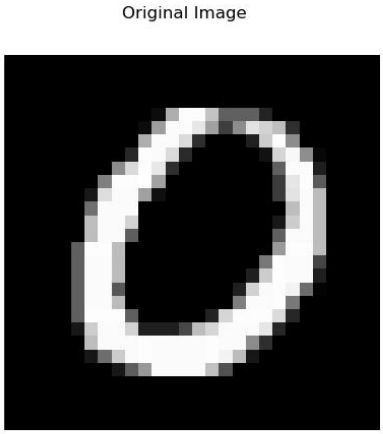
Note that my code saves all the processed images in ../results/hand_draw_img_result, for visual comparision that 10 digits I wrote above, they are transformed into below:

Although all the pre-processes effort, there is still wrong classification:

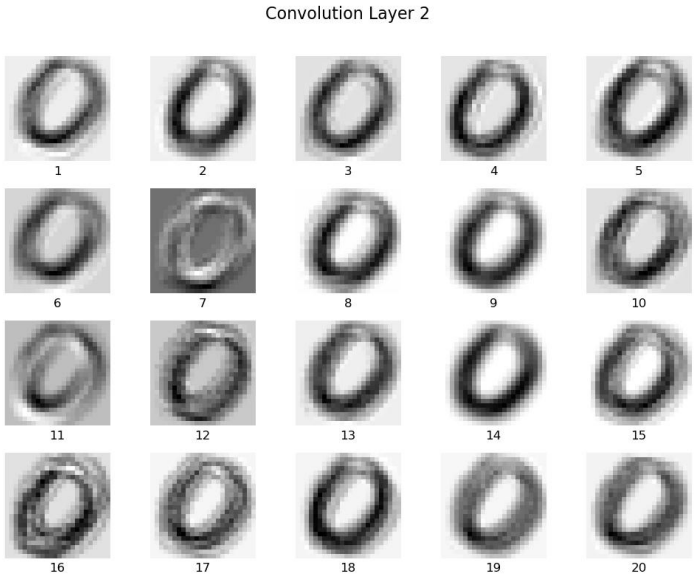
- The model predicts 9 for number 4, which assume the same reason explained in Q3.2, the loop structure on the upper part of number 4 and 9.
- For prediction 9 to 2, the only reason is the image I feed into the network is too sharp and too thin, which is possible loss detail if I down sample the image. That makes the edge of 9 looks like number 2.

4.1

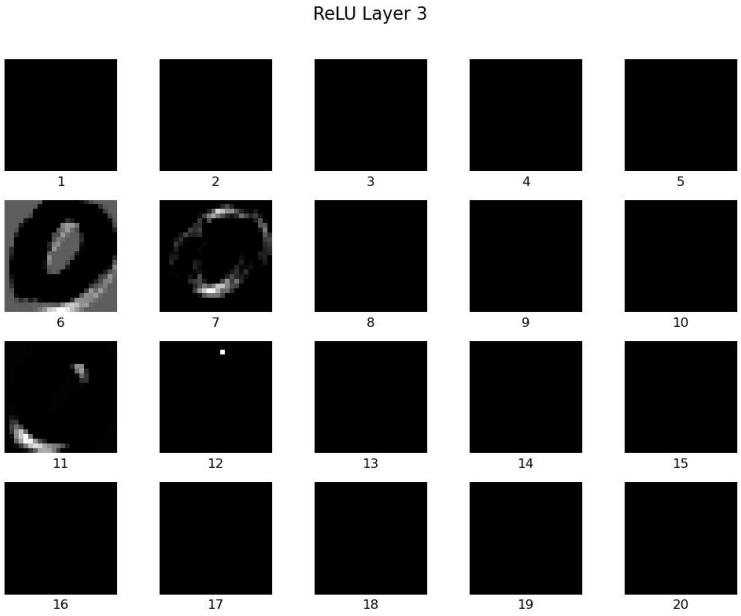
By running python3 vis_data.py, with original image number 0:



The convolution layer looks below:



And the Relu layer:



4.2

Compared to the original image, the colors in the convolution layer appear inverted, and each feature extraction approach behaves differently. Some methods emphasize the vertical and horizontal edges, while others add strokes or apply a Gaussian kernel to blur the original image. All the layers looks pretty cool, some of them have almost like a 3D effect, where I think the layer amplified the gray scale change in the edge of the original image.

Moving on to the ReLU layer, we can see that only 4 feature map has output. This indicates that most of the inputs from the convolution layer are dropped, meaning that most inputs are negative. The ReLU layer turns all negative values into zero. This also suggests that the model might be overfitting. As I mentioned in Q3.1, the required 2000 iterations are too many, and we should consider terminating the training earlier.

5

Image1

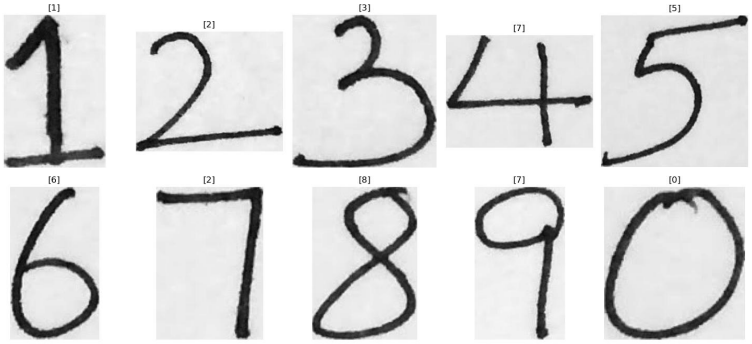


Image2

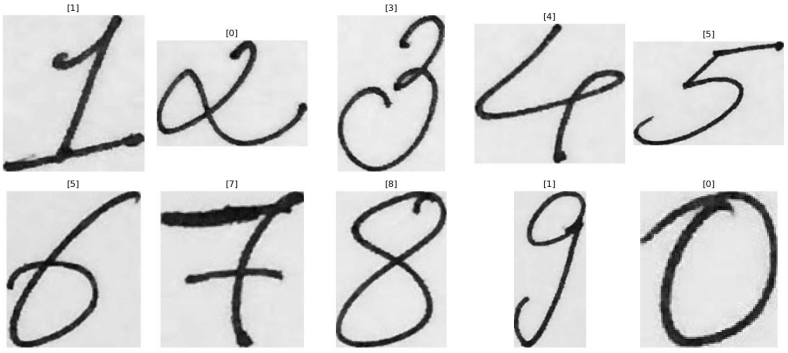


Image3

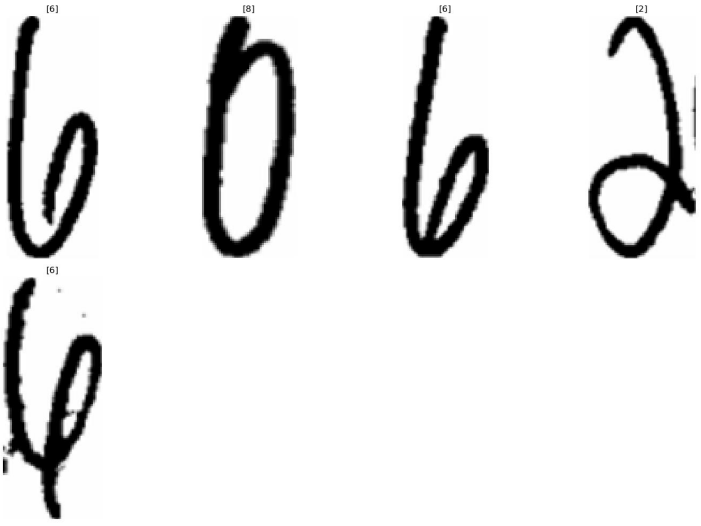
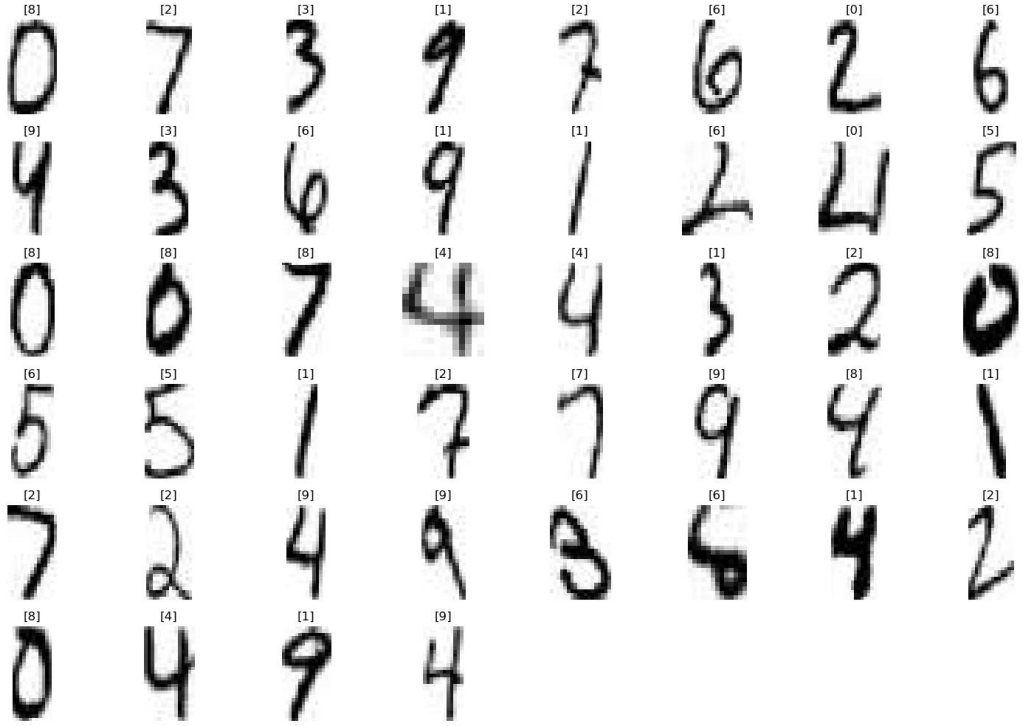


Image4



For the first three images, the predictions are relative more reliable due to the clear and sharp original images. In the code, I used GaussianBlur and dilation to enhance the images for better predictions by my model. I also added more padding to the iamge to make more look like the data we train the network. However, when using images of varying quality, such as image 4, the same set of parameters performs poorly, resulting in an accuracy drop below 70%. One thing that worth notice that when I tried to draw the box out of the picture using cv2, sometimes it will give random pixels as a “actual” digits. To fix that, I ignored all boxes with height or width under 7 pixels. In a nutshell, this model still has room for further tuning.