CMPT 412: Assignment 1 Digit recognition using a CNN

Name: Henry Chen
Student id: 301422117
Email: hca161@sfu.ca

NOTE FOR TA: I zipped all my files together into one zip file, unlike the the one given to us where all our files were contained in the large parent folder project1.

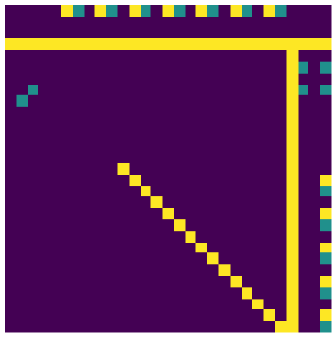
Part 1: forward pass

- Inner Product: I calculated the output data by taking the dot product of the transposed param['w'] matrix and the the input['data'] matrix and added a reshaped param['b'] matrix to fit the dimensions of the output
- Max Pooling: I iterate through each batch and channel via the kernel and perform max-pooling on the input data image. For each channel and pool window/kernel, I find the max value and store it in the output data. I reshape the output data at the end to reflect it's correct dimensions.
- Convulutional Layer: I prepped the input data by first using the provided im2col_conv_batch function. Then, for each batch, I apply convolution by multiplying the weights and adding the biases. I then reshape the output before returning it.
- Relu layer: I intialize all the output values to zero. I then use the np.maximum function on the input['data']. If the value is negative replace it with 0, if it is greater than or equal to zero the value is the input['data'].

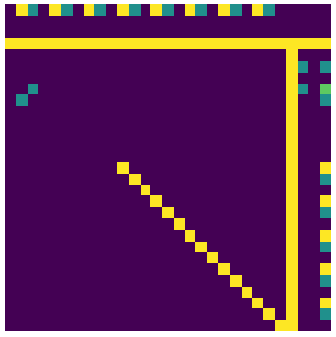
Image outputs saved in the results subfolder

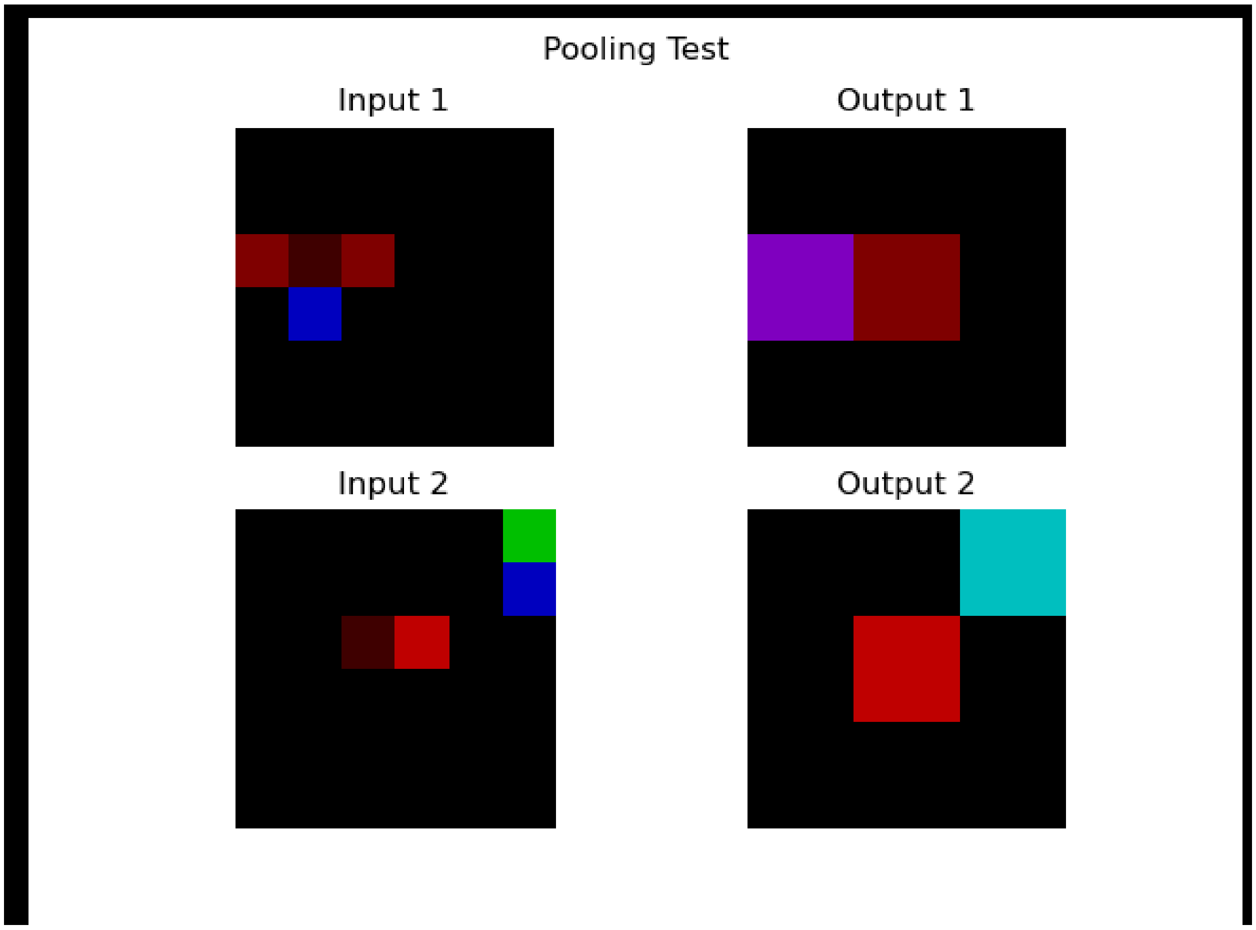
Inner Product Test

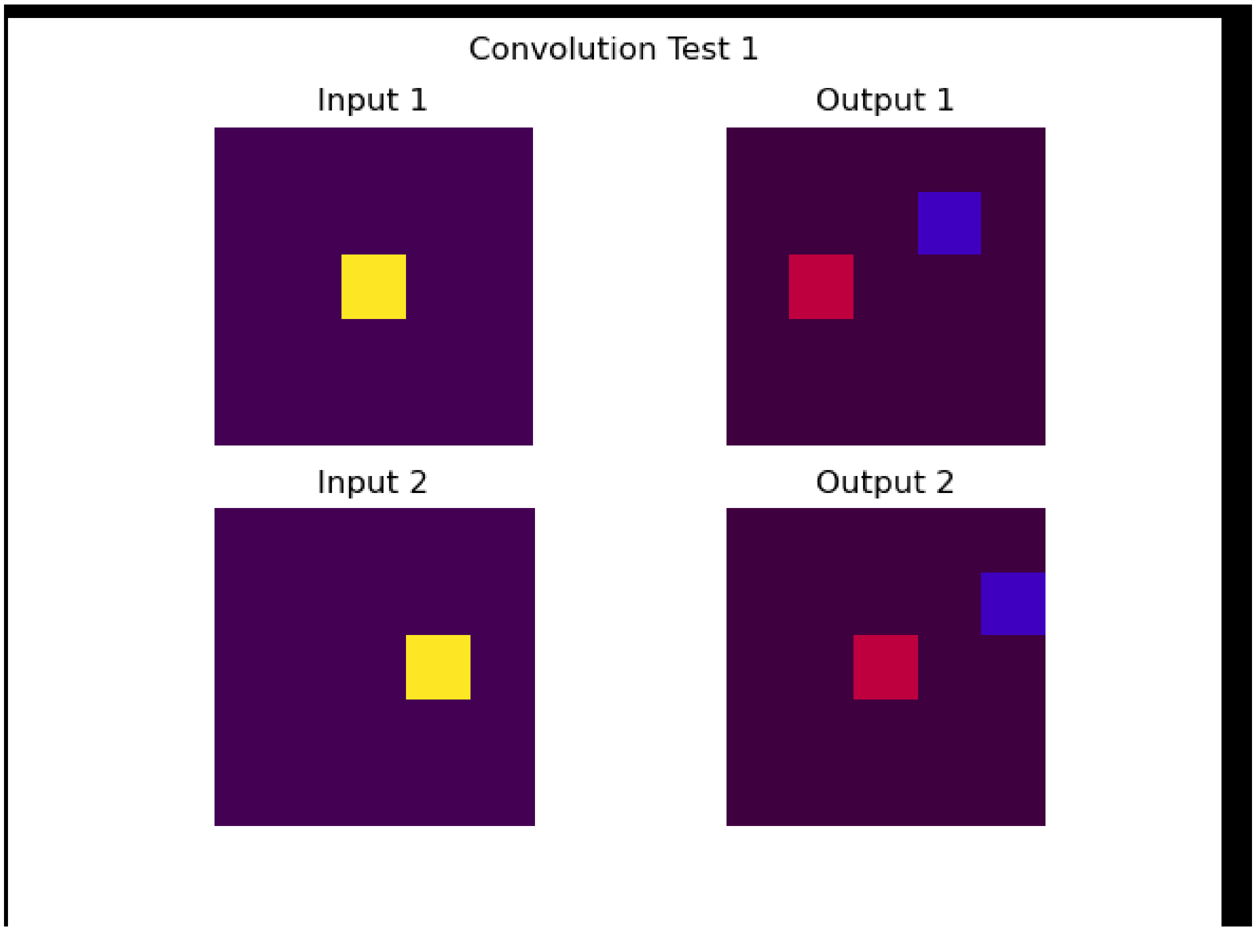
Batch 1

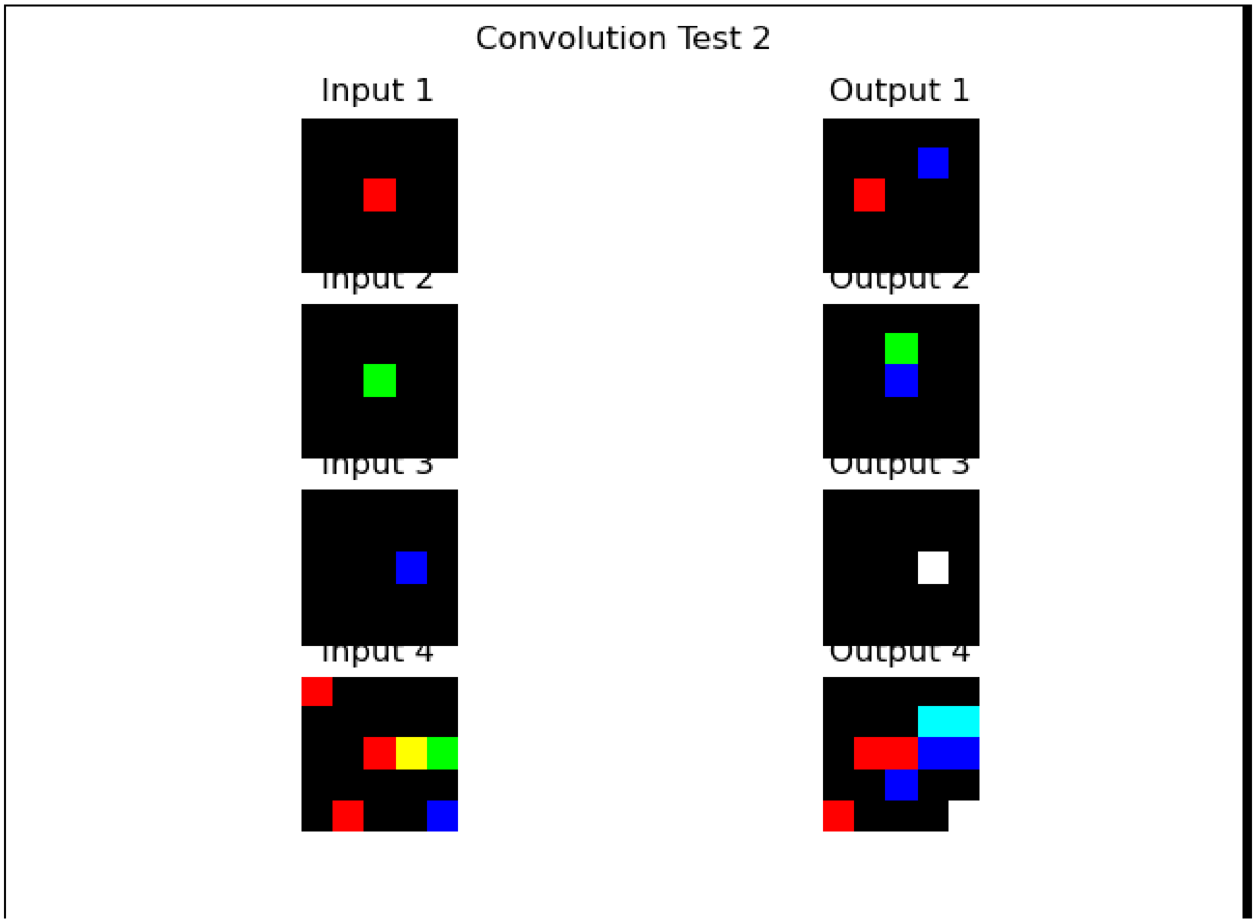


Batch 2




Pooling Test
Input 1
Output 1
Input 2
Output 2


Convolution Test 1
Input 1
Output 1
Input 2
Output 2

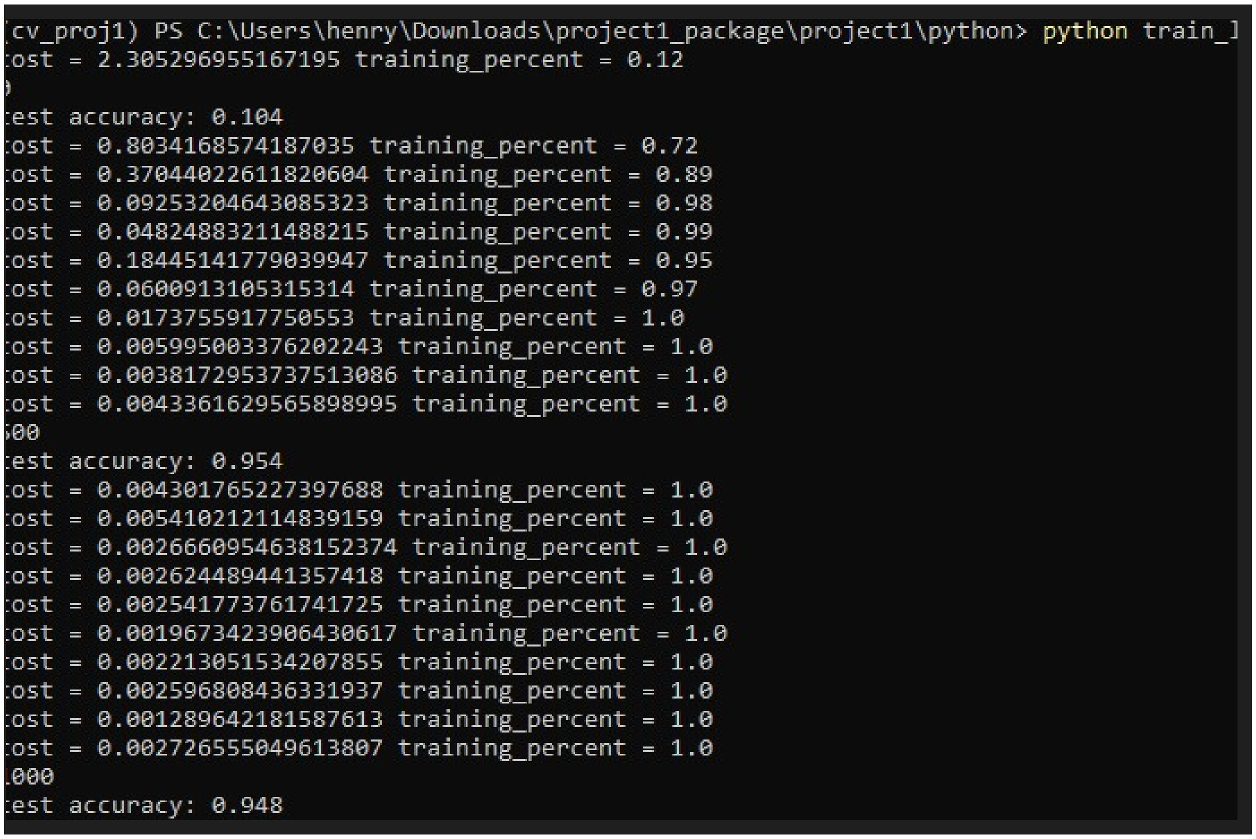


Part 2: back propagation

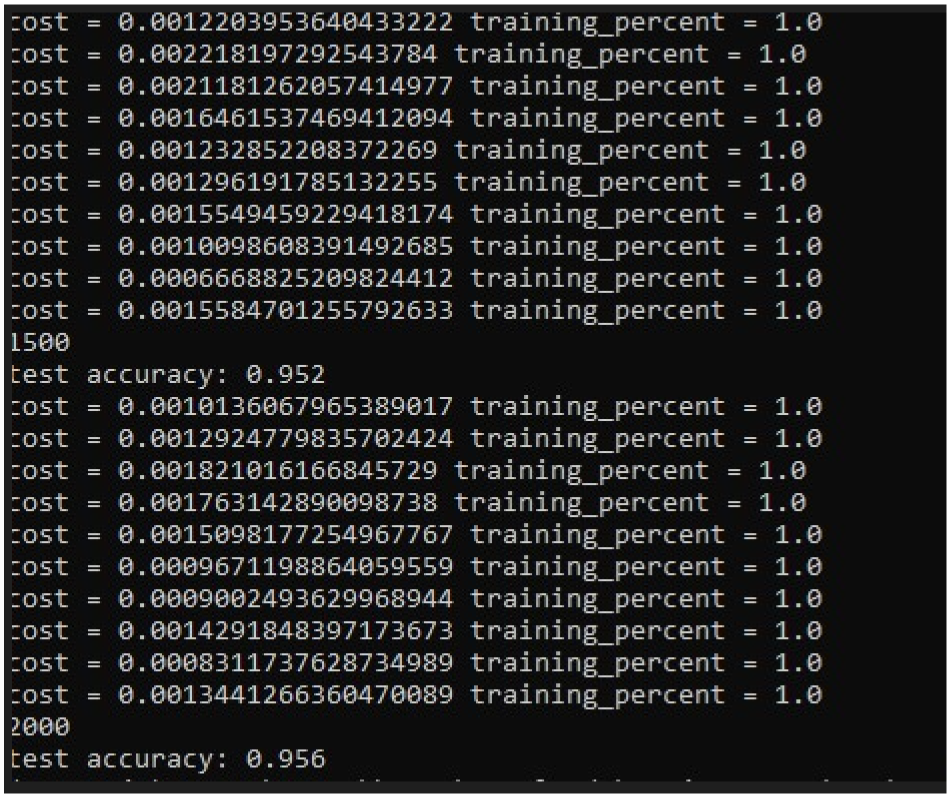
- Inner product: i computed the gradients for the layer's biases and weights by summing the output differences and doing matrix multiplication with the input data. The gradient with respect to input_data is calculated and stored in param_grad and input_od, which you can see in the code.
- Relu: I set elements in input_od to 1 if the input data is greater or equal to zero, since the derivative of a constant is just 1, otherwise it would be a 0. Then I multiply the output differences to get the final gradient values.

Part 3: Training

Ill just show the output of my training here in the form of images. My test accuracy jumped up quite quickly and remained around 95% through all the batches as you can see below. The lowest it ever being was 94.8% at n = 1000 and the highest being 95.6% at n = 2000



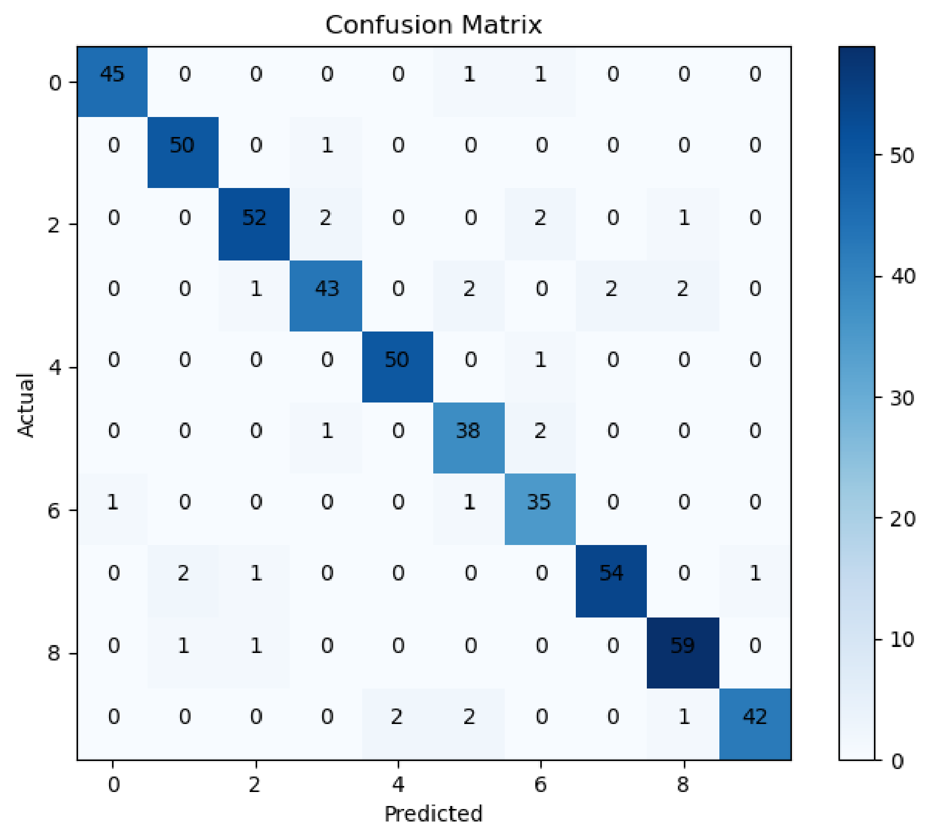
```
(cv_proj1) PS C:\Users\henry\Downloads\project1_package\project1\python> python train_l
cost = 2.305296955167195 training_percent = 0.12
)
test accuracy: 0.104
cost = 0.8034168574187035 training_percent = 0.72
cost = 0.37044022611820604 training_percent = 0.89
cost = 0.09253204643085323 training_percent = 0.98
cost = 0.04824883211488215 training_percent = 0.99
cost = 0.18445141779039947 training_percent = 0.95
cost = 0.0600913105315314 training_percent = 0.97
cost = 0.0173755917750553 training_percent = 1.0
cost = 0.005995003376202243 training_percent = 1.0
cost = 0.0038172953737513086 training_percent = 1.0
cost = 0.004336162956589895 training_percent = 1.0
500
test accuracy: 0.954
cost = 0.004301765227397688 training_percent = 1.0
cost = 0.00541021211483915 training_percent = 1.0
cost = 0.0026660954638152374 training_percent = 1.0
cost = 0.002624489441357418 training_percent = 1.0
cost = 0.002541773761741725 training_percent = 1.0
cost = 0.0019673423906430617 training_percent = 1.0
cost = 0.002213051534207855 training_percent = 1.0
cost = 0.002596808436331937 training_percent = 1.0
cost = 0.001289642181587613 training_percent = 1.0
cost = 0.002726555049613807 training_percent = 1.0
1000
test accuracy: 0.948
```



```
cost = 0.0012203953640433222 training_percent = 1.0
cost = 0.002218197292543784 training_percent = 1.0
cost = 0.0021181262057414977 training_percent = 1.0
cost = 0.0016461537469412094 training_percent = 1.0
cost = 0.001232852208372269 training_percent = 1.0
cost = 0.001296191785132255 training_percent = 1.0
cost = 0.0015549459229418174 training_percent = 1.0
cost = 0.0010098608391492685 training_percent = 1.0
cost = 0.0006668825209824412 training_percent = 1.0
cost = 0.0015584701255792633 training_percent = 1.0
1500
test accuracy: 0.952
cost = 0.0010136067965389017 training_percent = 1.0
cost = 0.0012924779835702424 training_percent = 1.0
cost = 0.001821016166845729 training_percent = 1.0
cost = 0.001763142890098738 training_percent = 1.0
cost = 0.0015098177254967767 training_percent = 1.0
cost = 0.0009671198864059559 training_percent = 1.0
cost = 0.0009002493629968944 training_percent = 1.0
cost = 0.0014291848397173673 training_percent = 1.0
cost = 0.0008311737628734989 training_percent = 1.0
cost = 0.0013441266360470089 training_percent = 1.0
2000
test accuracy: 0.956
```

Confusion Matrix:

Image saved in results folder named confusion.png

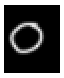


Based on both testing and the confusion matrix. My CNN often mistook 5's for 9's and 6's for 0's. This makes sense in the sense for 5/9 the bottom have of both digits are fairy similar with it's half loop, and 6/0 makes sense in the sense that these are the only digits with one circle in them. Perhaps more training could fix this issue.

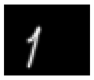
Real world testing:

Images located in myimages folder (sizing here is slightly off to what it actually is since im using snip and sketch)

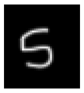
0.png:



1.png:

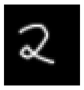


2.png:



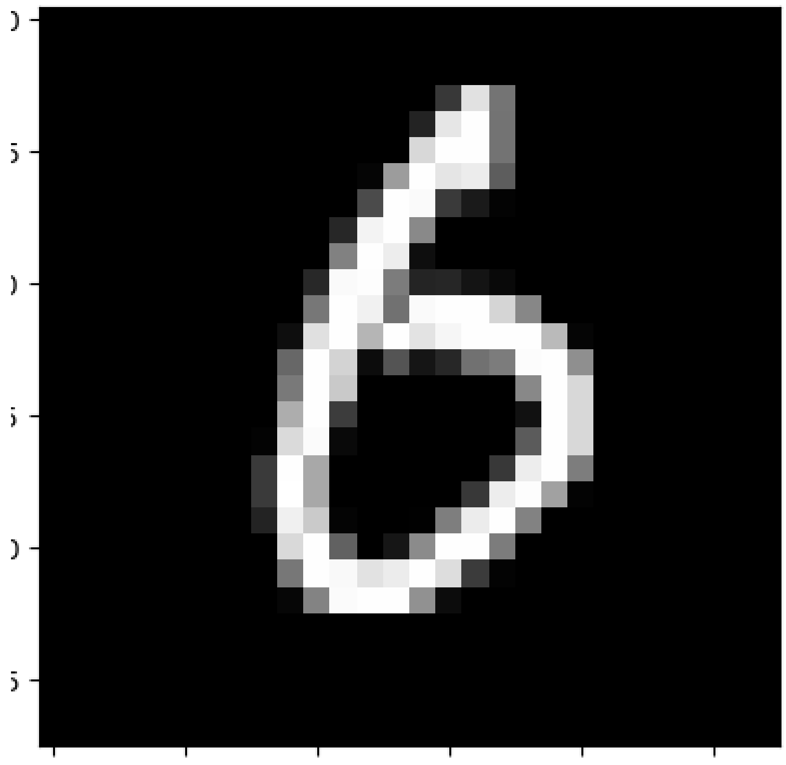
3.png:

4.png:

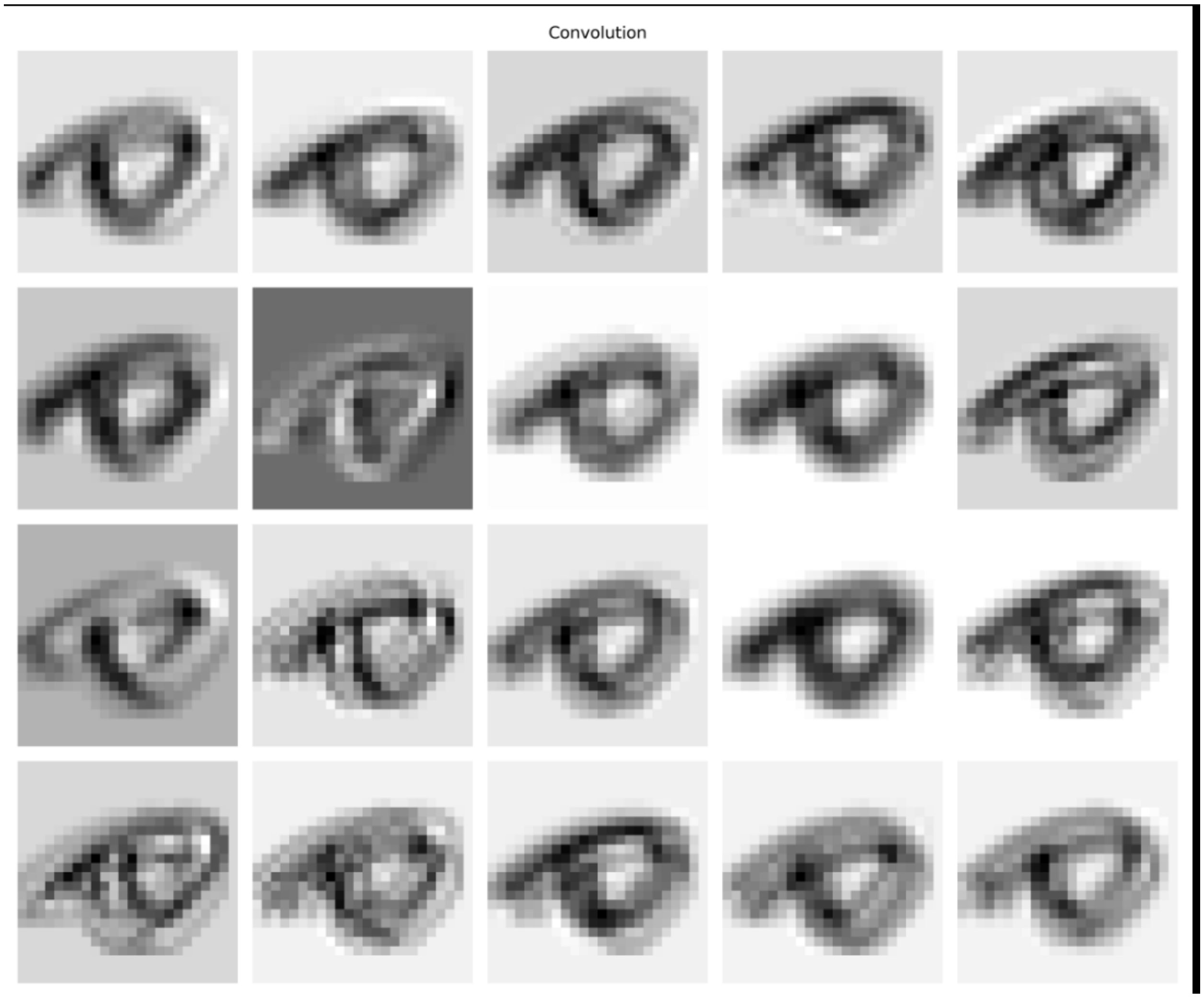


Note: I originally had each image match it’s file name ex. 2.png should contain an image of 2, but I swapped it around here to confirm my results
Results: [array([0, 3, 2, 3, 2], dtype=int64)] these are the printed results you can get from running python mytester.py. Here is guesses:
0.png correctly to be 0
1.png incorrectly to be 3
2.png incorrectly to be 2
3 png correctly to be 3
4.png correctly to be 2
Total accuracy of : 3/5

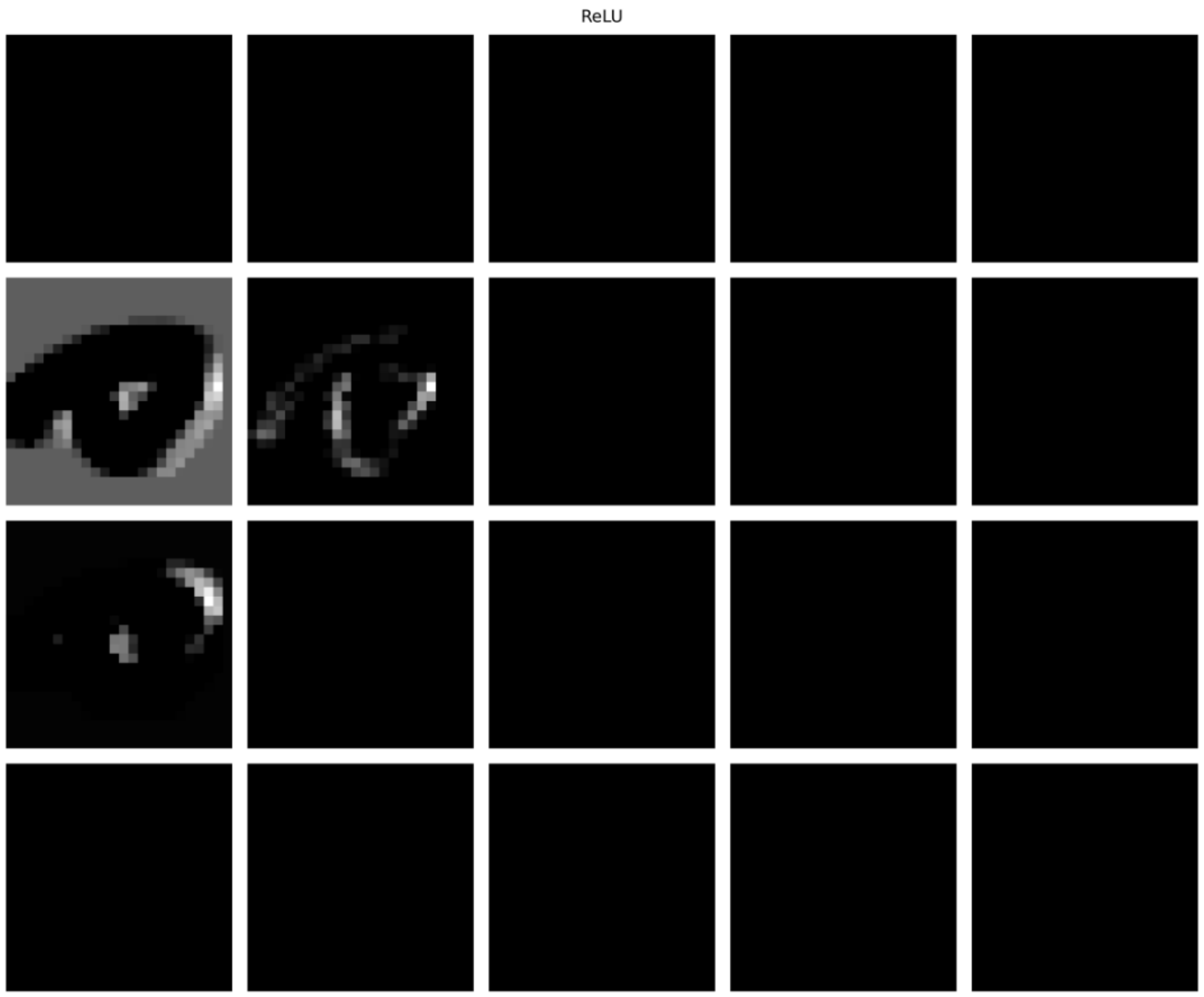
Part 4: Visualization
Images in this part can be found saved in results folder. (named relu.png and convolution.png)
convolution feature image -> the image here is of a 6. Changes everytime you run python vis_data.py
Input image:



Convultion.png:



Relu.png:



Both outputs of the convolution and relu make sense relative to the input image. In the convolution layer, the network begins to recongnize edges of our digit 6 and begins to outline the entire digit. We can attribute this to the filters we use in the convolution layer. The relu output is mostly almost all black, this is because the black pixels remain untouched in a relu while the white pixels get set to zero (This was mentioned in class).

Part 5: Image classification
Image1 score = 4/10
Image2 score = 3/10
Image3 score = 4/5
Image4 score = 19/50
Total score = 30/75, giving us a total accuracy of 40%
Note: these images are not saved in results for you to see them just run python ec.py and plt.show() will show you the images. Just close the current image to go to the next.
Image 1 bounding boxes:

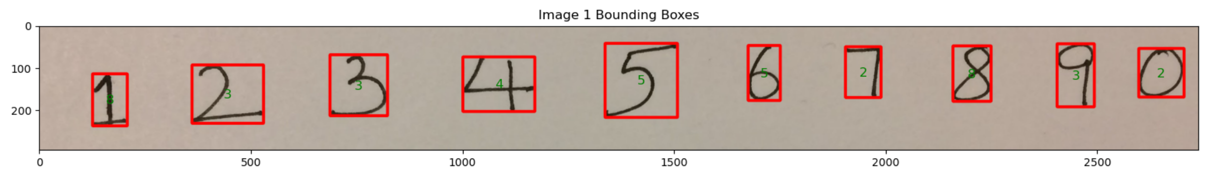


Image 2 bounding boxes:

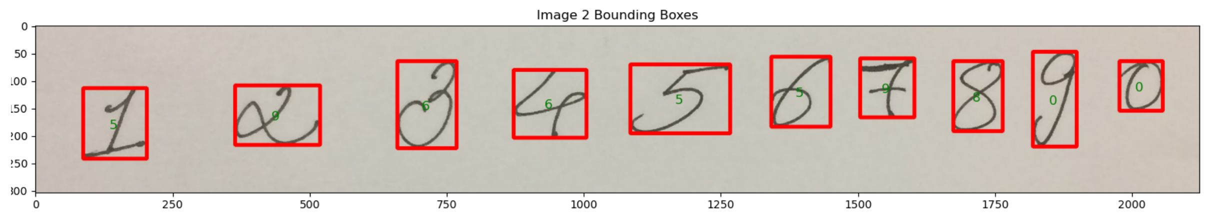


Image 3 bounding boxes:

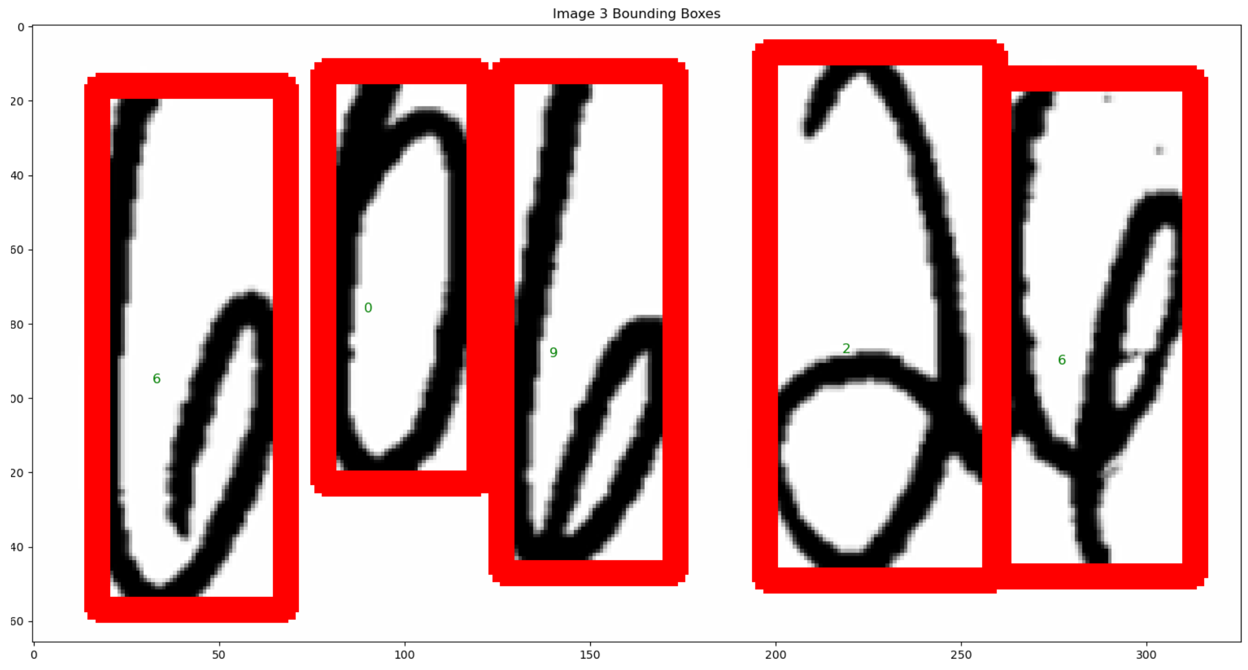
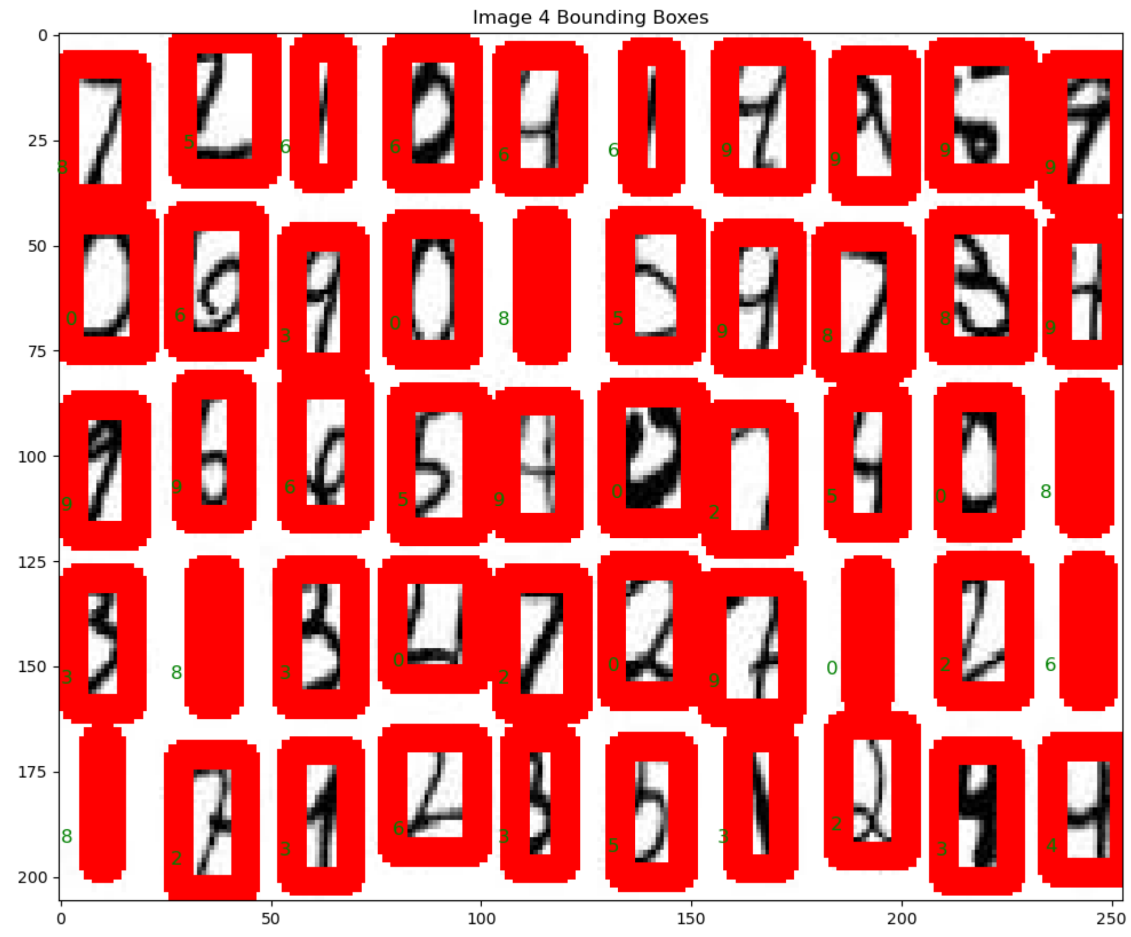


Image 4 bounding boxes:



The threshold I used worked quite well for image 1 and 2 but not quite as well for 3 and 4. I labelled each bounding box with the prediction the network made for each of those digits. (usually located on the left side of each bounding box).