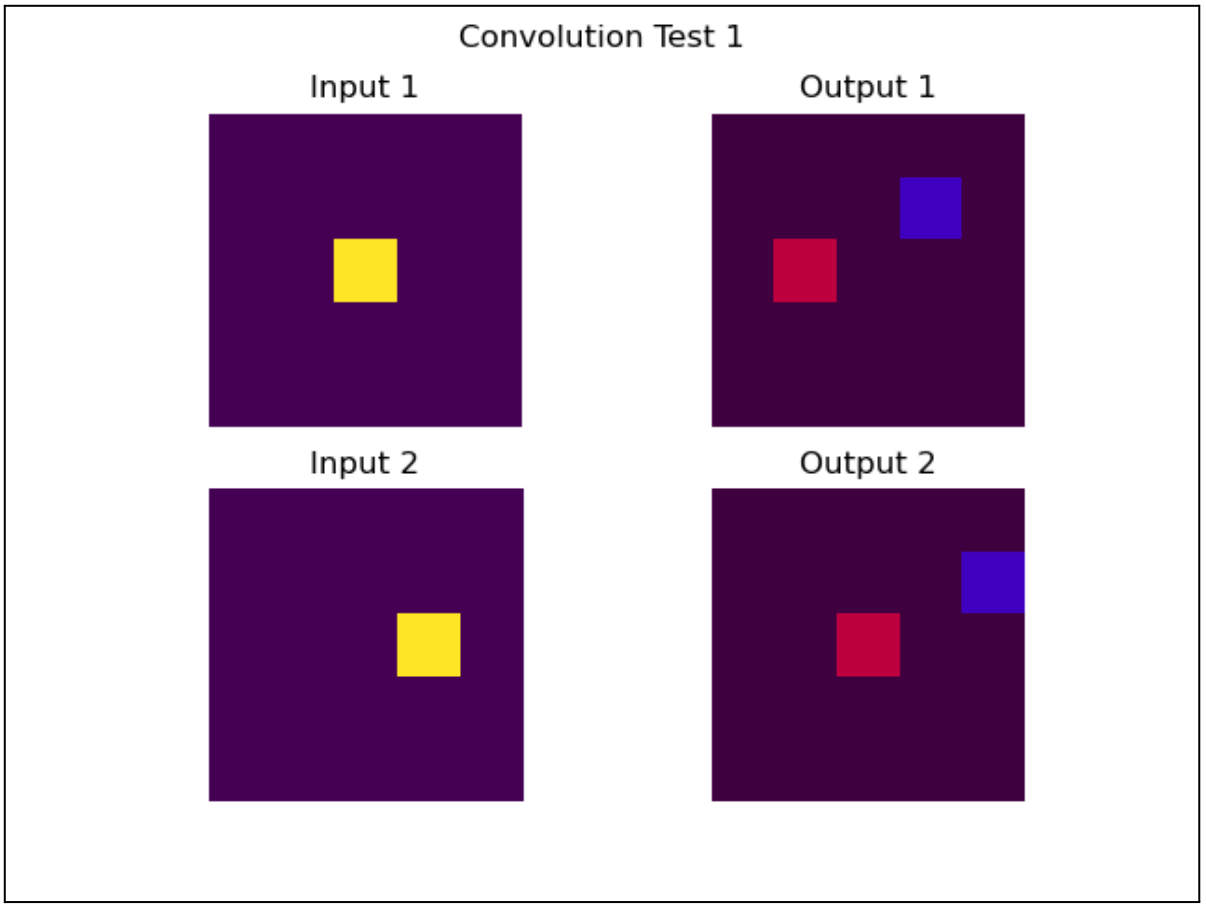
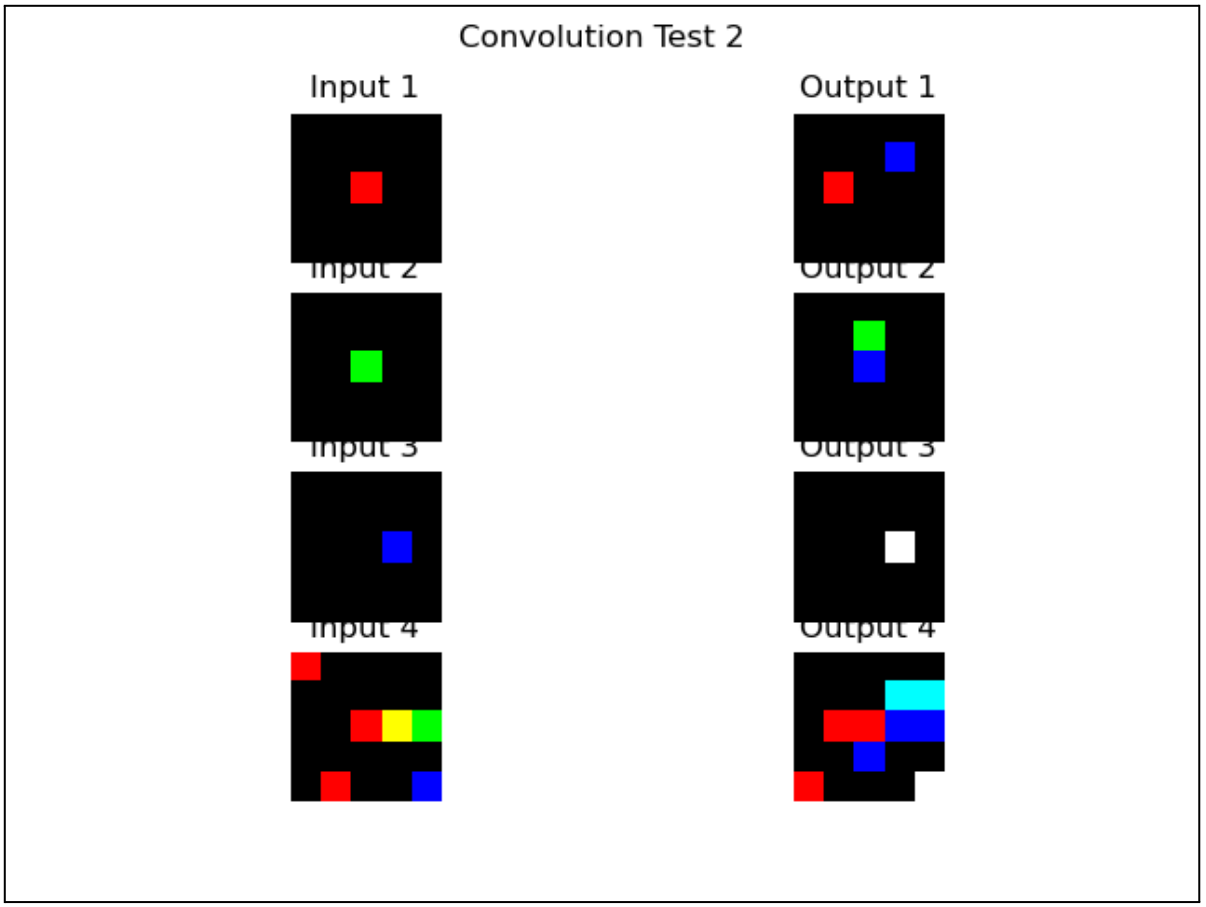
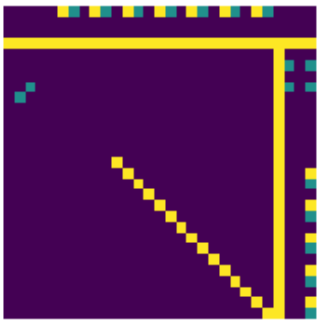

Convolution Test 1
Input 1
Output 1
Input 2
Output 2

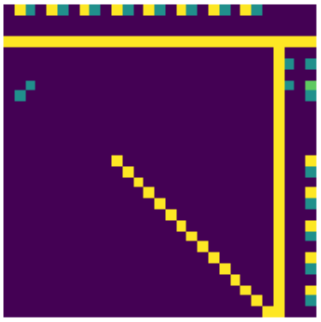

Convolution Test 2
Input 1
Output 1
Input 2
Output 2
Input 3
Output 3
Input 4
Output 4

Inner Product Test

Batch 1

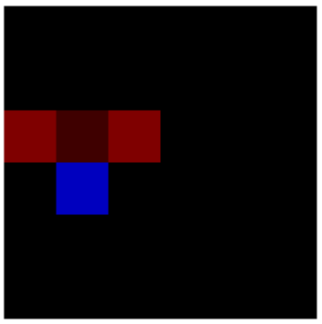


Batch 2

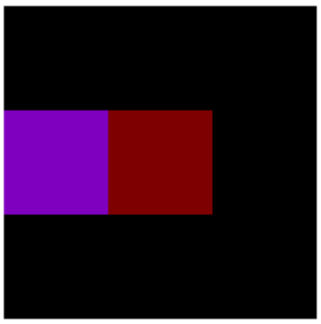


Pooling Test

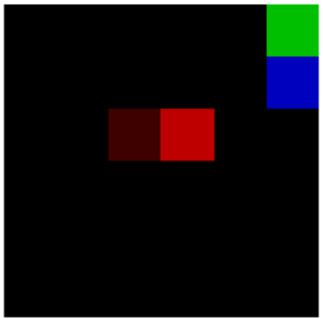
Input 1



Output 1



Input 2



Output 2

Q2
No report requirements in the assignment.

Q3.1
```
(cv_proj1) C:\Users\jared\OneDrive\SFU\SFU_Semester_7\CMPT412\project1_package\project1\python> python train_lenet.py
cost = 2.305296955167195 training_percent = 0.12
0
test accuracy: 0.104
cost = 0.8034168574187707 training_percent = 0.72
cost = 0.37044022611820787 training_percent = 0.89
cost = 0.09253204643085317 training_percent = 0.98
cost = 0.04824883211488215 training_percent = 0.99
cost = 0.18445141779039997 training_percent = 0.95
cost = 0.06009131053153044 training_percent = 0.97
cost = 0.017375591775055002 training_percent = 1.0
cost = 0.005995003376202278 training_percent = 1.0
cost = 0.0038172953737512986 training_percent = 1.0
cost = 0.004336162956589933 training_percent = 1.0
500
test accuracy: 0.954
cost = 0.004301765227397694 training_percent = 1.0
cost = 0.00541021211483918 training_percent = 1.0
cost = 0.0026660954638152057 training_percent = 1.0
cost = 0.002624489441357404 training_percent = 1.0
cost = 0.002541773761741736 training_percent = 1.0
cost = 0.0019673423906430695 training_percent = 1.0
cost = 0.0022130515342078548 training_percent = 1.0
cost = 0.002596808436331933 training_percent = 1.0
cost = 0.0012896421815876199 training_percent = 1.0
cost = 0.0027265550496137964 training_percent = 1.0
1000
test accuracy: 0.948
cost = 0.0012203953640433276 training_percent = 1.0
cost = 0.0022181972925437933 training_percent = 1.0
cost = 0.0021181262057415025 training_percent = 1.0
cost = 0.0016461537469412118 training_percent = 1.0
cost = 0.0012328522083722602 training_percent = 1.0
cost = 0.0012961917851322378 training_percent = 1.0
cost = 0.0015549459229418153 training_percent = 1.0
cost = 0.001009860839149272 training_percent = 1.0
cost = 0.0006668825209824389 training_percent = 1.0
cost = 0.0015584701255792655 training_percent = 1.0
1500
test accuracy: 0.952
cost = 0.0010136067965389108 training_percent = 1.0
cost = 0.0012924779835702333 training_percent = 1.0
cost = 0.0018210161668457334 training_percent = 1.0
cost = 0.0017631428900987403 training_percent = 1.0
cost = 0.0015098177254967856 training_percent = 1.0
cost = 0.0009671198864059536 training_percent = 1.0
cost = 0.0009002493629968988 training_percent = 1.0
cost = 0.0014291848397173608 training_percent = 1.0
cost = 0.0008311737628735089 training_percent = 1.0

(cv_proj1) C:\Users\jared\OneDrive\SFU\SFU_Semester_7\CMPT412\project1_package\project1\python>
```

Test accuracy = 0.952 = 95.2%

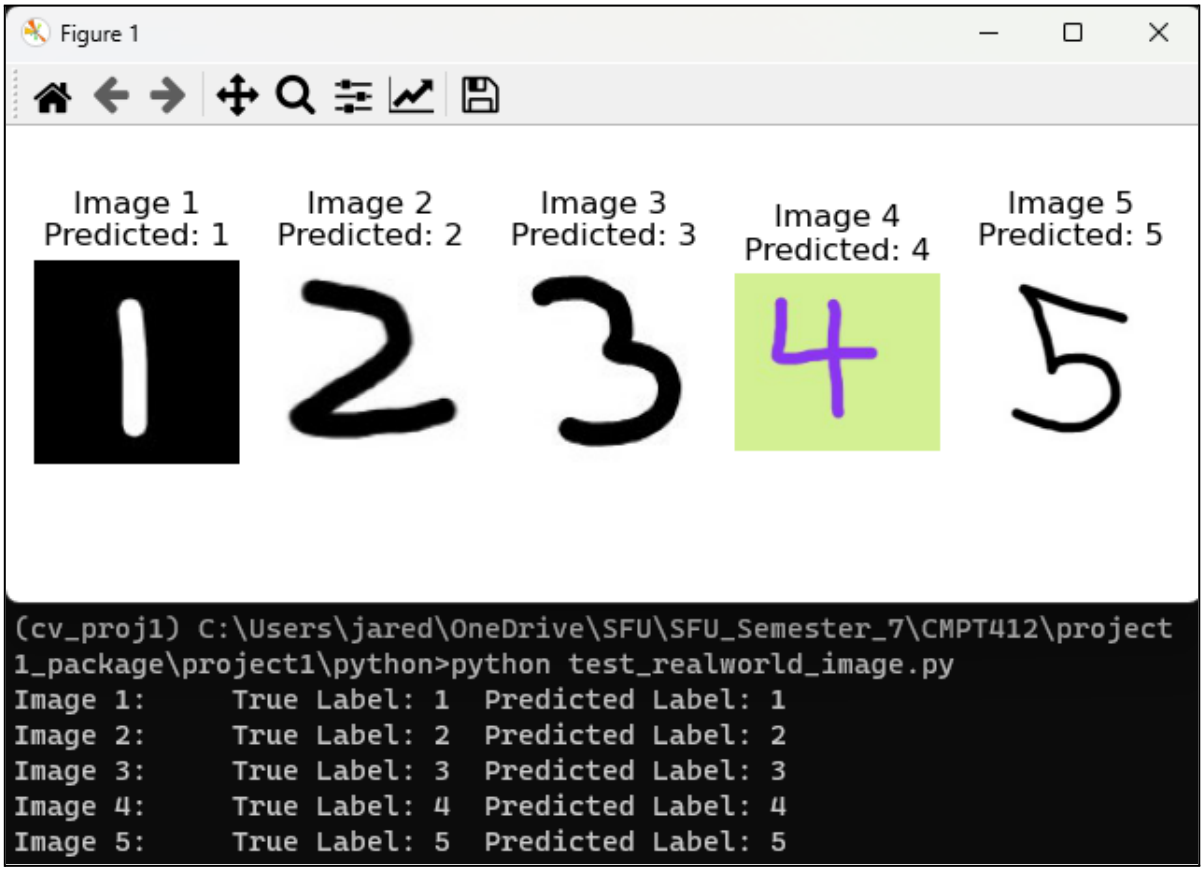
Q3.2

```
(cv_proj1) C:\Users\jared\OneDrive\SFU\SFU_Semester
_7\CMPT412\project1_package\project1\python>python
test_network.py
Confusion Matrix:
 [[57  0  0  0  0  0  1  0  0  0]
 [ 1 71  0  1  0  0  0  0  0  0]
 [ 0  1 40  0  0  0  0  1  1  0]
 [ 0  0  1 44  0  0  0  1  0  0]
 [ 0  0  0  0 39  0  0  0  0  5]
 [ 0  0  0  2  1 38  1  0  0  0]
 [ 2  0  0  0  1  0 43  0  0  0]
 [ 0  0  2  2  0  0  0 44  1  1]
 [ 1  0  0  1  0  0  0  0 41  0]
 [ 0  0  0  2  0  1  0  0  0 52]]
Top two confused pairs:
1. Class 4 and class 9 have been confused 5 times.
2. Class 0 and class 6 have been confused 3 times.
```

Class 4 and and class 9 are confusing because the 4 symbol resembles the 9 symbol significantly. A 4 is just a 9 with rounded edges. Also, the 6 class strongly resembles the 0 class because a 6 is just a 0 with a stick coming out, and sometimes zeros may have protruding sticks as well due to messy handwriting if they are considered insignificant in size.

Q3.3

I created the analyzed images by screenshotting the doodles from an online sketchpad (https://sketch.io/sketchpad/). I used various colors and pencil widths to show the adaptability of my network. All the preprocessing I did, aside from the original casual free-form rectangular crop, occurred in my code. My code for this step is in test_realworld_image.py.



Q4.1

Below is the output from vis_data.py

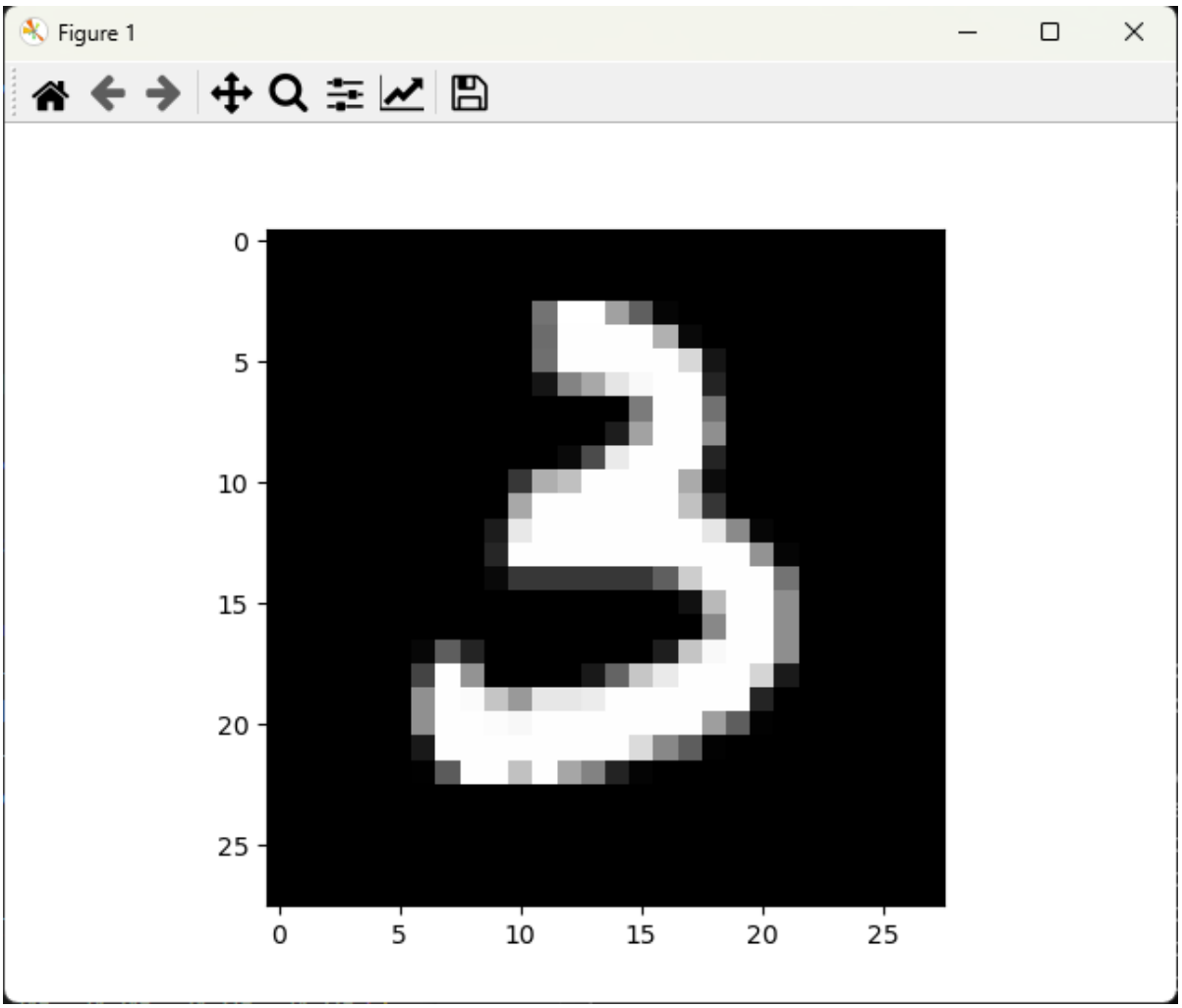


Figure 1
0
5
10
15
20
25
0
5
10
15
20
25

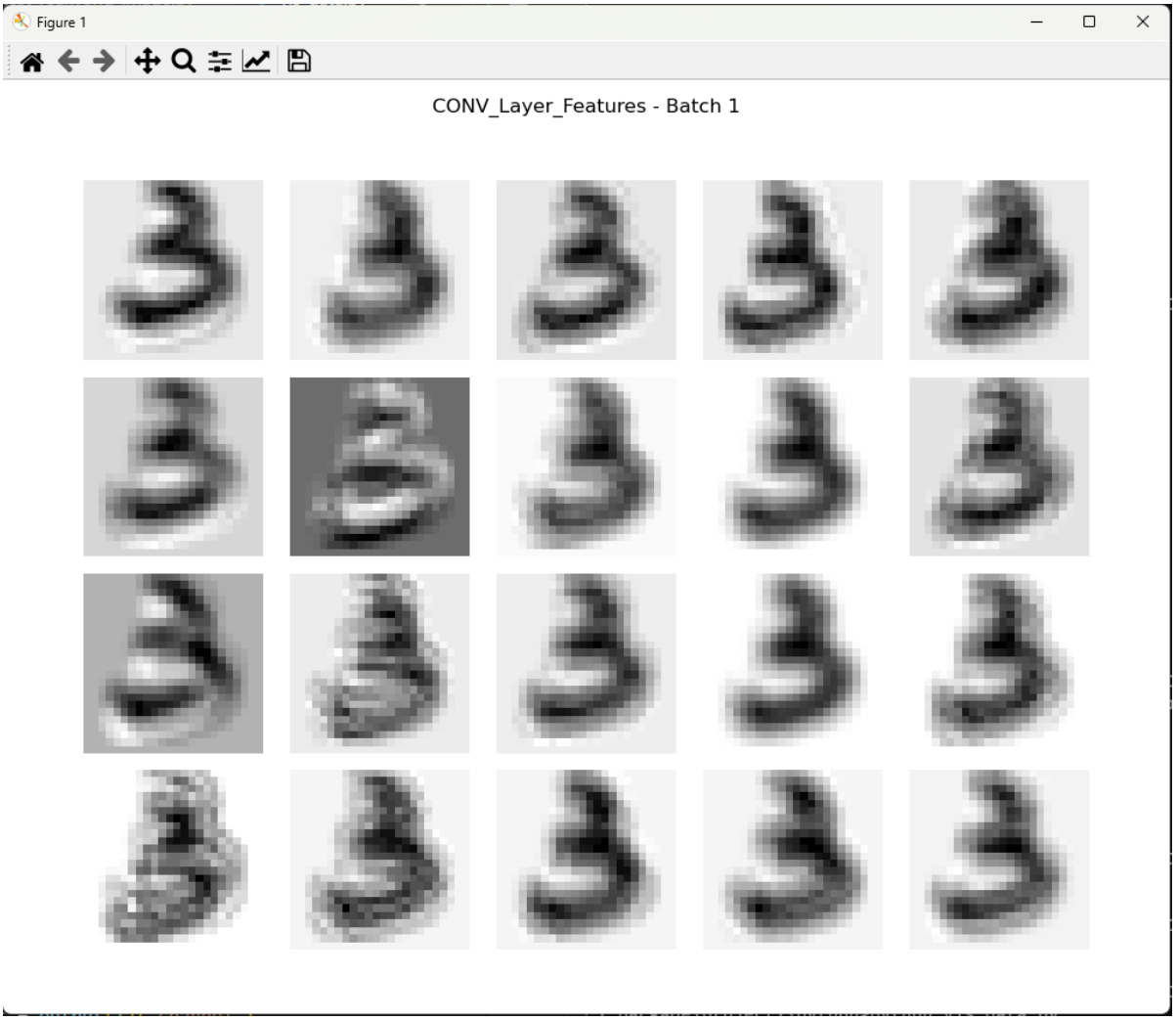
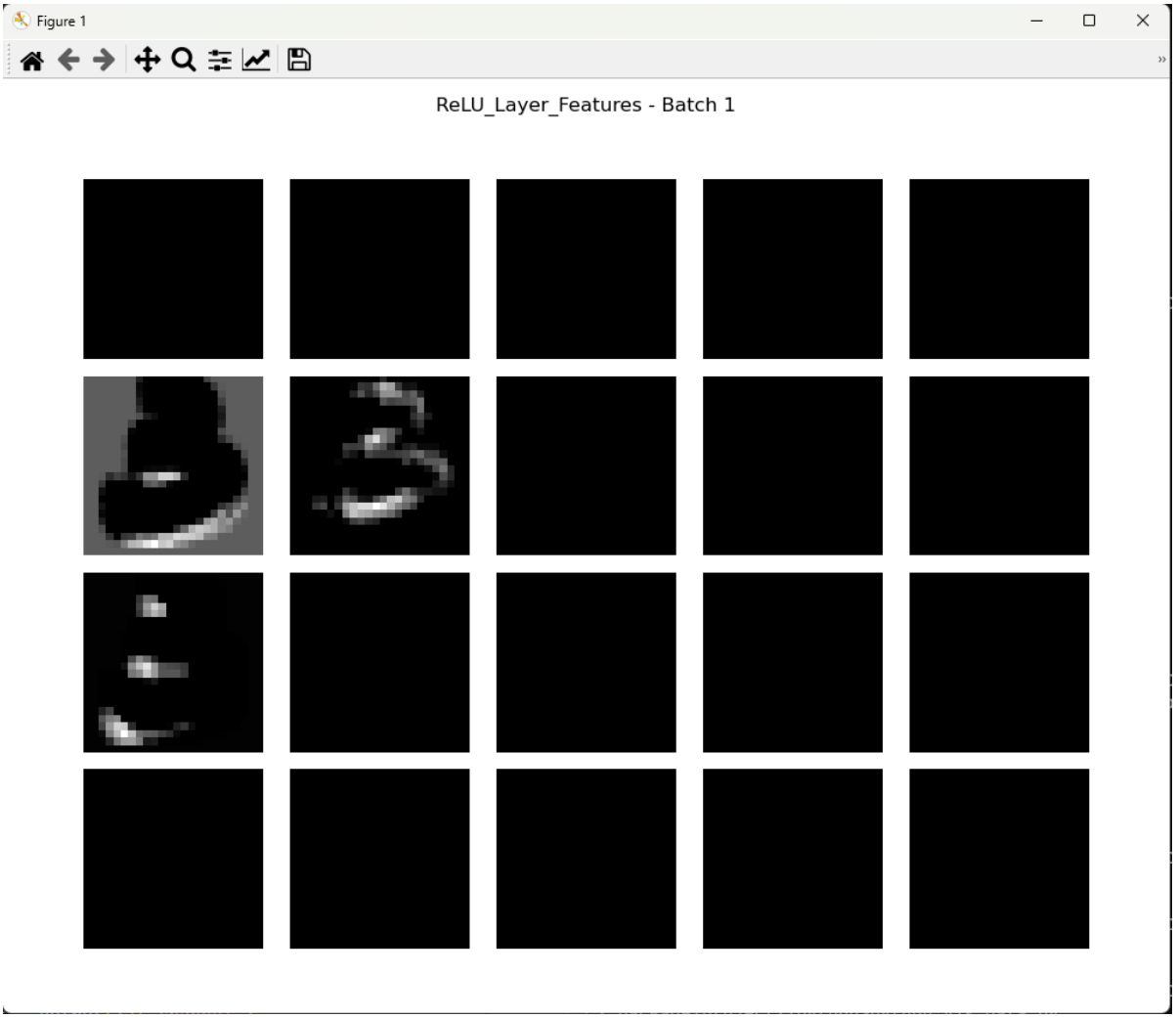


Figure 1
CONV_Layer_Features - Batch 1



Q4.2

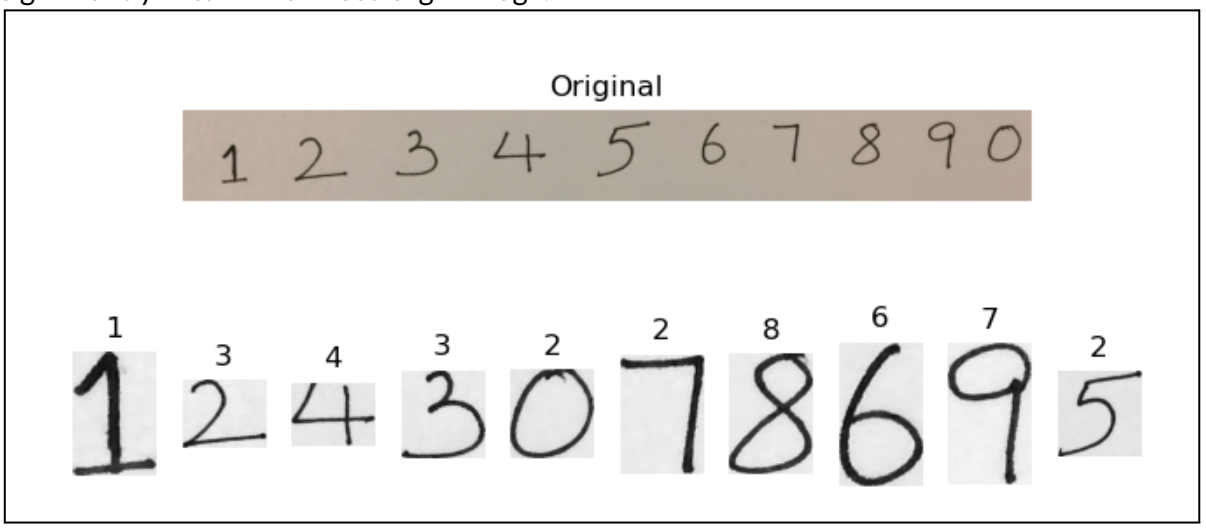
Original Image
It's a normalized grayscale 28x28 image with a black background and a handwritten letter as the white foreground.

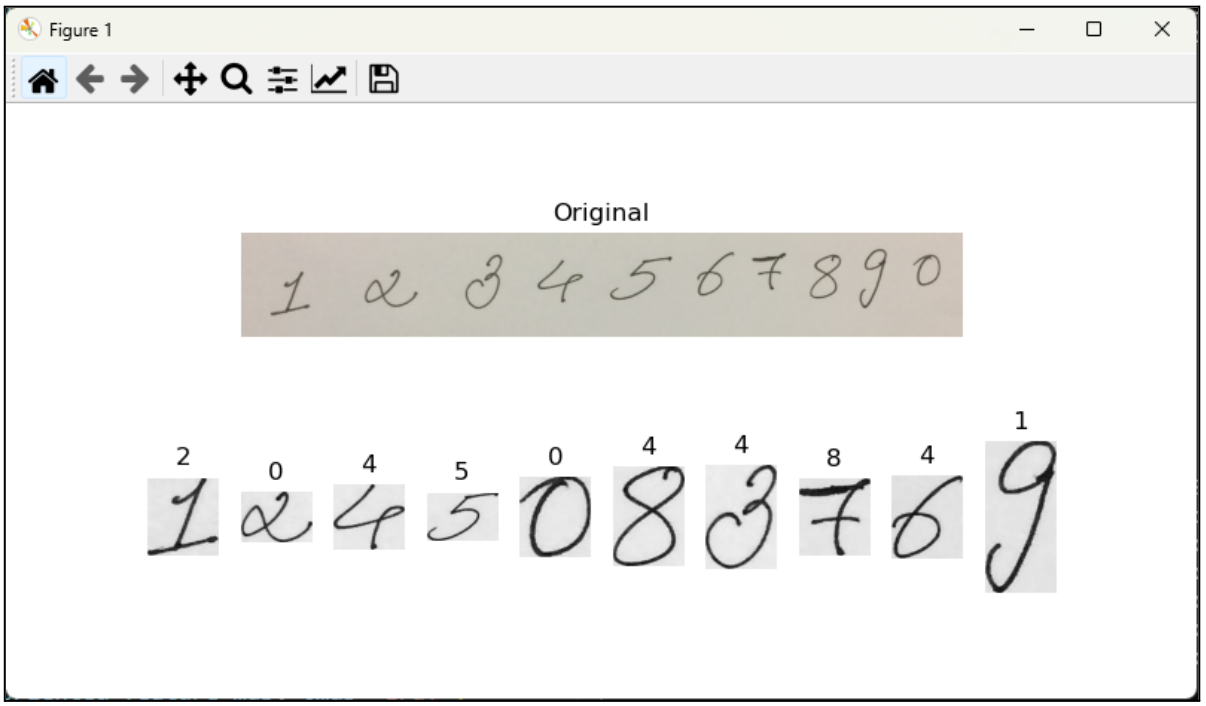
Convolutional Layer Feature Maps - First Layer
These feature maps seem to be focused on edge and corner detection, which, according to the lecture, is more common in the initial layers. The edge detection of some of these feature maps seem very much like directional edges, similar to the ones you might find by hardcoding a sobel matrix convolution.

ReLU Layer Feature Maps - Second Layer
This layer hides unimportant features by setting negative values to zero. By hiding less important values, the feature map draws more attention to significantly more important values. This “hiddenness” causes the feature maps to have much more blackness present. Due to this being a deeper layer, it is less defined and more abstract than the first layer and the original image.

Q5

For me, ec.py plots the original image, with the extracted digits beneath it, and the predictions for each extracted digit typed above an image of the digit itself. The network performed significantly better on the last larger image.





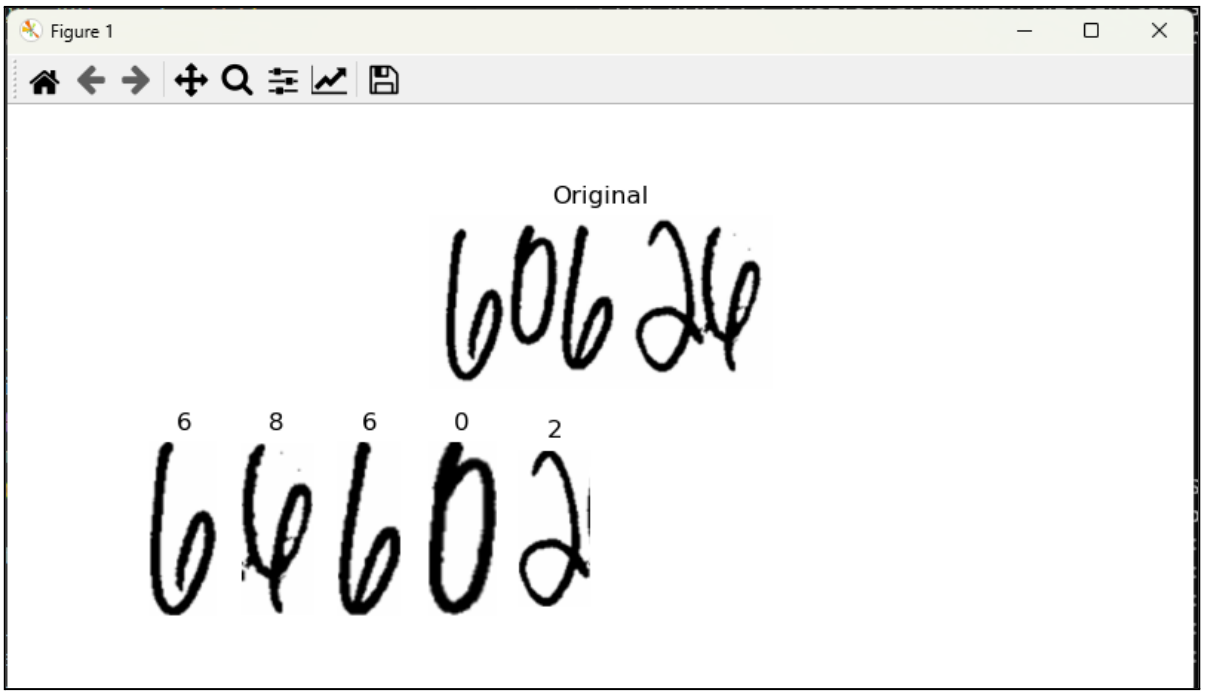
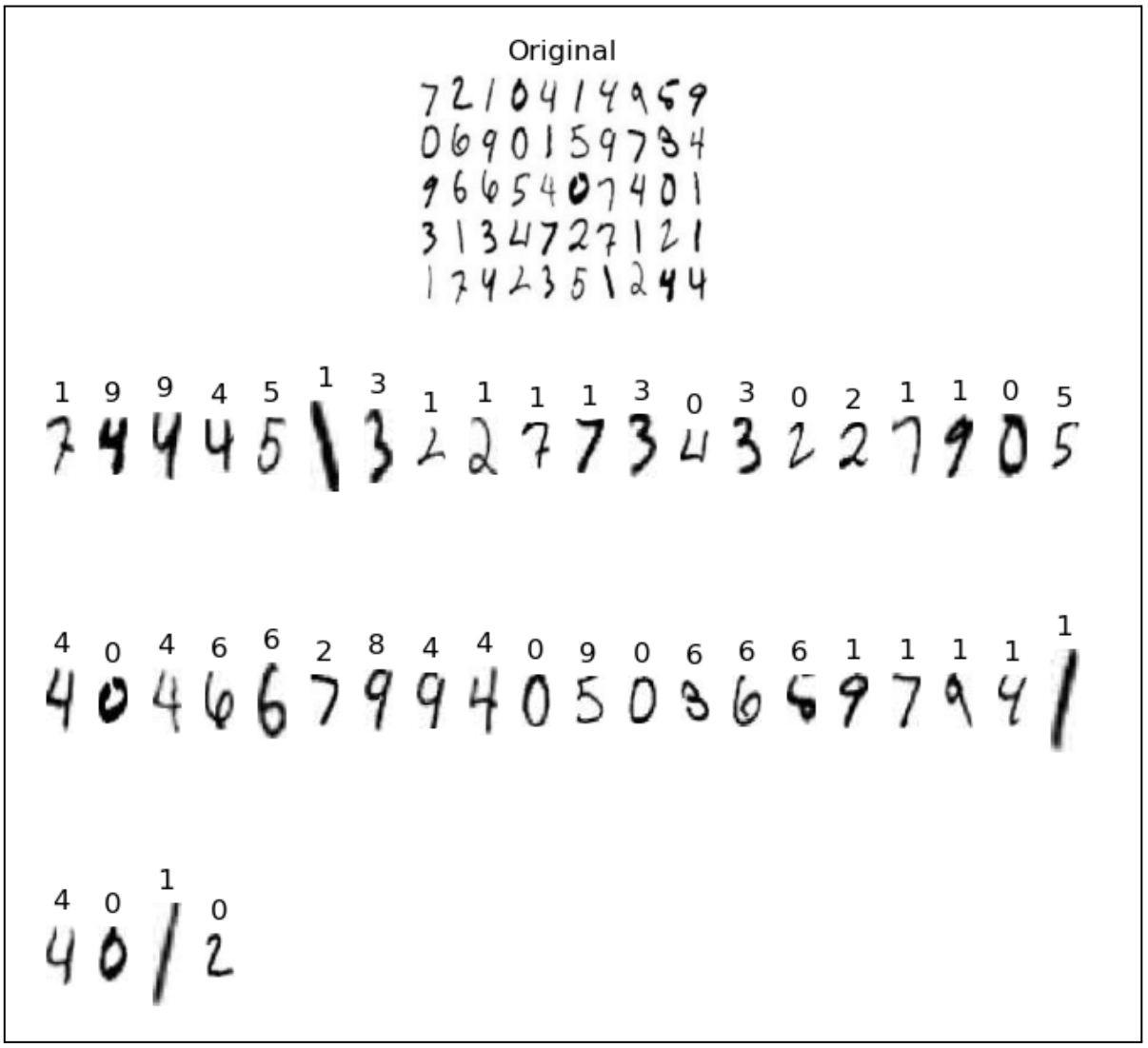


Figure 1
Original
6 8 6 0 2


Original
1 9 9 4 5 1 3 1 1 1 1 3 0 3 0 2 1 1 0 5
4 0 4 6 6 2 8 4 4 0 9 0 6 6 6 1 1 1 1 1
4 0 1 0