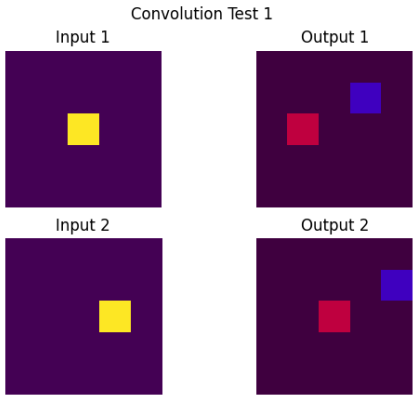
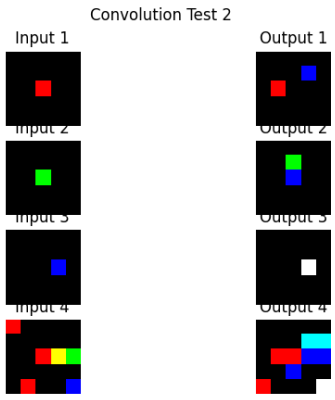
Project 1
Digit recognition with convolutional neural networks

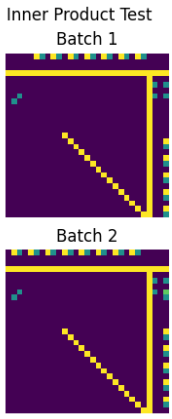
Name: Ruixuan Liu
Student ID: 301383988
Lateday use: 1day

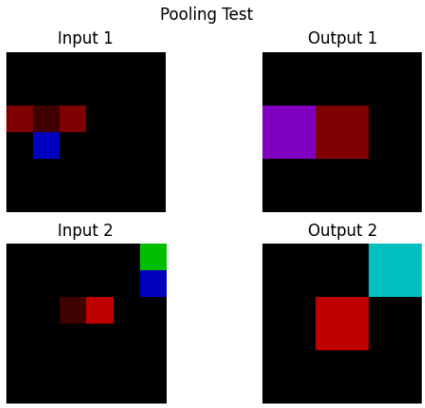
Part 1: Forward Pass

1. Inner Product: This function is doing dot product of W and x and add the bias. Which W is param["w"]. T, x is input['data'] and bias is param["b"].T.
2. Max Pooling: Moments with large amounts of data are reduced by filters to easily manipulated matrices. For example, here only the largest values in the feature map are selected. So, in this 4D matrix, only use the larger value to be selected.
3. Convolution Layer: By using *im2col_conv_batch* function, we get a 3D output for (k*k*c, h_out*w_out, batch_size). I looped from batch size, and for each batch, I multiply W.T with the first two dimensional matrices of X for each batch and add bias. finally transpose result and reshape it into 2D.
4. Relu Layer: Simply set the pixel values of images to: if > 0 to itself, if < 0 to 0.









Part 2: Backward Pass

1. Relu Layer: For max = if x > 0 then = x, if x <= 0 then = 0 perform the derivation. We get if x > 0 then = 1, if x <= 0 then = 0. So we just output the derivative of the previous level multiplied by the derivative of this level.

2. Inner Product Layer:

$$\frac{\partial l}{\partial w_i} = \frac{\partial l}{\partial h_i}\frac{\partial h_i}{\partial w_i}$$

$$\frac{\partial l}{\partial h_{i-1}} = \frac{\partial l}{\partial h_i}\frac{\partial h_i}{\partial h_{i-1}}$$

The two formulas are used to find the storage location represented by each value. dl/dh are stored in output['diff'], and dh/dw are input['data'], which are dot-multiplied to get param_grad['w']. The weighting of dl/dh is bias. dot-multiplication of param['w'] and dl/dh gives gradient of the loss w.r.t the input layer.

Part 3: Training

1.

```
test accuracy: 0.952
cost = 0.0010136067965389062 training_percent = 1.0
cost = 0.0012924779835702387 training_percent = 1.0
cost = 0.0018210161668457286 training_percent = 1.0
cost = 0.001763142890098738 training_percent = 1.0
cost = 0.0015098177254967765 training_percent = 1.0
cost = 0.0009671198864059536 training_percent = 1.0
cost = 0.0009002493629968988 training_percent = 1.0
cost = 0.0014291848397173673 training_percent = 1.0
cost = 0.0008311737628735011 training_percent = 1.0
cost = 0.001344126636047 training_percent = 1.0
2000
test accuracy: 0.956
```

2.

```
(cv_proj1) sakura@Sakuras-MacBook-Pro python % python3 test_network.py
[[58.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0. 57.  1.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0. 51.  0.  1.  0.  1.  1.  0.  0.]
 [ 0.  0.  1. 46.  0.  0.  0.  1.  0.  1.]
 [ 0.  0.  1.  0. 42.  0.  2.  0.  0.  2.]
 [ 3.  0.  0.  2.  0. 49.  0.  0.  1.  0.]
 [ 4.  0.  0.  0.  0.  0. 46.  0.  0.  0.]
 [ 0.  0.  1.  0.  0.  1.  0. 42.  0.  0.]
 [ 1.  0.  0.  0.  1.  0.  1.  0. 42.  1.]
 [ 0.  1.  0.  1.  1.  1.  0.  0.  0. 35.]]
```

For my confusion matrix, the most difficult to recognize is 9, followed by 4, 7, and 8. I think the network is mainly trained to recognize the shape of the image, so it makes it difficult to recognize images that are almost SHAPE.

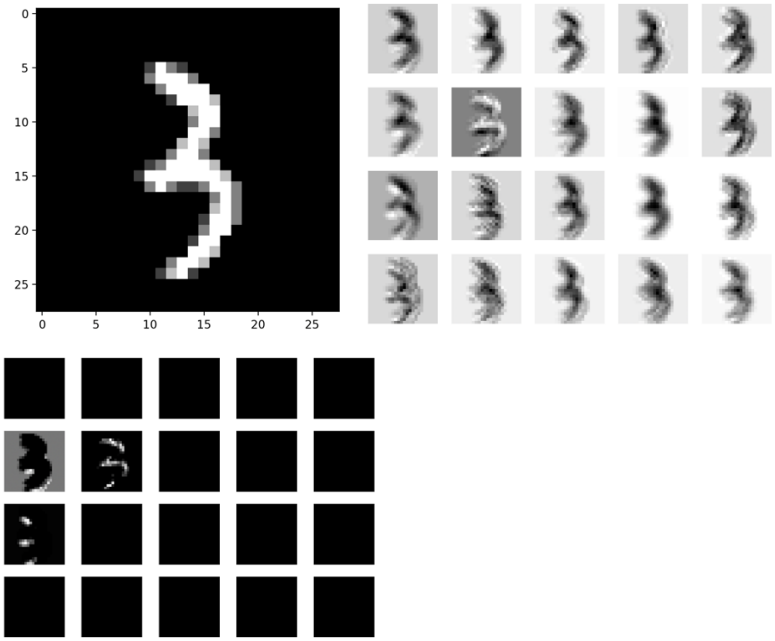
3.

```
(cv_proj1) sakura@Sakuras-MacBook-Pro python % python3 test_digit.py
actual: 1 predict 9
actual: 5 predict 3
actual: 8 predict 9
actual: 9 predict 3
actual: 3 predict 3
```

I use these 5 samples to recognized and predict the value. But get very low accuracy, only have 1 correct and 4 incorrect. As I explained with the previous question, the model does not recognize 3, 5, 8, and 9 well.

Part 4: Visualization

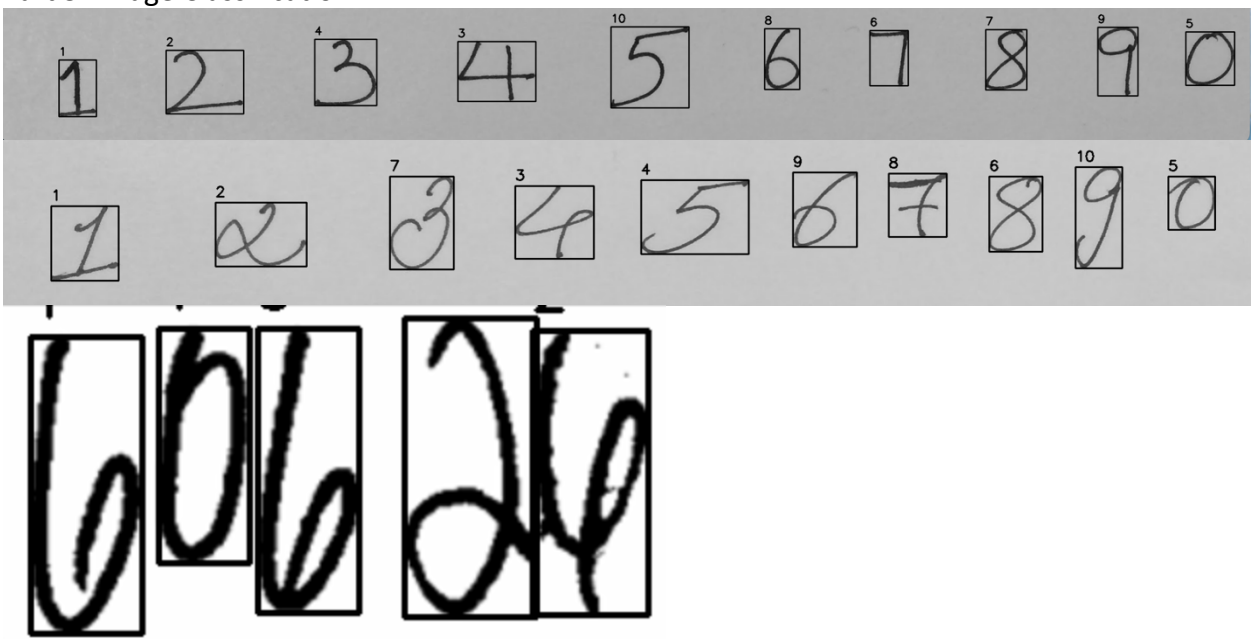
1.

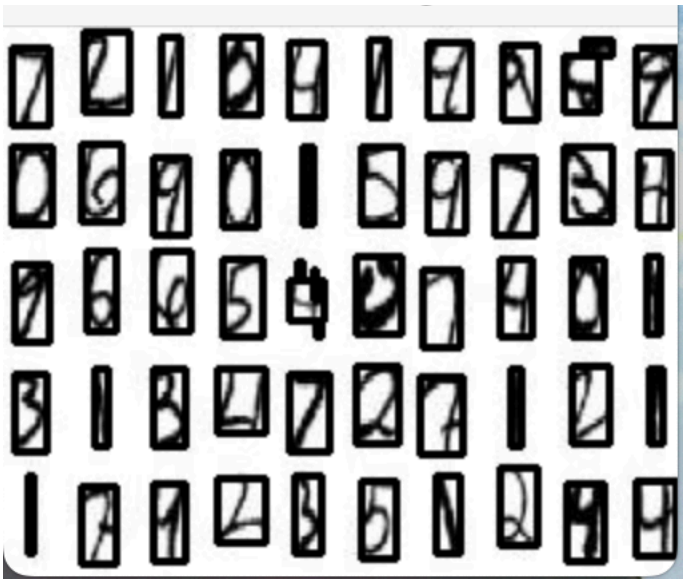


The first image is the original image of grayscale type, the second image is the visualization of convolution layer, and the last image is the visualization of relu layer. I found that my convolution layer had fair edge definition but found that most of the images in the relu layer's viewable view were black, indicating that they were not observed by the relu. Our algorithm can be further enhanced.

2. The original image is not processed in any way, usually consists of RGB colors and contains details and textures. the image in the feature map is filtered by a filter for specific features. The features are different for each image, removing a lot of details and possibly highlighting edges, corners, etc. And because of convolution etc. the matrix is not the same as the original. And due to dimensional operations on the matrix such as convolution, the pixel size of the image is smaller than the original image.

Part 5: Image Classification





I found all the numbers for the first three images, but the last one is too dense, and the image size is much smaller, so I still get a number read twice after I add the pad. Combined, I got 22/70 correct, which is probably around 31%. I think my main problem here is with threshold, not reading the numbers in completely which leads to a drop in accuracy. And one thing, I've noticed that png format images have a bug of too large size in Q3.3, but not over here. But the problem here is that the numbers it reads in are randomly distributed, not in the order of the pictures, which I think is caused by CPU parallel management or different running time of each network.

```
Image: ../images/image1.JPG
Recognized Digits: 0, 0, 2, 3, 2, 8, 9, 4, 7, 7
Image: ../images/image2.JPG
Recognized Digits: 2, 0, 4, 2, 0, 3, 5, 9, 3, 0
Image: ../images/image3.png
Recognized Digits: 6, 6, 6, 0, 2
Image: ../images/image4.jpg
Recognized Digits: 8, 3, 9, 4, 3, 5, 3, 3, 2, 2, 4, 2, 3, 9, 3, 0, 4, 3, 0, 2, 8, 8, 4, 2, 3, 3, 9, 0, 4, 6, 3, 0, 6, 4, 2, 9, 6, 6, 2, 5, 3, 3, 0, 6, 6, 7, 2, 3,
 5, 6, 3, 5, 0, 5, 0
```