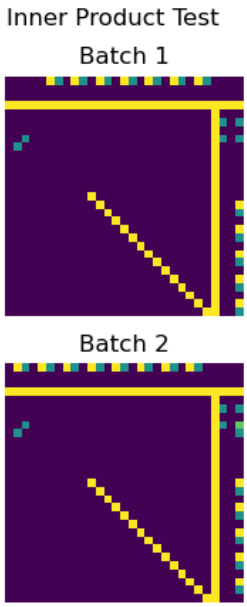
Project1 - Report

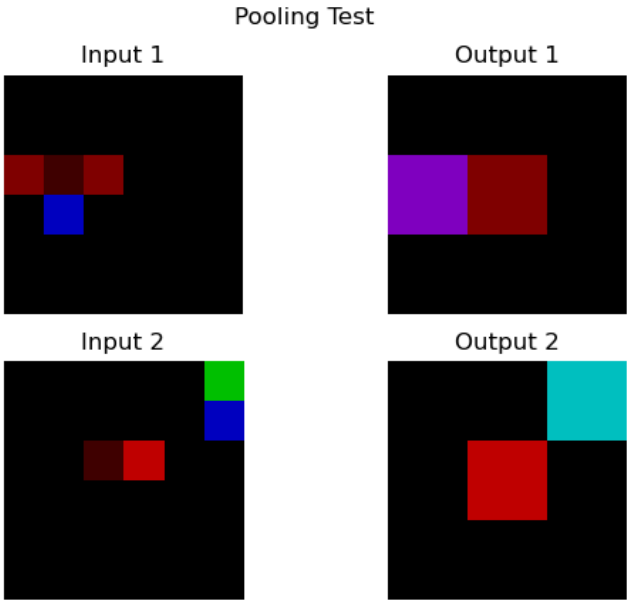
Rongsheng Qian

301449387

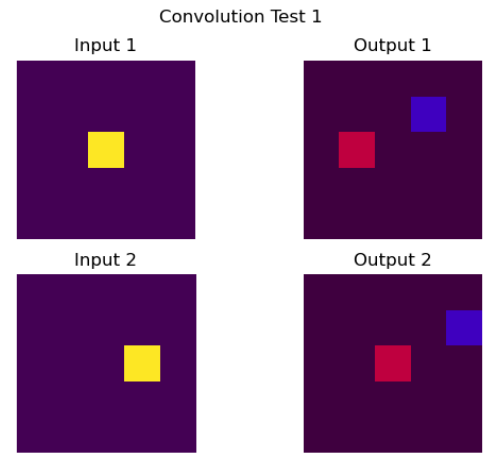
Q 1.1 Inner Product Layer

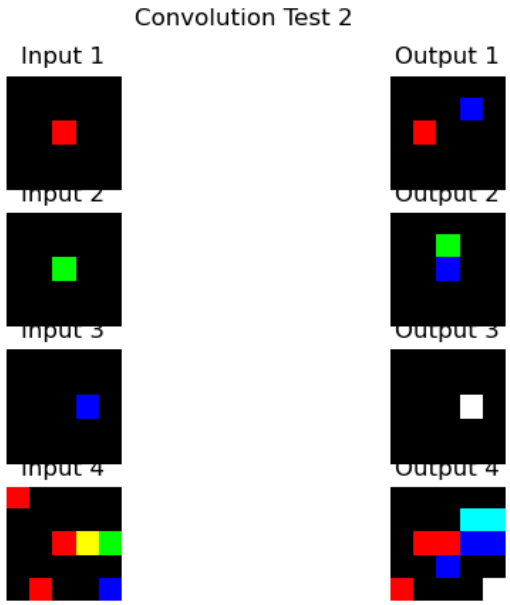


Q 1.2 Pooling Layer



Q 1.3 Convolution Layer





Q 1.4 ReLU

Problem I meet:

When I am running train_lenet to train my network there will report a warning:

"RuntimeWarning: divide by zero encountered in log nll = -np.sum(np.log(P[I == 1]))”

Solution:

I think there is sth wrong caused by copying pointer in relu forward. I don't know how to fix it so i just implemented it in another way. Then the network worked and accuracy reached 95. I have tried np.maximum() function. It works for me. (My original method is out = in; out[out<0]=0 which returns error)

Q 2.1 ReLU

Problem I meet:

None

Solution:

None

```python
def relu_backward(output, input_data, layer):
    ###### Fill in the code here ######
    # Replace the following line with your implementation.
    #input_od = np.zeros_like(input_data['data'])
    last_diff = output["diff"]
    data = input_data["data"]

    diff_h = np.where(data < 0, 0, 1)

    input_od = last_diff * diff_h

    return input_od
```

Q 2.2 Inner Product layer

Problem I meet:

Output param[“b”]’s size doesn’t match

Solution:

Do transpose on the output param[“b”] to let its shape become [1,_]

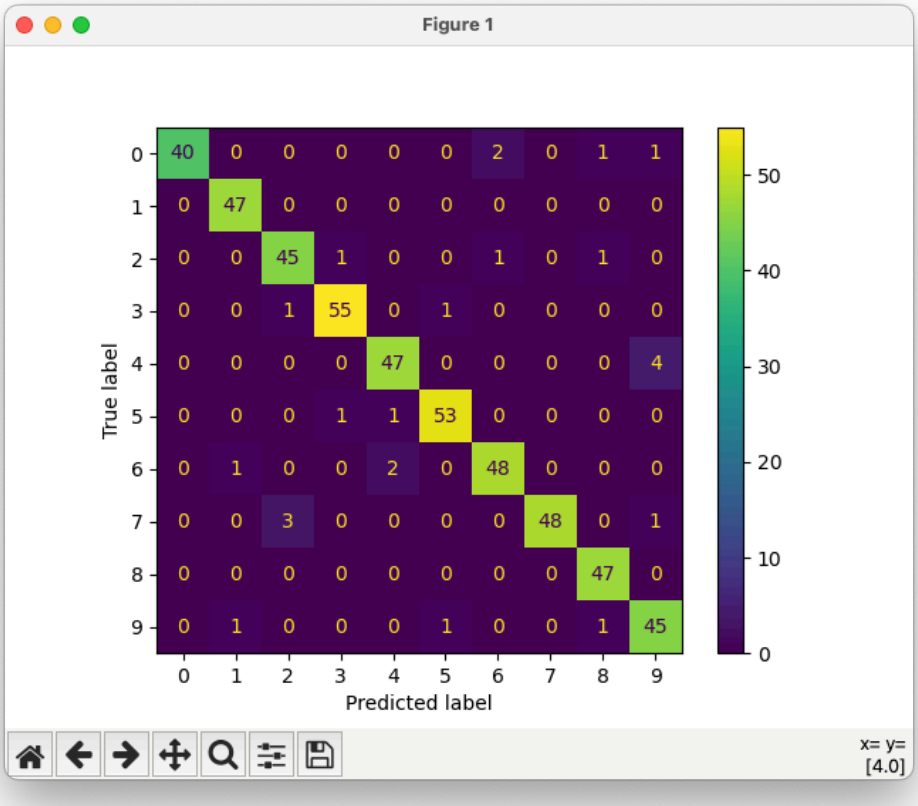
```python
def inner_product_backward(output, input_data, layer, param):
    """
    Backward pass of inner product layer.

    Parameters:
    - output (dict): Contains the output data.
    - input_data (dict): Contains the input data.
    - layer (dict): Contains the configuration for the inner product layer.
    - param (dict): Contains the weights and biases for the inner product layer.
    """
    param_grad = {}
    ###### Fill in the code here ######
    # Replace the following lines with your implementation.
    param_grad['b'] = np.zeros_like(param['b'])
```

```
    param_grad['w'] = np.zeros_like(param['w'])
    input_od = None

    last_diff = output["diff"]
    data = input_data["data"]
    batch_size = input_data["batch_size"]
    input_od = np.zeros_like(data)

    input_size = input_data["data"].shape[0]
    output_size = last_diff.shape[0]

    learning = np.ones((batch_size,1))
    diff_h = param["w"].T
    #print(last_diff.shape,learning.shape)
    param_grad['b'] = np.matmul(last_diff,learning).T

    for batch in range(batch_size):
        hi = data[:,batch]
        for i in range(output_size):
            tempt = hi * last_diff[i,batch]
            param_grad['w'][:,i] = param_grad['w'][:,i] + tempt
        # Done
        input_od[:,batch] = np.matmul(last_diff[:,batch],diff_h)

    return param_grad, input_od
```

Q 3.1 Training

```
0
test accuracy: 0.104
500
test accuracy: 0.954
1000
test accuracy: 0.948
1500
test accuracy: 0.952
2000
test accuracy: 0.956
```

Q 3.2 Test the network



	precision	recall	f1-score	support
0.0	1.00	0.91	0.95	44
1.0	0.96	1.00	0.98	47
2.0	0.92	0.94	0.93	48
3.0	0.96	0.96	0.96	57
4.0	0.94	0.92	0.93	51
5.0	0.96	0.96	0.96	55

```
           6.0       0.94      0.94      0.94        51
           7.0       1.00      0.92      0.96        52
           8.0       0.94      1.00      0.97        47
           9.0       0.88      0.94      0.91        48
    accuracy                             0.95       500
   macro avg       0.95      0.95      0.95       500
weighted avg       0.95      0.95      0.95       500
```

The top two confused pairs of classes are (4->9) which has 4 wrong cases and (7->2) which have 3 wrong cases.

For the pair (4,9), the network may predict 4 as 9. So 9.0 get the lowest precision (0.88) in ten digit. The reason why it happened was the upper part of “4” sometimes closed (or nearly closed) by handwriting which seemed like 9.

For the pair (7,2), the network may predict 7 as 2. So 2.0 get the second lowest precision (0.92) in ten digit. The reason why it happened was the upper part of “7” looks realy like the upper part of “2” and sometimes the corner of “7” is not sharp enough which is more likely to predict as “2”.

Q 3.3 Real-world testing

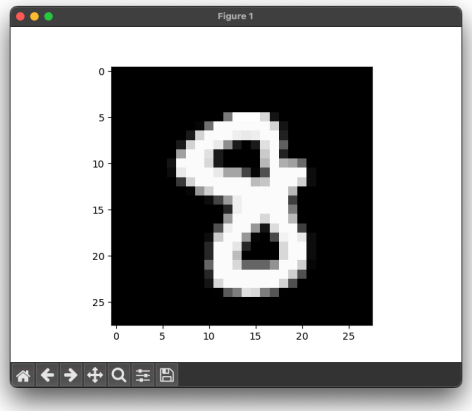
```
predict: [7]; label: 7
predict: [4]; label: 4
predict: [0]; label: 0
predict: [2]; label: 2
predict: [9]; label: 9
```

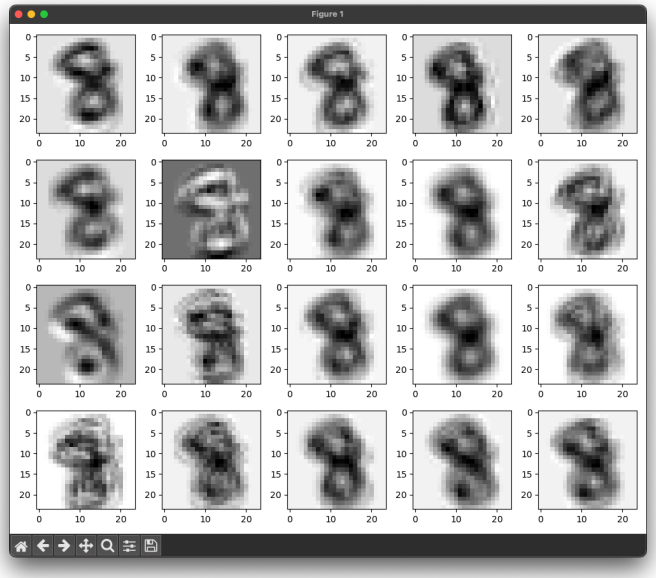
Using Minst digit picture can get the high accuracy.(100% for 5 images)

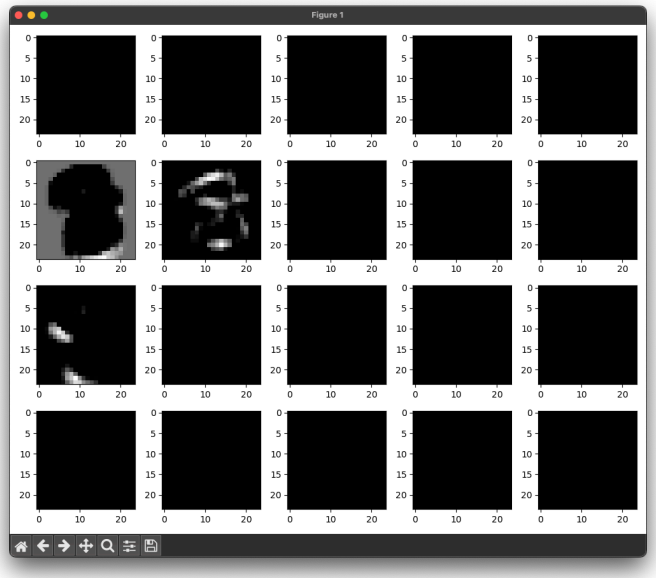
Network works well when the image looks like the hand-write stuff in MINST (same writing type and bold font). We use other "type" hand-write the network may have lower accuracy due to the lack of training data. And If we don’t use the bold font image the accuracy rate will be extremely low. (Lower than 40%)

Problem I meet: Using Minst digit picture can get the (3/5) accuracy. Reason & solution: Forget to normalize the image from (0-255) to (0-1)

Q 4.1 Visualization





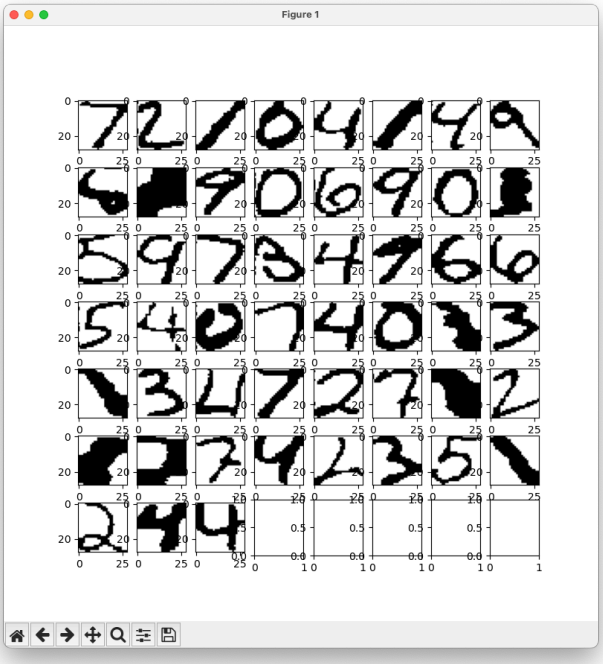


Q 4.2 Compare

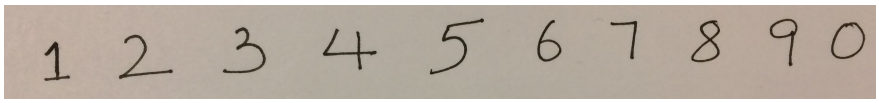
From the original input to the conv layer. The image go through 20 filters and do the convolution. We can easily see the “8” shape in each 20 images in conv layers. The difference between imput and conv layer is the “8” is blotted iyt by a camera blur and have clear edges for same filters’ results. The digit color changed into white and the background color changed into white for most of the output. **It should be noted that most of the values in this layer are negative but imshow() function still represents them as a white or grey color.**

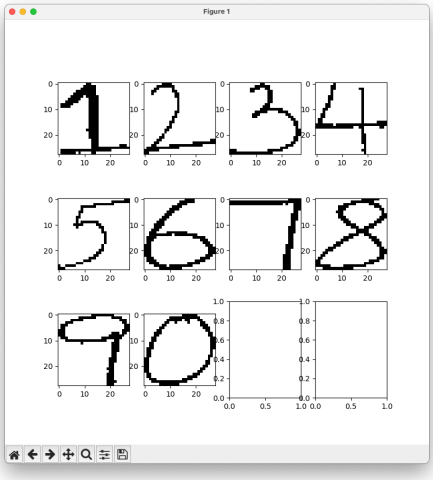
From the conv layer to relu layer. The images go through max(0,x) which means many negative values will be discarded in this layer. Comparing with the previous conv layer, this layer is dark and only a small area is white which are **features of the original input**. For example in **rwo2 column1 image** we can see the edge of “8” clearly after filtering by relu which may be useful to help network to recognize the digit.

Part 5 Image Classification

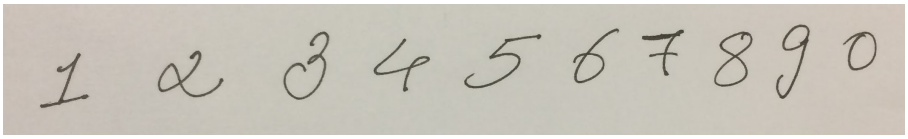


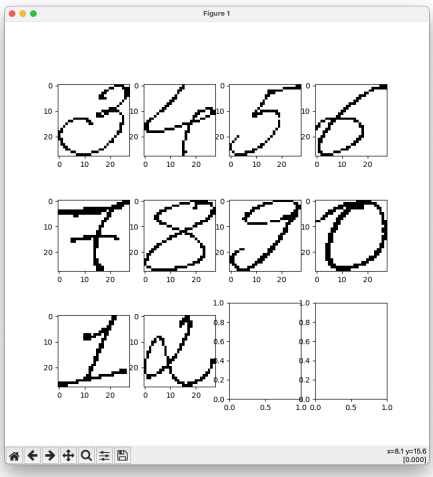

```
image4.jpg
prediict: [8, 8, 6, 9, 4, 7, 2, 4, 9, 1, 2, 8, 6, 2, 8, 1, 8, 8, 8, 2, 4, 6, 8, 6, 8, 4, 9, 1, 4, 0, 4, 8, 8, 2, 0, 0, 3, 8, 1, 3, 1, 0, 1, 2, 9, 2, 2, 2, 2, 2, 5]
label:    [7, 2, 1, 0, 4, 1, 4, 9, 5, 9, 0, 6, 9, 0, 1, 5, 9, 7, 3, 4, 9, 6, 4, 5, 4, 0, 7, 4, 0, 1, 3, 1, 3, 4, 7, 2, 7, 1, 2, 1, 1, 7, 4, 2, 3, 5, 1, 2, 4, 4]
```



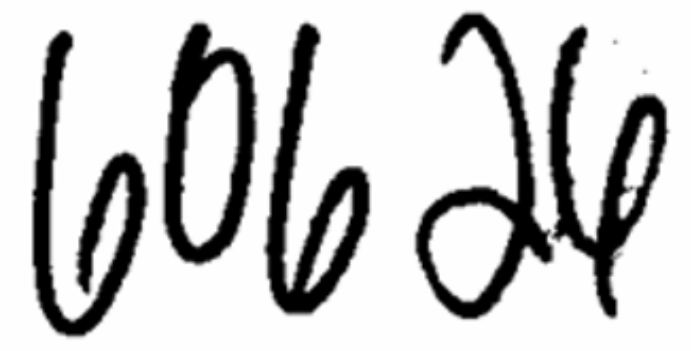


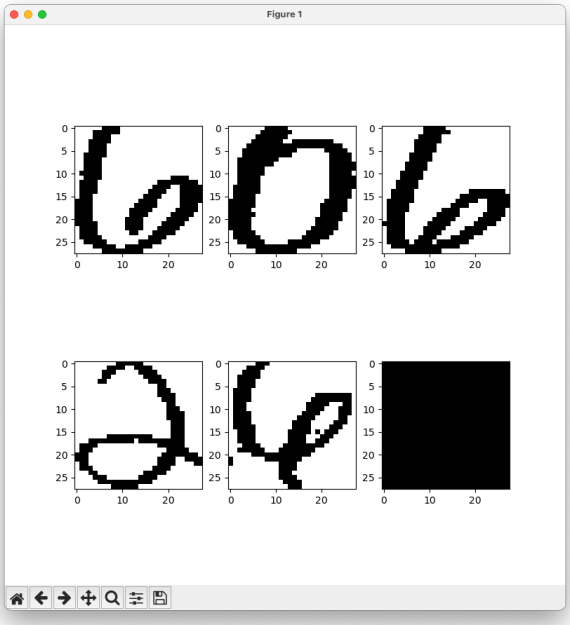

```
image1.JPG
prediict: [8, 8, 4, 4, 9, 4, 8, 8, 6, 8]
label:    [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
```



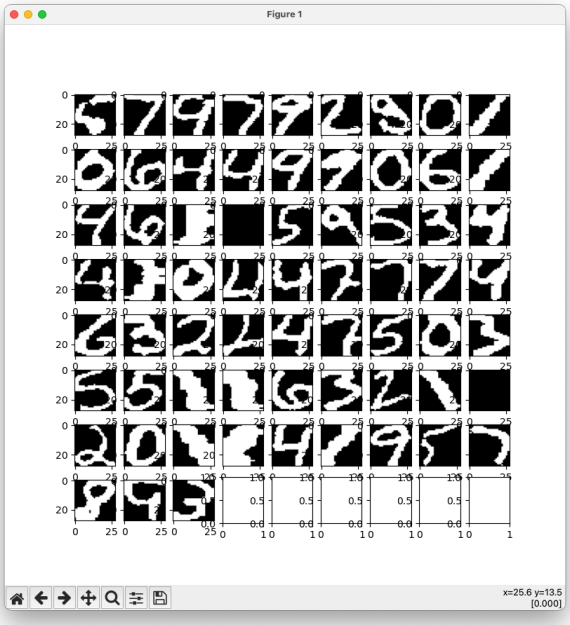



```
image2.JPG
prediict: [4, 4, 8, 4, 9, 6, 8, 8, 8, 4]
label:    [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
```






```
image3.png
prediict: [4, 8, 8, 6, 4, 1]
label:    [6, 0, 6, 2, 4]
```




```
minst.jpg
prediict: [2, 2, 4, 3, 8, 0, 3, 1, 0, 0, 6, 6, 2, 7, 1, 0, 4, 8, 7, 0, 8, 1, 4, 4, 7, 4, 9, 2, 2, 0, 5, 4, 4, 1, 2, 4, 5, 3, 0, 2, 4, 4, 3, 0, 5, 7, 7, 3, 0, 6, 3, 6, 3, 1, 2, 0, 2, 8, 4, 8, 7, 2, 2, 8, 4, 6]
label:    [7, 2, 1, 0, 4, 1, 4, 9, 5, 9, 0, 6, 9, 0, 1, 5, 9, 7, 3, 4, 9, 6, 4, 5, 4, 0, 7, 4, 0, 1, 3, 1, 3, 4, 7, 2, 7, 1, 2, 1, 1, 7, 4, 2, 3, 5, 1, 2, 4, 4, 6, 3, 5, 5, 6, 0, 4, 1, 9, 5, 7, 8, 9, 3]
```

Comment

I use image from MINST in 3.3 question can get almost 100% accuracy and my test accuracy is 95%. However for this part I get the low accuracy.

Our network seems work well when the image looks like the hand-write stuff in MINST (same writing type and foreground & background color and bold font). The accuracy of MINIST type picture (the last example image) seems better than previous four example picture. I guess that if we use other "type" hand-write the network may not realize them well due to the lack of training data.

And for different color of digit and its background color, I have to use different cv.threshold.

For the previous four examples which is black digit and white blackground I can use (0-255) `_, binary_image = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)`. However for the last MINST one I have to change the threshold to (127,255) `_, binary_image = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY)` which does not work for the previous four examples.