Project 1

Digit recognition with convolutional neural networks

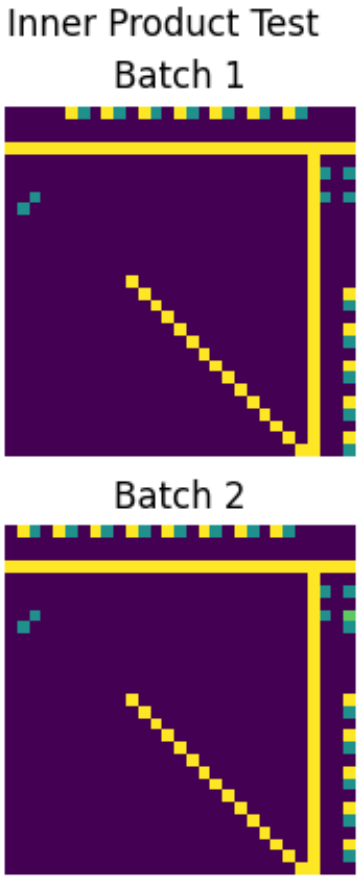
Wenxiang He

301417521

Due date: 23:59 9/24 (2023)

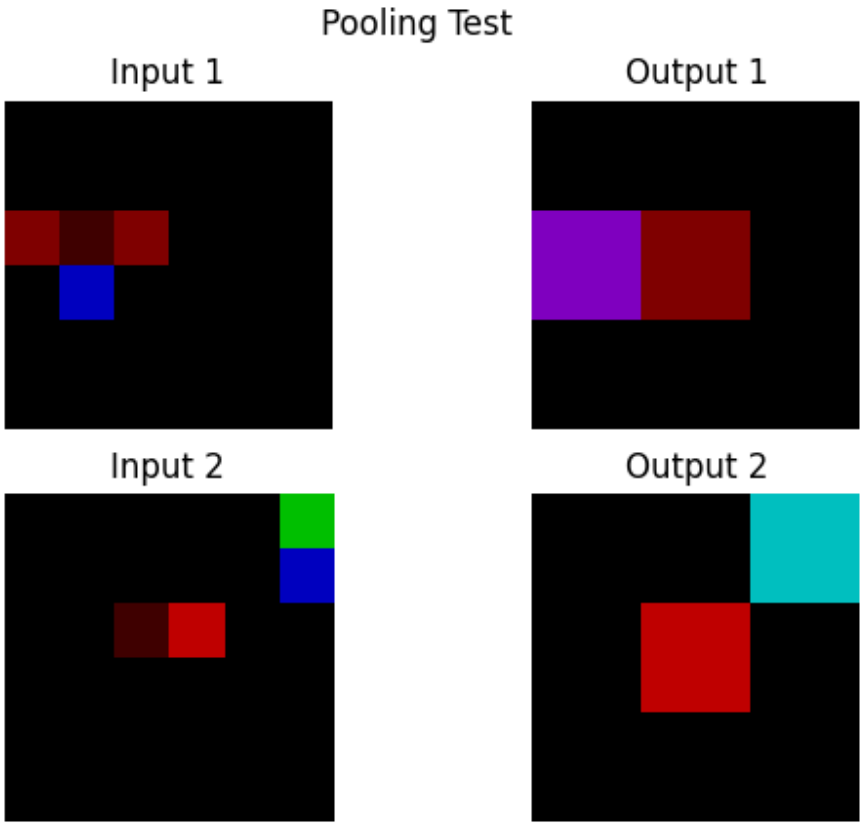
Q 1.1 Inner Product Layer - 1 Pts

For this part, I use f (x) = W x + b and format W, x, b into proper size for matrix production, and save the f (x) into output data.



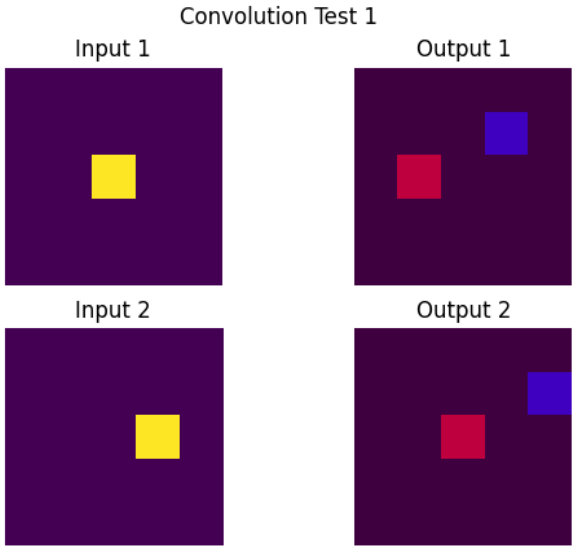
Q 1.2 Pooling Layer - 1 Pts

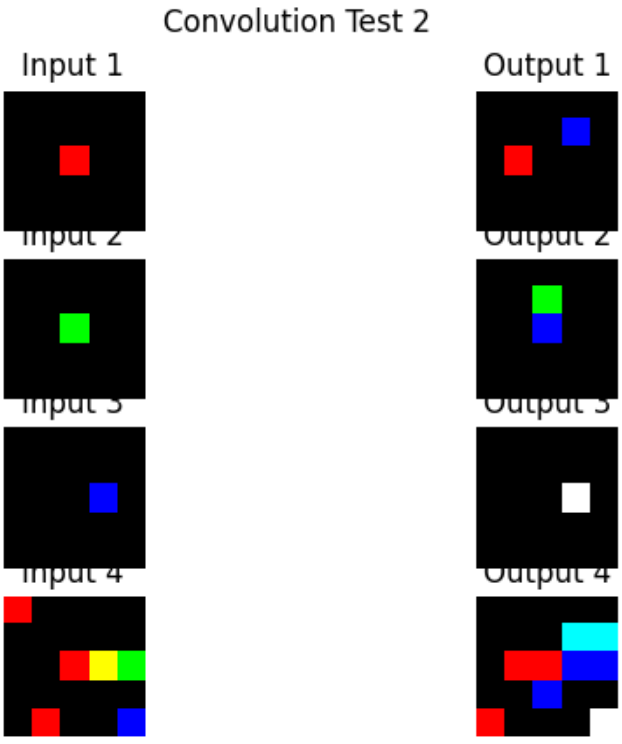
For this part, I first reshape the input data from 2D matrix to 4D matrix, and traverse through batch_size, c, h_out, w_out, to get local max in kernel from input data, and save it into output data. Finally, reshape the 4D matrix to 2D matrix.



Q 1.3 Convolution Layer - 1 Pts

For this part, I use the function im2col_conv_batch to make a convolution process to the input data, and format W, x, b into proper size for matrix production, and save the result into output data.





Q 1.4 ReLU - 1 Pts

For this part, I simply check the data in the input matrix. If it is greater than 0, keep it, otherwise save 0 into it.

I write one test script test_relu.py

```
import numpy as np
import matplotlib.pyplot as plt
from relu import relu_forward

def test_relu():
    input_data = {'data': np.zeros((3*3,1))}
    input_data['data'][1, 0] = 0.5
    input_data['data'][2, 0] = 0.25
    input_data['data'][6, 0]= -0.5

    input_data['width'] = 3
    input_data['height'] = 3
    input_data['channel'] = 1
    input_data['batch_size'] = 1

    relu_forward(input_data)

test_relu()
```

Here is the correct output:

```
(cv_proj1) PS D:\sfu-fall-2023\cmpt 412\project1\project1_package\project1\python> python test_relu.py
input:
 [[ 0.  ]
 [ 0.5 ]
 [ 0.25]
 [ 0.  ]
 [ 0.  ]
 [ 0.  ]
 [-0.5 ]
 [ 0.  ]
 [ 0.  ]]
output:
 [[0.  ]
 [0.5 ]
 [0.25]
 [0.  ]
 [0.  ]
 [0.  ]
 [0.  ]
 [0.  ]
 [0.  ]]
```

Q 2.1 ReLU - 1 Pts

For this part, I get the gradient form the last layer, and get derivative of input data, multiply them and save into input_od

```
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'])
    # gradient from last layer
    relu_diff = output["diff"]
    # derivative for input data
    input_od = input_data['data'] > 0
    input_od = relu_diff * input_od

    return input_od
```

Q 2.2 Inner Product layer - 1 Pts

For this part, I use the similar process to calculate θl/θw, θl/θh, and θl/θb and save them into param_grad, input_od.

```
def inner_product_backward(output, input_data, layer, param):
    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'])

    # dY  dl/dy
    dY = output["diff"]
    # xT
    input_transp = input_data['data'].T
    # dw dl/dw = xT * dY
    param_grad['w'] = np.dot(input_data['data'], dY.T)
    param_grad['b'] = np.sum(dY.T, axis=0)
    # dl/dhi-1 : dx = dY * wT
    input_od = np.dot(param["w"], dY)

    return param_grad, input_od
```

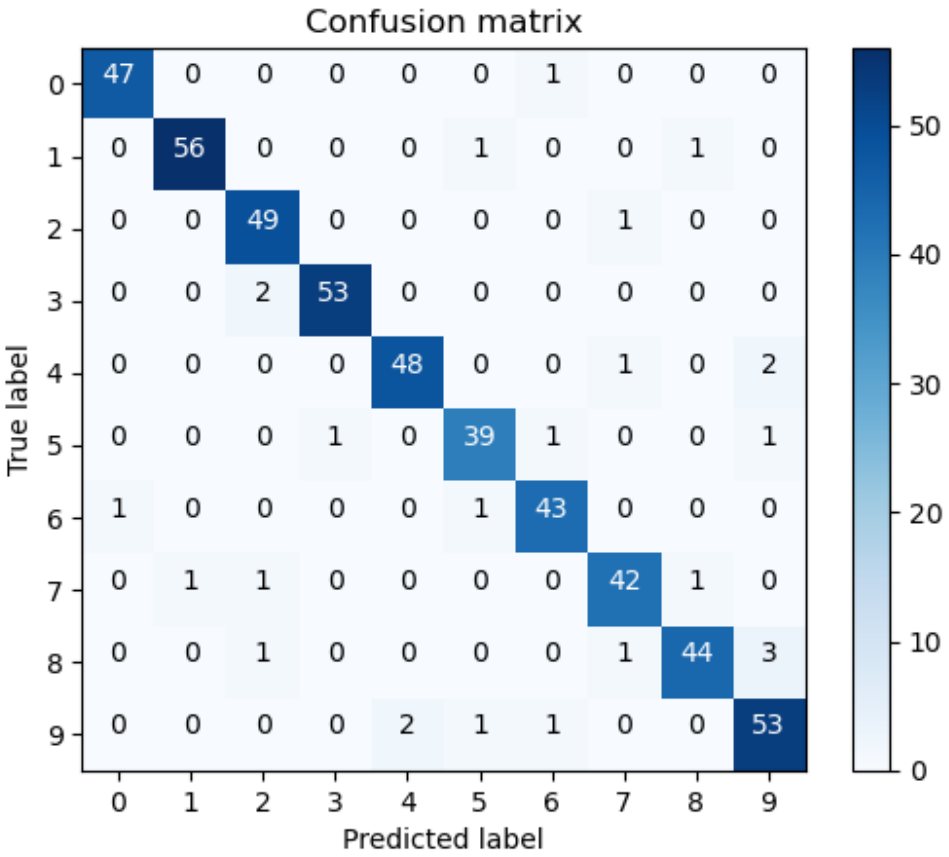
Q 3.1 Training - 1 pts

Here is the accuracy of training:

```
(cv_proj1) PS C:\Users\xiaom\Desktop\project1\python> python train_lenet.py
cost = 2.305296955167195 training_percent = 0.12
0
test accuracy: 0.104
cost = 0.8034168574187097 training_percent = 0.72
cost = 0.37044022611820937 training_percent = 0.89
cost = 0.09253204643085242 training_percent = 0.98
cost = 0.04824883211488296 training_percent = 0.99
cost = 0.18445141779040056 training_percent = 0.95
cost = 0.06009131053153033 training_percent = 0.97
cost = 0.017375591775055176 training_percent = 1.0
cost = 0.0059950033762022825 training_percent = 1.0
cost = 0.0038172953737512943 training_percent = 1.0
cost = 0.004336162956589914 training_percent = 1.0
500
test accuracy: 0.954
cost = 0.004301765227397744 training_percent = 1.0
cost = 0.005410212114839199 training_percent = 1.0
cost = 0.002666095463815 16 training_percent = 1.0
cost = 0.0026244894413573736 training_percent = 1.0
cost = 0.0025417737617417723 training_percent = 1.0
cost = 0.001967342390643068 training_percent = 1.0
cost = 0.0022130515342078435 training_percent = 1.0
cost = 0.0025968084363319436 training_percent = 1.0
cost = 0.001289642181587618 training_percent = 1.0
cost = 0.0027265550496138064 training_percent = 1.0
1000
test accuracy: 0.948
cost = 0.001220395364043329 training_percent = 1.0
cost = 0.002218197292543784 training_percent = 1.0
cost = 0.0021181262057415034 training_percent = 1.0
cost = 0.0016461537469412107 training_percent = 1.0
cost = 0.001232852208372267 training_percent = 1.0
cost = 0.0012961917851322443 training_percent = 1.0
cost = 0.001554945922941811 training_percent = 1.0
cost = 0.0010098608391492742 training_percent = 1.0
cost = 0.0006668825209824434 training_percent = 1.0
cost = 0.0015584701255792733 training_percent = 1.0
1500
test accuracy: 0.952
cost = 0.0010136067965389151 training_percent = 1.0
cost = 0.001292479835702357 training_percent = 1.0
cost = 0.0018210161668457331 training_percent = 1.0
cost = 0.0017631428900987311 training_percent = 1.0
cost = 0.0015098177254967698 training_percent = 1.0
cost = 0.0009671198864059581 training_percent = 1.0
cost = 0.0009002493629969033 training_percent = 1.0
cost = 0.001429184839717365 training_percent = 1.0
cost = 0.000831173762873509 training_percent = 1.0
cost = 0.0013441266360470045 training_percent = 1.0
2000
test accuracy: 0.956
test accuracy: 0.956
(cv_proj1) PS C:\Users\xiaom\Desktop\project1\python>
```

After 500 iterations, the training percent has reached 1.0, and the test accuracy has been steady around 95%. Meanwhile, there is a significant decrease in loss function (cost) through the iteration.

Q 3.2 Test the network - 1 Pts



The results on the diagonal show the correctly predicted value. For the most two confused pairs, we need to check the value that is not on the diagonal and not 0. Here I choose:

Row 4 and Column 3 (2), the real value is 3 and mispredict to 2.

Row 9 and Column 10 (3), the real value is 8 and mispredict to 9.

*There are also values not on diagonal but equal to 2, they can also be used as top 2 confused pairs.

The reason:

The reason for confusion between 3 and 2 is that the spatial structure of the top half of 3 and 2 is very close, it is also the same for 8 and 9. This makes them have similar feature values. Additionally, the number of images and image quality of these numbers in the training set will also be affected.

Q 3.3 Real-world testing - 1 Pts

I write one test script test_real.py to get picture and predict

```
import numpy as np
from utils import get_lenet
from load_mnist import load_mnist
from scipy.io import loadmat
from conv_net import convnet_forward
from init_convnet import init_convnet
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from PIL import Image

# Load the model architecture
layers = get_lenet()
params = init_convnet(layers)

# Load the network
data = loadmat('../results/lenet.mat')
params_raw = data['params']

for params_idx in range(len(params)):
    raw_w = params_raw[0,params_idx][0,0][0]
    raw_b = params_raw[0,params_idx][0,0][1]
    assert params[params_idx]['w'].shape == raw_w.shape, 'weights do not
have the same shape'
    assert params[params_idx]['b'].shape == raw_b.shape, 'biases do not
have the same shape'
    params[params_idx]['w'] = raw_w
    params[params_idx]['b'] = raw_b

def test_single_image(img_path, params, layers):
    # open image
    img = Image.open(img_path)

    # Convert the image to grayscale
    img = img.convert('L')

    # Resize the image to 28x28
    img = img.resize((28, 28))
```

```
    # revert color to adapt minist data
    img = Image.fromarray(255 - np.array(img))

    plt.imshow(img, cmap='gray')
    plt.title('My Image')
    plt.axis('off')
    plt.show()

    # Convert image to numpy array and normalize
    img_array = np.array(img, dtype=np.float32) / 255.0

    # Reshape the image to match the expected input shape for the model
    # (batch_size, height, width, channel)
    img_array = np.reshape(img_array, (28, 28), order='F')
    # img_array = img_array.reshape(1, 28, 28, 1)

    # Update the batch_size for the network
    layers[0]['batch_size'] = 1

    # Forward pass
    _, P = convnet_forward(params, layers, img_array, test=True)

    # Get the predicted class
    predicted_class = P.argmax(axis = 0)

    return predicted_class[0]

# Test the function
img_path = '../images/5.jpg'
predicted_class = test_single_image(img_path, params, layers)
i = Image.open(img_path)
plt.imshow(i, cmap='gray')
plt.title(f"Predicted: {predicted_class}")
plt.axis('off')
plt.show()
print(f"The predicted class for the image is: {predicted_class}")
```

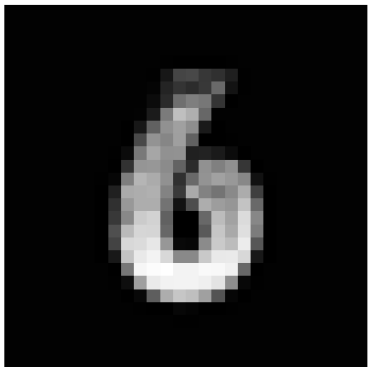
Here are five real word samples and the corresponding predicted value:

Predicted: 5

My Image

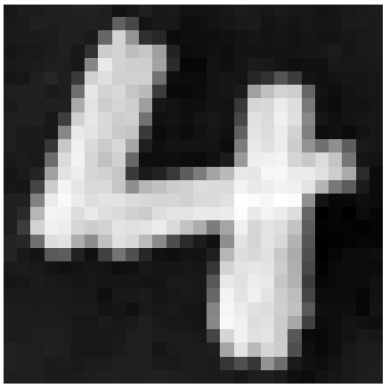
Predicted: 6

My Image



Predicted: 4

My Image



Predicted: 8

My Image

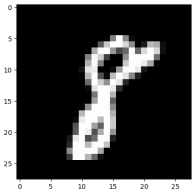
Predicted: 3

My Image

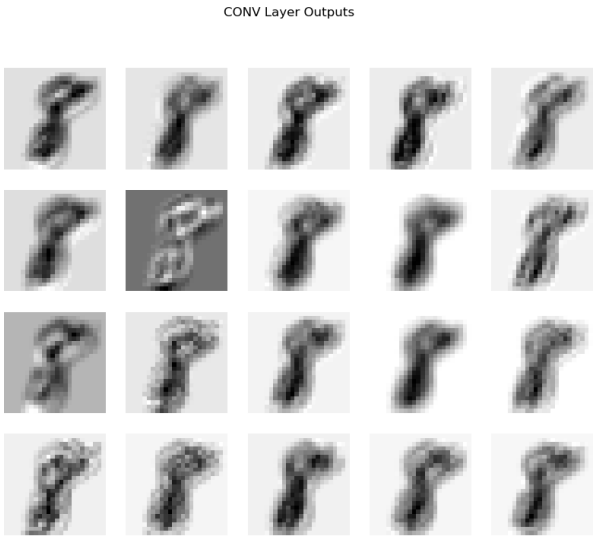
As shown above, all 5 real world samples have been correctly predicted.

Q 4.1 - 1 Pts

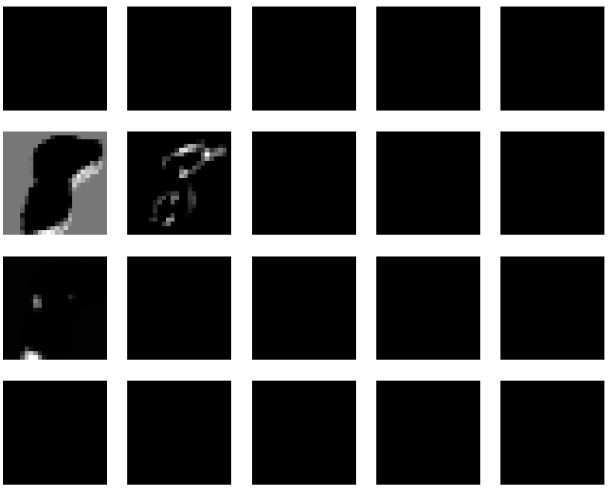
Input image:



Convolution layer:



Relu Layer:



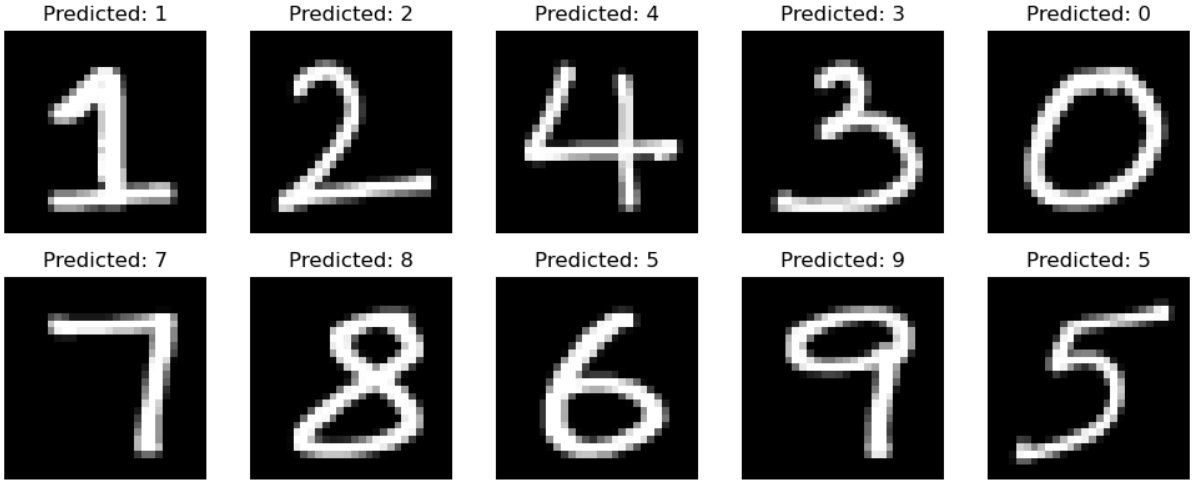
Q 4.2 - 1 Pts

The difference between input image and convolutional layer output:
Depending on the weights and biases applied, certain areas of the initial image may appear augmented or distorted when reaching the output of the convolutional layer. The purpose of the convolutional layer is to recognize local patterns in the image. It also use function $f(x) = W x + b$. The convolution kernel (or filter) is convolved with the image to generate a feature map that highlights the specific pattern corresponding to the convolution check. Therefore,there may be blur and enhancement in the convolutional layer output, that is due to the specific features extracted in the image by these convolutional cores.

The difference between convolutional layer output and ReLU layer output:
The ReLU layer is an activation function layer whose main function is to introduce nonlinearity. The ReLU activation function has the form $f(x) = \max(0, x)$, which sets all negative values to 0 while leaving positive values unchanged. Therefore, if the values of the convolutional layer output are positive, then these outputs will be the same after passing through the ReLU layer. While the negative value will be set to 0, and results in many dark images in ReLU layer output.

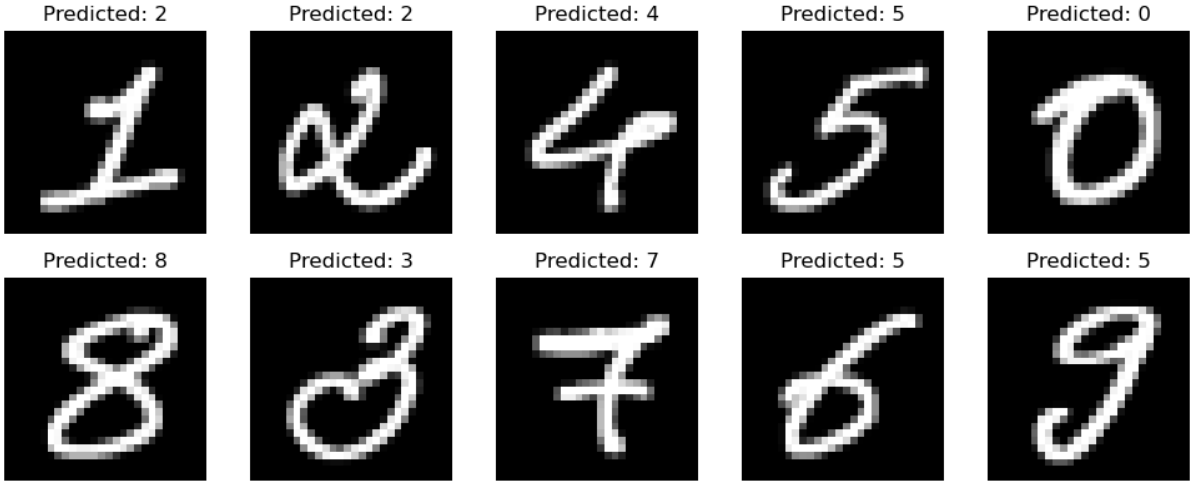
Part 5 Image Classification - 2 Pts

Image1:



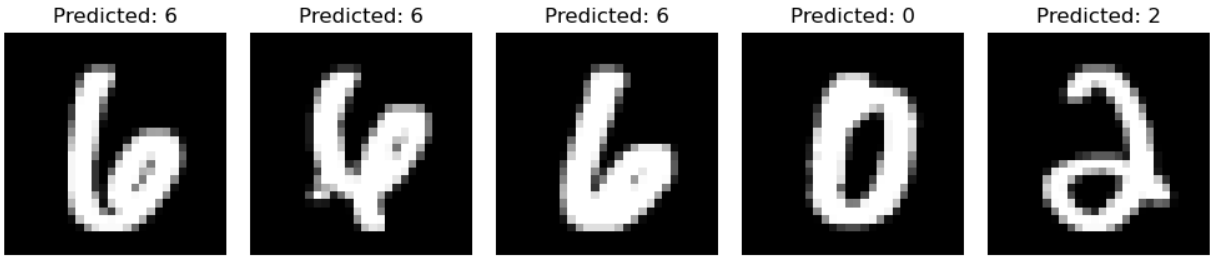
9 out of 10

Image2:



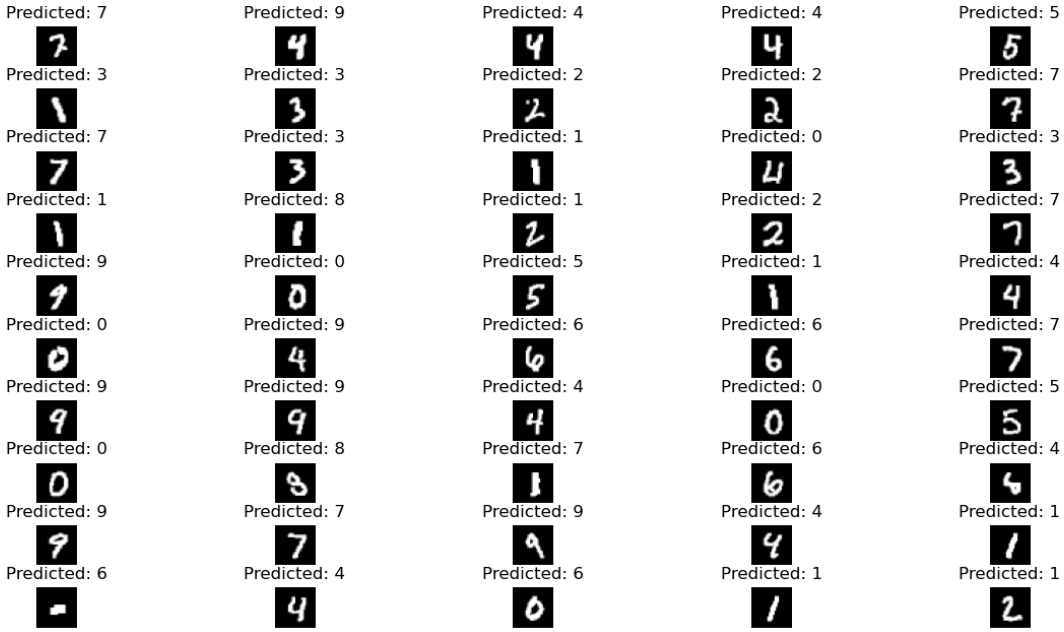
8 out of 10

Image3:



4 out of 5

Image4:



39 out of 50

For this part, in order to better predict, after putting the number into the bounding box, I use padding, cv2.erode, and cv2.dilate and reverse color to make sub images more like the image in the minist train set. It indeed increases the prediction rate.