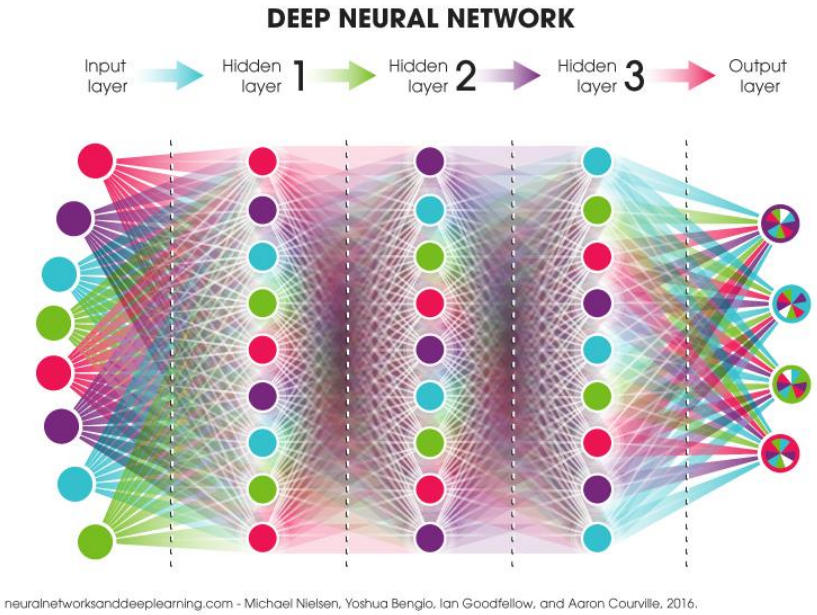
Project 5

Digit Recognition with Convolutional Neural Networks

Computer Vision - CMPT 762



Architecture:

- Input - 1×28×28
- Convolution - k = 5, s = 1, p = 0, 20 filters
- ReLU
- MAXPooling - k=2, s=2, p=0
- Convolution - k = 5, s = 1, p = 0, 50 filters
- ReLU
- MAXPooling - k=2, s=2, p=0
- Fully Connected layer - 500 neurons
- ReLU
- Loss layer

The input is passed to each layer in a structure with the following fields.

- height - height of the feature maps
- width - width of the feature maps
- channel - number of channels / feature maps
- batch size - batch size of the network. In this implementation, you will implement the mini-batch stochastic gradient descent to train the network. The idea behind this is very simple, instead of computing gradients and updating the parameters after each image, we doing after looking at a batch of images. This parameter batch size determines how many images it looks at once before updating the parameters.
- data - stores the actual data being passed between the layers. This is always supposed to be of the size [height × width × channel, batch size]. You can resize this structure during computations, but make sure to revert it to a two-dimensional matrix. The data is stored in a column major order. The row comes next, and the channel comes the last.
- diff - Stores the gradients with respect to the data, it has the same size as data. Each layer's parameters are stored in a structure param. You do not touch this in the forward pass.
- w - weight matrix of the layer
- b - bias

param_grad is used to store the gradients coupled at each layer with the following properties:

- w - stores the gradient of the loss with respect to w.
- b - stores the gradient of the loss with respect to the bias term.

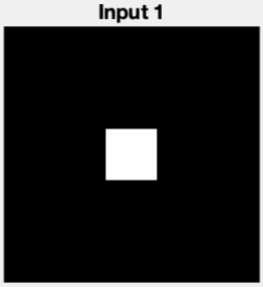
3. Programming

Part 1: Forward Pass

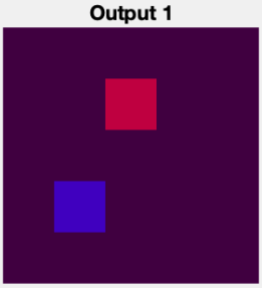
Run test_components.m to get the visualization results.

Convolution Test 1

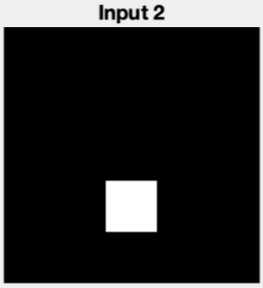
Input 1



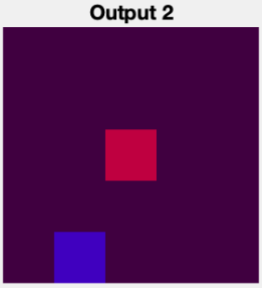
Output 1



Input 2

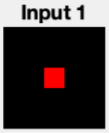


Output 2



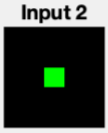
Convolution Test 2

Input 1



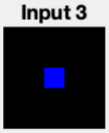
Output 1

Input 2



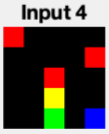
Output 2

Input 3



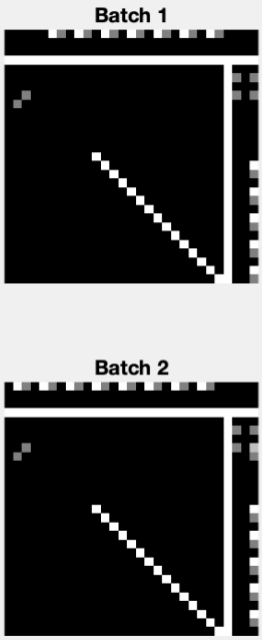
Output 3

Input 4

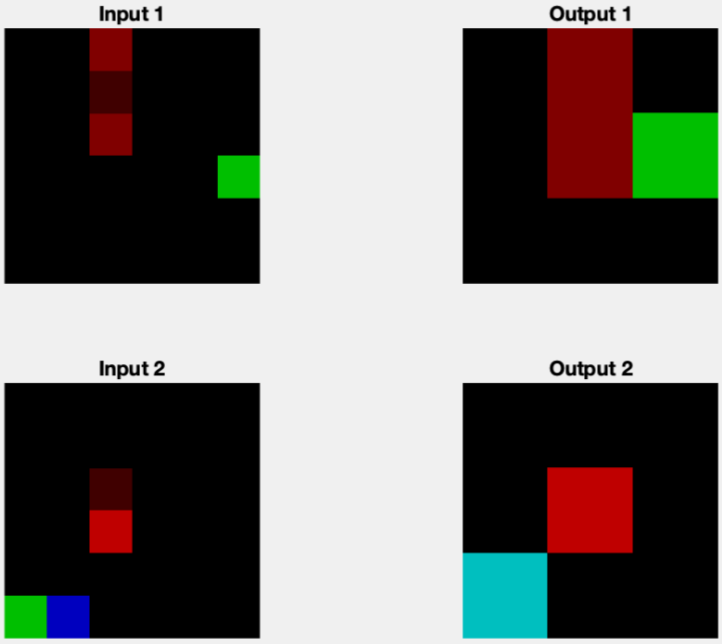


Output 4

Inner Product Test



Pooling Test



Q 1.1 Inner Product Layer

```
function [output] = inner_product_forward(input, layer, param)
```

Q 1.2 Pooling Layer

```
function [output] = pooling_layer_forward(input, layer)
```

Q 1.3 Convolution Layer

```
function [output] = conv_layer_forward(input, layer, param)
```

Q 1.4 ReLU

```
function [output] = relu_forward(input)
```

Part 2 Back propagation

Q 2.1 ReLU

```
function [input_od] = relu_backward(output, input, layer)
```

Q 2.2 Inner Product layer

```
function [param_grad, input_od] = inner_product_backward(output, input,
layer, param)
```

Putting the network together

This part has been done for us and is available in the function convnet forward.

```
function [output, P] = convnet_forward(params, layers, data)
```

Part 3 Training

The function conv_net puts both the forward and backward passes together and trains the network.

Q 3.1 Training

Run the script train_lenet.m for 3000 iterations. Test accuracy after training for 3000 more iterations is 97%.

cost = 0.273491 training_percent = 0.910000
cost = 0.279565 training_percent = 0.910000
cost = 0.176619 training_percent = 0.920000
cost = 0.127344 training_percent = 0.950000
cost = 0.191895 training_percent = 0.960000
test accuracy: 0.944000

cost = 0.192910 training_percent = 0.930000
cost = 0.131836 training_percent = 0.970000
cost = 0.115812 training_percent = 0.970000
cost = 0.103636 training_percent = 0.970000
cost = 0.124224 training_percent = 0.980000
test accuracy: 0.960000

cost = 0.111115 training_percent = 0.960000
cost = 0.113216 training_percent = 0.940000
cost = 0.134874 training_percent = 0.960000
cost = 0.067548 training_percent = 0.990000
cost = 0.095426 training_percent = 0.980000
test accuracy: 0.966000

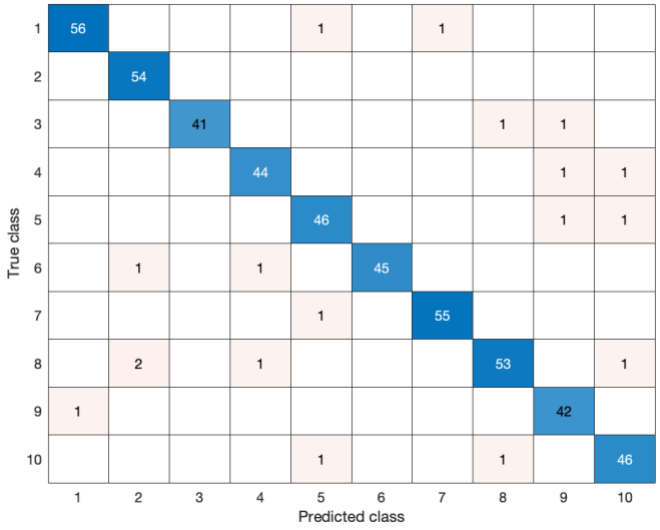
cost = 0.086685 training_percent = 0.980000
cost = 0.106186 training_percent = 0.950000
cost = 0.034245 training_percent = 1.000000
cost = 0.048397 training_percent = 1.000000
cost = 0.060728 training_percent = 0.970000
test accuracy: 0.968000

cost = 0.069977 training_percent = 1.000000
cost = 0.068312 training_percent = 0.980000
cost = 0.063643 training_percent = 0.980000
cost = 0.084625 training_percent = 0.960000
cost = 0.083214 training_percent = 0.980000
test accuracy: 0.970000

cost = 0.083081 training_percent = 0.970000
cost = 0.026531 training_percent = 1.000000
cost = 0.044653 training_percent = 0.980000
cost = 0.056298 training_percent = 0.980000
cost = 0.049833 training_percent = 0.990000
test accuracy: 0.970000

Q 3.2 Test the network

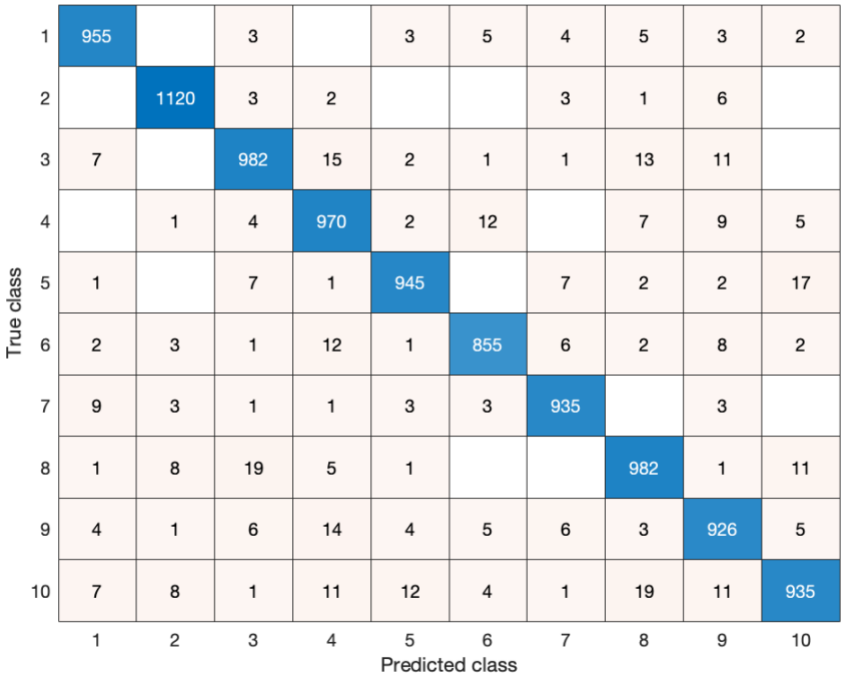
You can see the confusion matrix here.



56	0	0	0	1	0	1	0	0	0
0	54	0	0	0	0	0	0	0	0
0	0	41	0	0	0	0	1	1	0
0	0	0	44	0	0	0	0	1	1
0	0	0	0	46	0	0	0	1	1
0	1	0	1	0	45	0	0	0	0
0	0	0	0	1	0	55	0	0	0
0	2	0	1	0	0	0	53	0	1
1	0	0	0	0	0	0	0	42	0
0	0	0	0	1	0	0	1	0	46

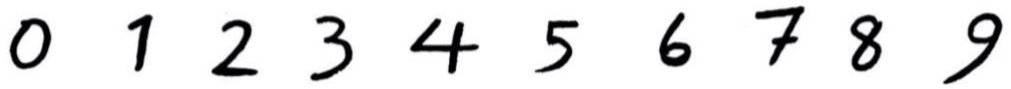
Here you can see 1 and 7 are the top two confused pairs of classes. (They could look like each other in handwritten text). In fact, for this pair we have the most wrong predictions in total by predicting the other number. This confusion matrix will change every time we run the code because we randomly select a dataset and compute confusion matrix on them.

So, I've changed the flag of fullset to true to compute the confusion matrix for all data. 2 and 7 are one of the top two confused pairs of classes here.



Q 3.3 Real-world testing

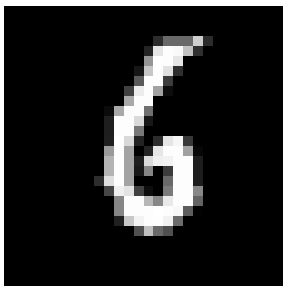
Run test_real_world.m to see the result of the system on real-world digit examples. I wrote digits myself and then manually crop them.



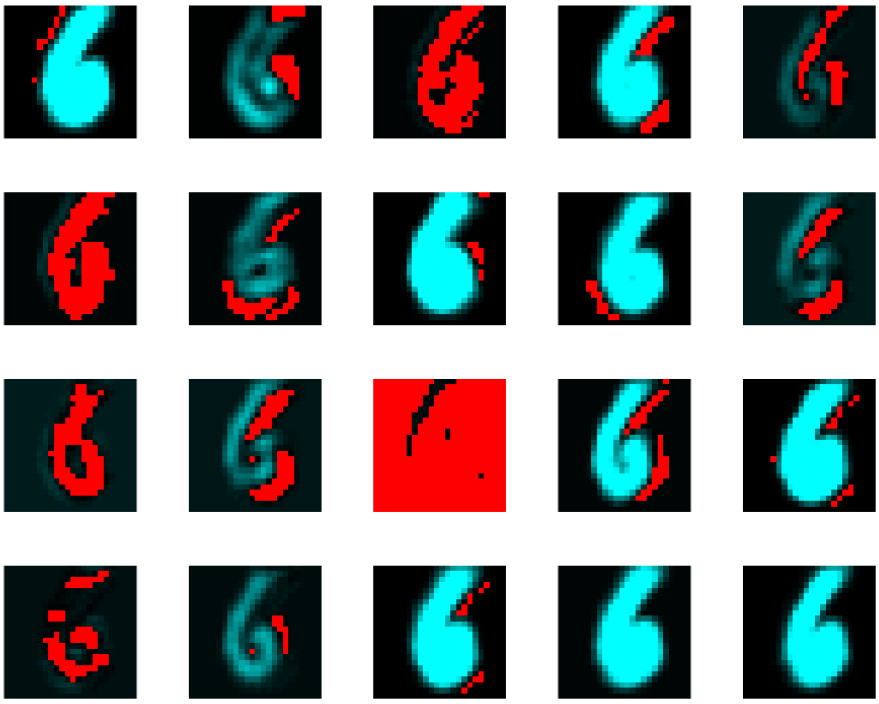
Here you can see the prediction for every digit. The upper number is the prediction for that input.

Part 4 Visualization

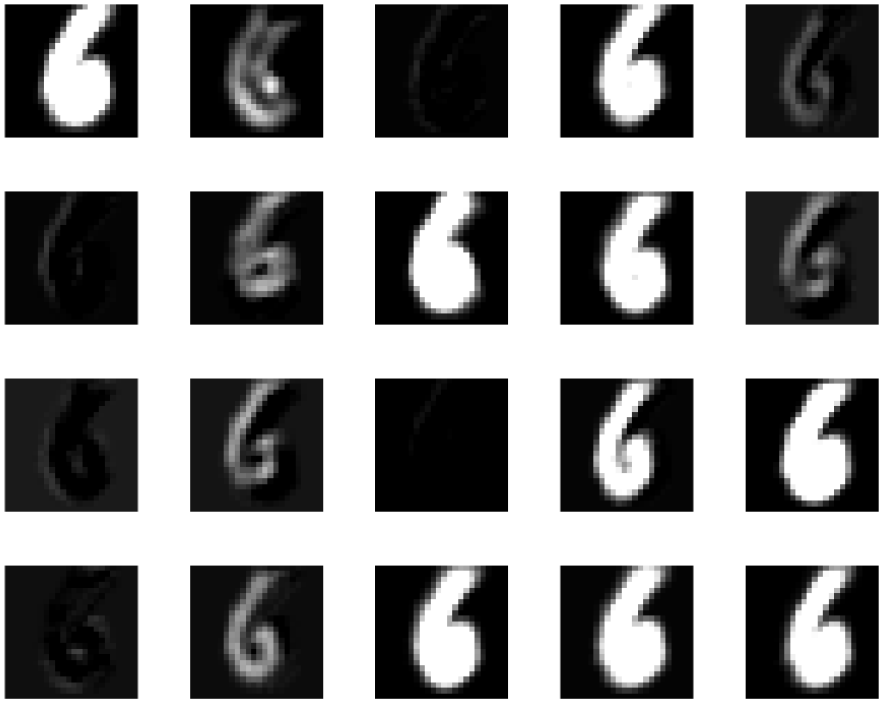
Q 4.1



input



20 features of that image at CONV layer



20 features of that image at ReLU layer

Q 4.2

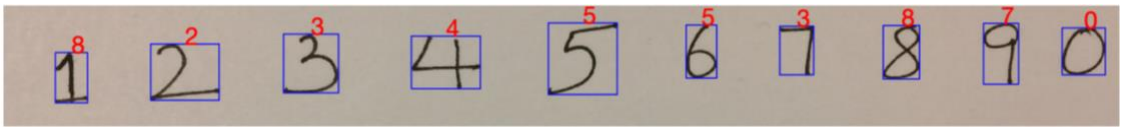
Because we have negative values for feature maps for convolution layer, we choose red color channel to display negative values and two other channels for positive values.

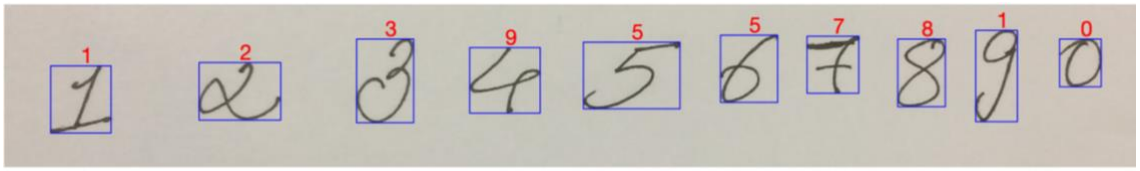
The first feature map is the output of the convolution layer. These images are filtered version of the original image. The weights for these filters have been learned in the training process. For example, here we can capture have blurry image, vertical and horizontal lines of input image. Another feature map is the output of the ReLU layer. The images are similar to the feature map from the previous layer and the only difference is that for the pixels that had a negative value have changed into zero.

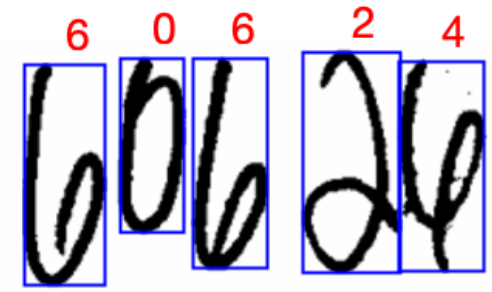
Part 5 Image Classification

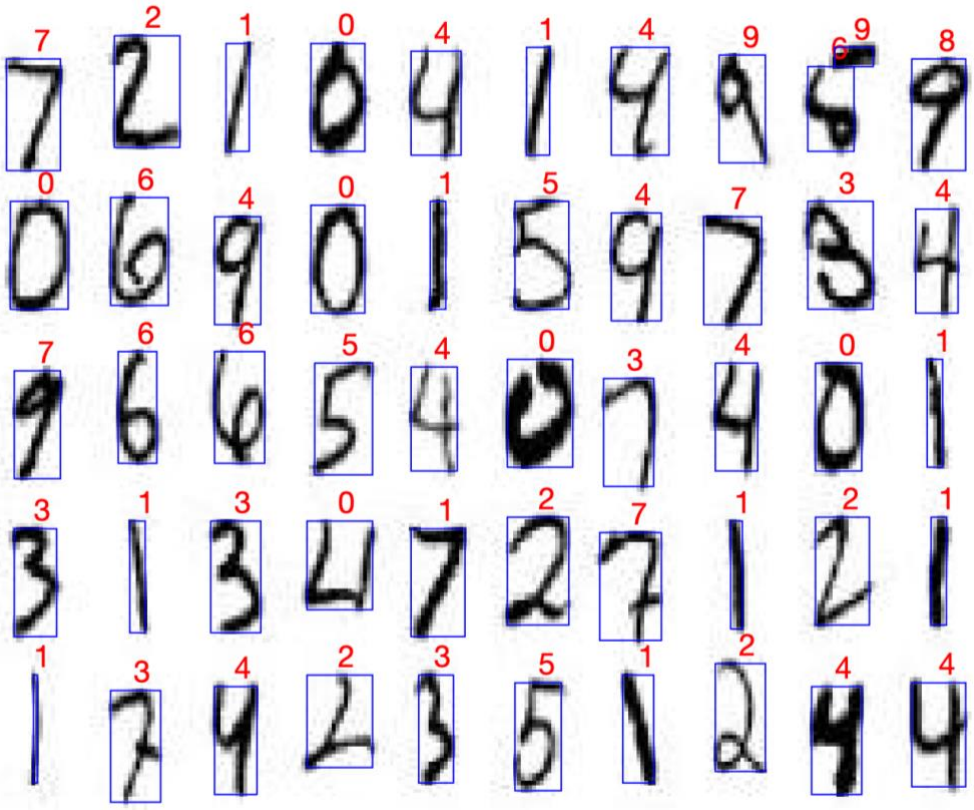
Run ec.m to get the results.
First, I applied thresholding on the image to get a binary image. Then I found connected components by matlab built-in function (bwconncomp). Finally, I padded every connected component to get a square image for every digit then resize it to 28*28 and passing through the network.

We got good results here. But wee should be careful in choosing threshold. Also for some digits if it is not connected we predict as multiple digits like the last example.









I also run the code on my handwritten from part 3.