Computer Vision - Project 1 Write-up
Digit recognition with convolutional neural networks

Nasrin Seifi-301394381

Forward-Backward Completion

In this task, I completed the forward and backward scripts conv_layer_forward.m, inner_product_forward.m, pooling_layer_forward.m, relu_forward.m, inner_product_backward.m, and relu_backward.m in a way that they properly contribute to the training process.
The training script (train_lenet.m) reported test accuracy: 0.97 after running for 3000 more iterations.
The testing script (test_network.m) reported the following accuracy and confusion matrix on the held-out test data.

```
>> test_network
Accuracy : 95.40%
The confusion matrix:
```

Zero	53	0	1	0	0	0	0	0	0	0
One	0	57	0	0	0	0	0	0	0	0
Two	0	0	46	2	0	0	0	0	0	0
Three	0	0	0	57	0	0	0	0	1	0
Four	0	0	1	0	43	0	0	0	0	1
Five	0	2	0	1	1	36	0	0	0	1
Six	0	0	0	0	0	1	58	0	0	0
Seven	0	0	3	0	0	0	0	41	0	2
Eight	1	0	1	0	0	0	0	0	45	0
Nine	0	0	0	0	4	0	0	0	0	41

The test results seem to be reasonably good, however, there still exist the room for improvement for classes like 4 and 9 which can be written pretty like each other in hand-written format and it can confuse the model during the classification (the confusion matrix shows that).

Real_world Testing

In this task, I used the test_network.m to implement the script for testing on the real-world data. The code is pretty much the same, but the batch size is 1 because we feed each image into the trained neural network and classify it. I have already hand-annotated the character images and my code computes the accuracy of the images being classified. For this part we use the same approach as the ec.m. First, convert image to gray scale then resize it to 28*28 and reshape it to a 2D matrix (784,1). The image should be transposed, and the important values should be white so I transform image pixels with (1 – pixel value). The script reports the following accuracy. The classification results of the annotated images are as follows (each cell is in the format of predicted(actual)).

```
>> test_real
2(2),
3(3),
4(4),
5(5),
7(7),

Real case accuracy: 100.0%
```

Figure 1Real world digit images

Visualization Analysis

In this section, I provide some analysis of the network internal layers as requested in the hand-out.

Figures 3 and 4 (the output matrices are normalized between 0 and 255 for clarity in visualization) are the output feature maps of the trained LeNet on the image of Figure 2. As it can be seen, the Convolution layer (2nd layer) is acting as the feature extractors (extracting features like edges, borders, or the horizontal/vertical gradient maps). The ReLU layer (3rd layer) is only removing negative values and replacing them with 0 which leads to more understandable feature maps for the network.

Figure 2 A sample image from MNIST dataset

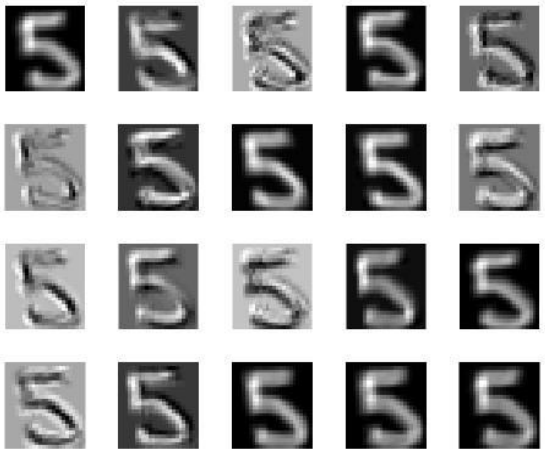


Figure 3 The output of 20 filters in Second Layer (ConvNet)

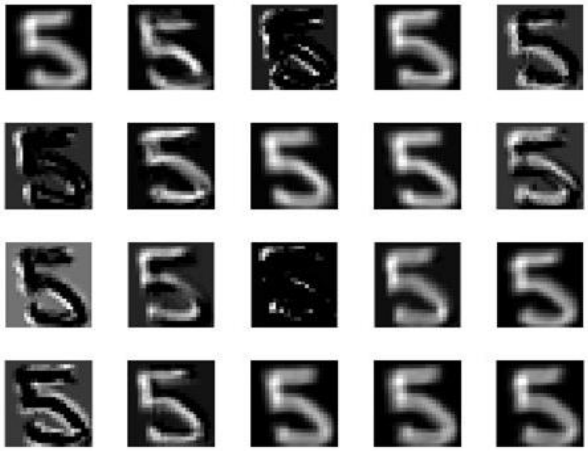


Figure 4 The output of 20 filters in Third Layer (ReLU)

Image Classification

In this section, I implemented the script (ec.m) which looks at the images of written characters, segments the images into actual characters (the segmented characters are visualized in Figure 4), resizes the character images into 28 × 28 and then feeds each image into the trained neural network and classifies them. I have already hand-annotated the character images and my code computes the accuracy of the images being classified.

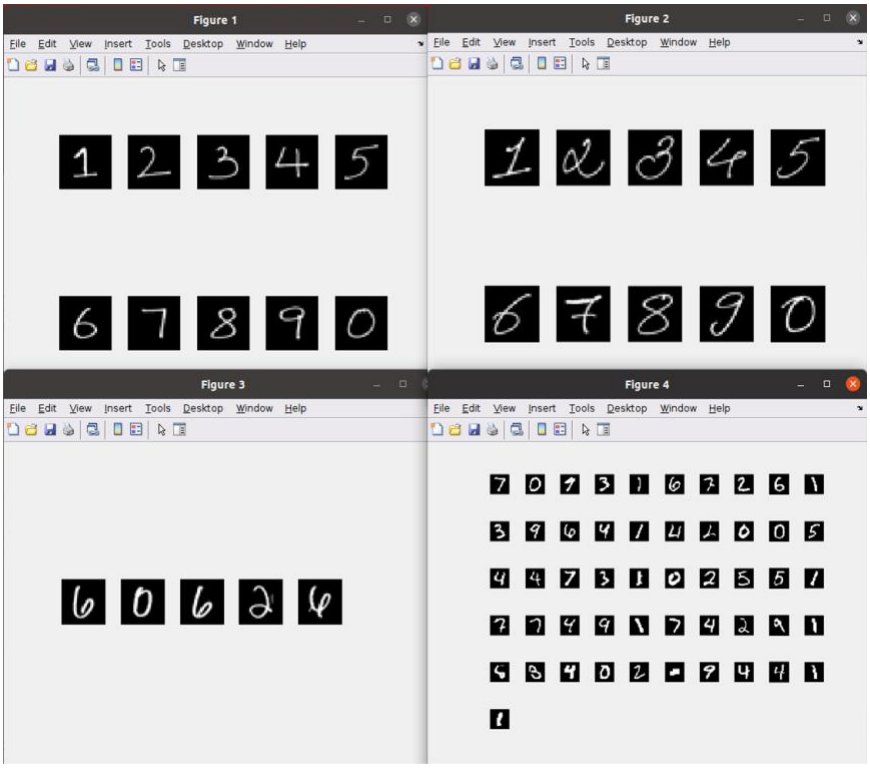
The classification results of the annotated images are as follows (each cell is in the format of predicted(actual)).

```
>> ec
1(1), 2(2), 3(3), 4(4), 5(5), 6(6), 7(7), 8(8), 9(9), 0(0),

1(1), 2(2), 3(3), 9(4), 5(5), 5(6), 7(7), 8(8), 7(9), 0(0),

6(6), 0(0), 6(6), 2(2), 4(4),
7(7), 0(0), 7(9), 3(3), 1(1), 6(6), 7(7), 2(2), 6(6), 1(1),
3(3), 4(9), 6(6), 4(4), 1(1), 4(4), 2(2), 0(0), 0(0), 5(5),
4(4), 4(4), 1(7), 3(3), 1(1), 0(0), 2(2), 5(5), 5(5), 1(1),
7(7), 3(7), 4(4), 9(9), 1(1), 7(7), 4(4), 2(2), 9(9), 1(1),
6(5), 3(3), 4(4), 0(0), 2(2), 4(-1), 8(9), 4(4), 4(4), 1(1),
8(1),
```

Real case accuracy: 86.7%



The trained model does a good job detecting hand-written characters, however, in some cases it still is not able to capture the correct number. One such example is mis-interpreting number 4 in image 4 with 9. It also mis-classifies in Images 4, 1 as 7 and 7 as 9. However, it captures the numbers in the third image, all correctly. One point of error can be the different approaches of normalizing the images as the pre-processed MNIST data comes with in comparison to what I have done (padding and resizing). The other point is that some numbers are really hard to read, even for myself, in comparison to nice well-formatted MNIST numbers. That's why the accuracy is about 10% lower in this experiment.

Test Component