CMPT 412

Project 5

Digit Recognition with Convolutional Neural Nets

Using 1 free late day

3.1 Training

In this part, all the pieces of the network are put together and a pretrained model is trained for 3000 more iterations. I was able to reach a training accuracy of 99% and a test accuracy of 97% after 3000 iterations with a loss of 0.049833. Below is a screenshot of this result.

```
iter =

        3000

cost = 0.049833 training_percent = 0.990000
test accuracy: 0.970000
```

At an earlier iteration, namely, 2500, I reached an accuracy of 100%, however, due to the assignment requirements of 3000 iterations, I used the 3000 iteration model throughout the project. The network weights are saved in lenet.mat and can be found inside the matlab folder.

3.2 Test the Network

After testing the model using the modified test_network.m script, I arrived at the confusion matrix below.

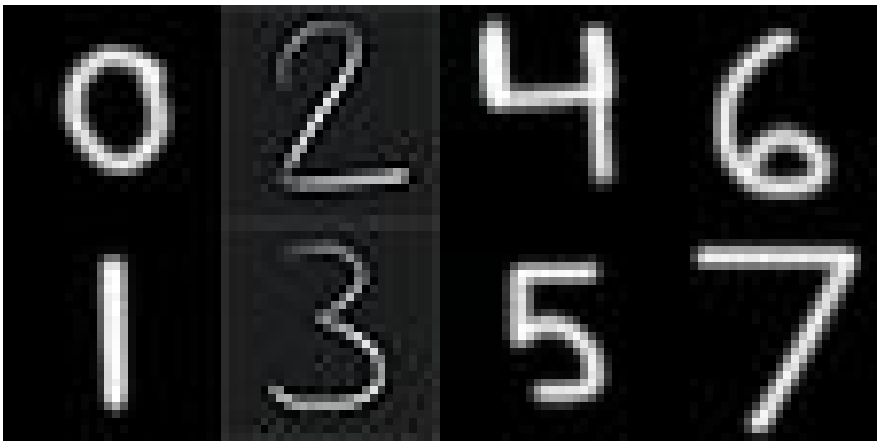
```
C =

    48     0     0     0     0     0     0     0     0     0
     0    57     0     0     0     0     0     0     0     0
     0     0    57     1     0     0     0     2     0     0
     0     1     0    47     0     0     0     0     1     0
     0     0     0     0    38     0     1     0     1     1
     0     0     1     1     0    45     0     0     0     0
     0     0     0     0     0     0    60     0     0     0
     0     0     0     0     0     0     0    45     0     0
     0     0     1     1     1     0     1     0    43     0
     0     1     0     0     0     0     0     0     1    44
```

As we can see, the majority of the entries are on the diagonal, which is exactly what a good model should demonstrate. We can see that 7 has been classified incorrectly as a 2, two times. This makes sense because 7 looks similar to a 2 in some sense. Another confused

pair is 3. We can see that it was classified as a 2, 5, and 8. These are the most similar numbers to a 3 in terms of shape. For example, 3 looks like half of an 8. Moreover, 2 and 5 are also quite similar to 3, which justifies this confusion of the system.

3.3 Real-World Testing

In this section we are going to demonstrate the performance of the system under real world scenarios. I created some hand-written digits, namely numbers from 0 to 7 inclusive. For the numbers 2 and 3, I used an iPhone's touch screen to create the digits. For the rest of the numbers, I used Adobe Photoshop on my laptop and created the digits using the brush tool. I then cropped and resized all the images using Adobe Photoshop such that they are 28x28 pixels. I then fed them to the network and the result was an amazing 100% accuracy for all 8 digits. Below are the sample images used for testing. As you can see, 2 and 3 look different as they were created using a different medium. This was done in order to test the performance under different conditions.



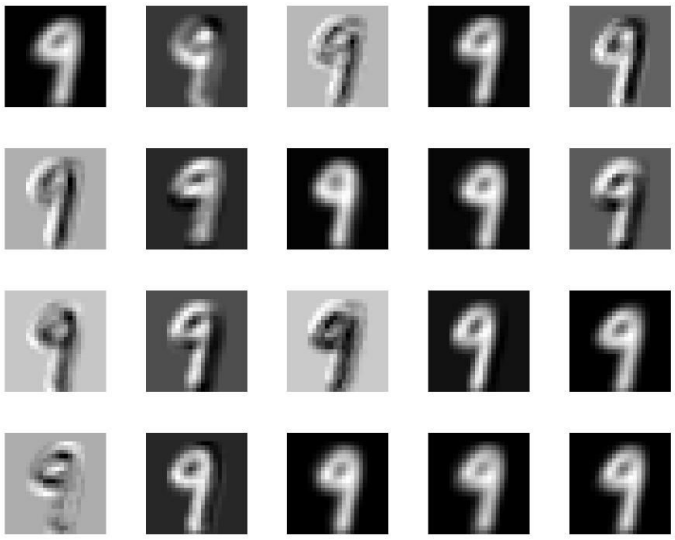
Below is the result generated from the realWorldTesting.m script which feeds the above images to the network.

```
>> realWorldTesting
Predictions are below (should be 0 1 2 3 4 5 6 7):
     0     1     2     3     4     5     6     7
```

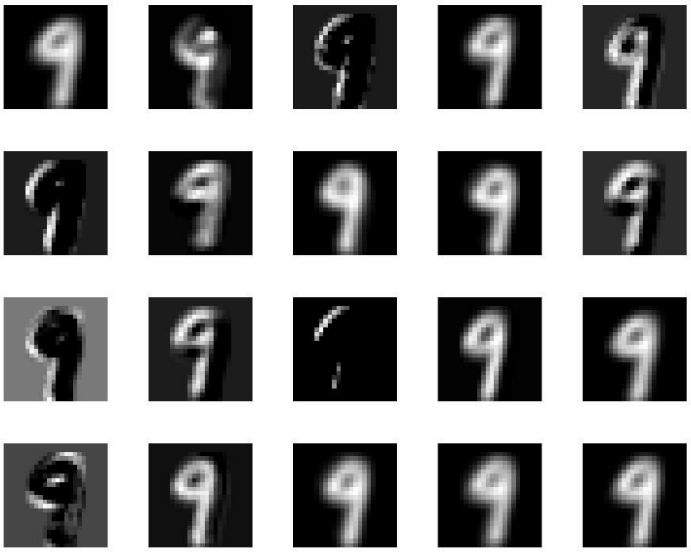
4.1 Visualization

In this section, we are going to visualize the output of the second and third layers of the network, namely the Conv and ReLU layers. Below are the outputs organized in 4x5 format.

Layer 2 (Conv)



Layer 3 (ReLU)



4.2 Discussion

At first glance, we can see that the digit is still recognizable and is clearly a 9, even after 2 layers of processing. We can also see that the main difference between the two outputs is that ReLU seems to have suppressed many of the pixel values. This makes sense, because the ReLU function is defined such that it suppresses values smaller than 0 and keeps the values above 0 as they are. That is the reason we have most of the backgrounds set to black in the ReLU layer after mapping and scaling values to 0-255 range for visualization. In addition, we can see that individual outputs, look less detailed with some information loss. The Conv layer on the other hand, seems to have optimized and developed filters in an abstract way which are hard to explain precisely and understand their purpose in the overall network. A few of the outputs look identical or at least very similar to the naked eyes. However, we can see that after the image has passed through both layers, some of the filters seem to have developed similar to some famous filters. For example, a sobel-like filter seems to have emerged and is detecting edges in some sense. Some even look like they are detecting vertical lines, as if they are the x gradient sobel filter! In general, each filter has been optimized by the system in such a way as to detect distinct features in the image by breaking it down, suppressing some parts, and highlighting other parts. This is the power of convolutional neural networks. We do not need to figure out the best combination of filters to apply or what features to detect for a specific task; we let the system do it for us and learn the best combination. It does not even have to make sense to us humans as long as it makes sense to the machine!