# Handwritten digits recognition project report

## 2.1 Define the network

In this section, I completed the forward function. The x is our input image, which I expect to be the shape (784,). The W1x is going to be matrix multiplication of our input x and weight 1 W1. While a1 is W1x plus the bias we might have. Then we apply the sigmoid function to receive the value of f1.

Now, when it comes to W2x, it needs to be matrix multiplication of our weight 2(W2) with f1. And the a2 is just W2x plus the bias 2. Lastly, in order to get the y_hat, we need to apply softmax to our a2 to get our y_hat.

## 2.2 Initialize the network

I first initialized my weight 1 with random Gaussian variables with a standard deviation 0.1, shape of W1 is (64,784). And use the same process initialized weight 2 with random Gaussian variables with a standard deviation 0.1, shape of W1 is (10,64). While for b1 is shape (1,64) with all value zeros, and b2 shape (1,10) with all value zeros.

## 2.3 Compute the gradient

In this section, I computed all the gradients in the update_grad function. One thing that needs to keep in mind is that when we do the matrix multiplication, we need to check for all the variable shapes in order to perform the correct output. Performed transpose or reshape the matrix if it's needed. Incorrect shape computation will not able to train our network properly.

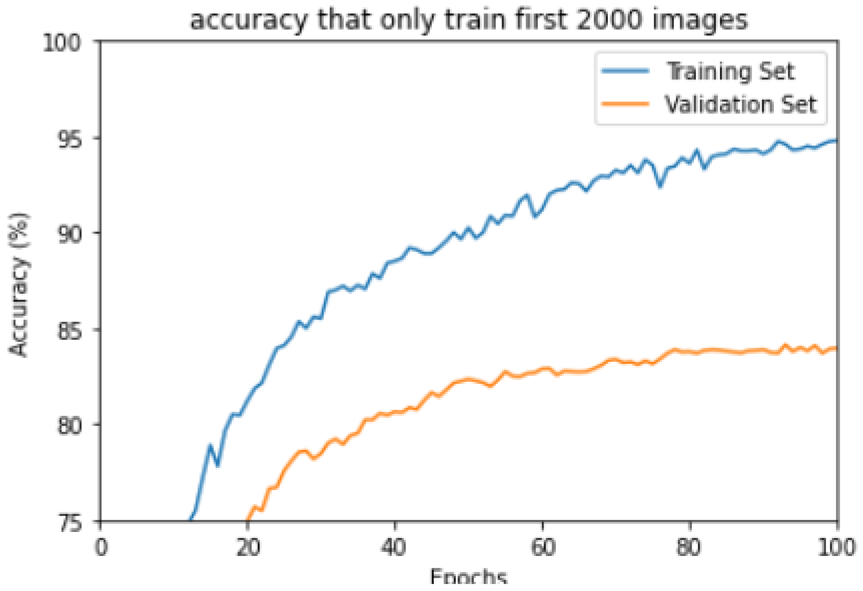
## 2.4 Training

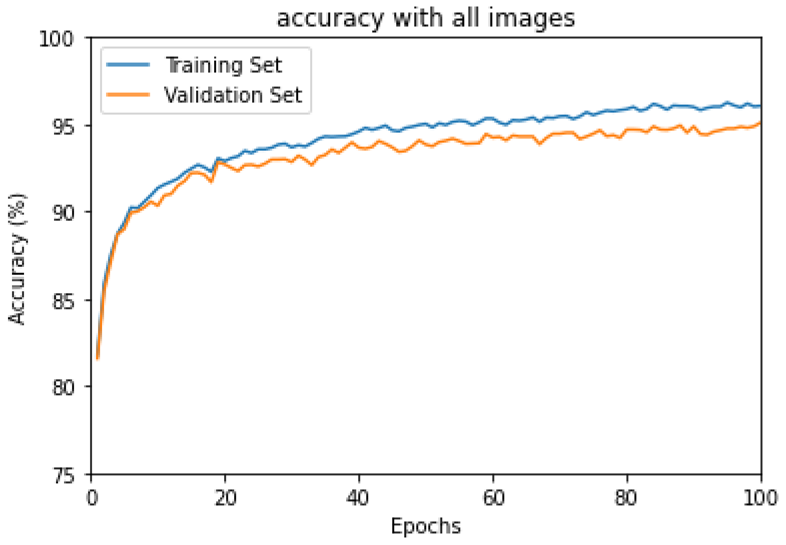
In this section, I trained the network using stochastic gradient descent with a batch size of 256, a cross-entropy loss function, and a learning rate of 0.001.

## 2.5 Over-fitting?

We first trained our network with only the first 2000 images from the training dataset, and at the end of each training epoch, I computed the accuracy over the entire 2000 training set and validation set. The first picture is the graphic representation of this accuracy with only trained by 2000 images.

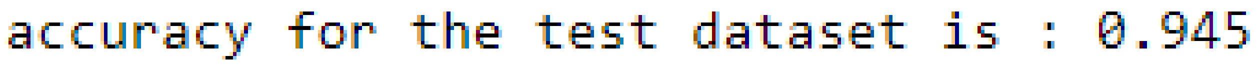
Later, we trained our network with the entire 50000 images training set with the same process described above. We see that the accuracy rate is much higher than only training with 2000 images. We can conclude that with more training data, our network performs much better prediction with identifying handwritten digit images.





## 2.6 How well does it work?

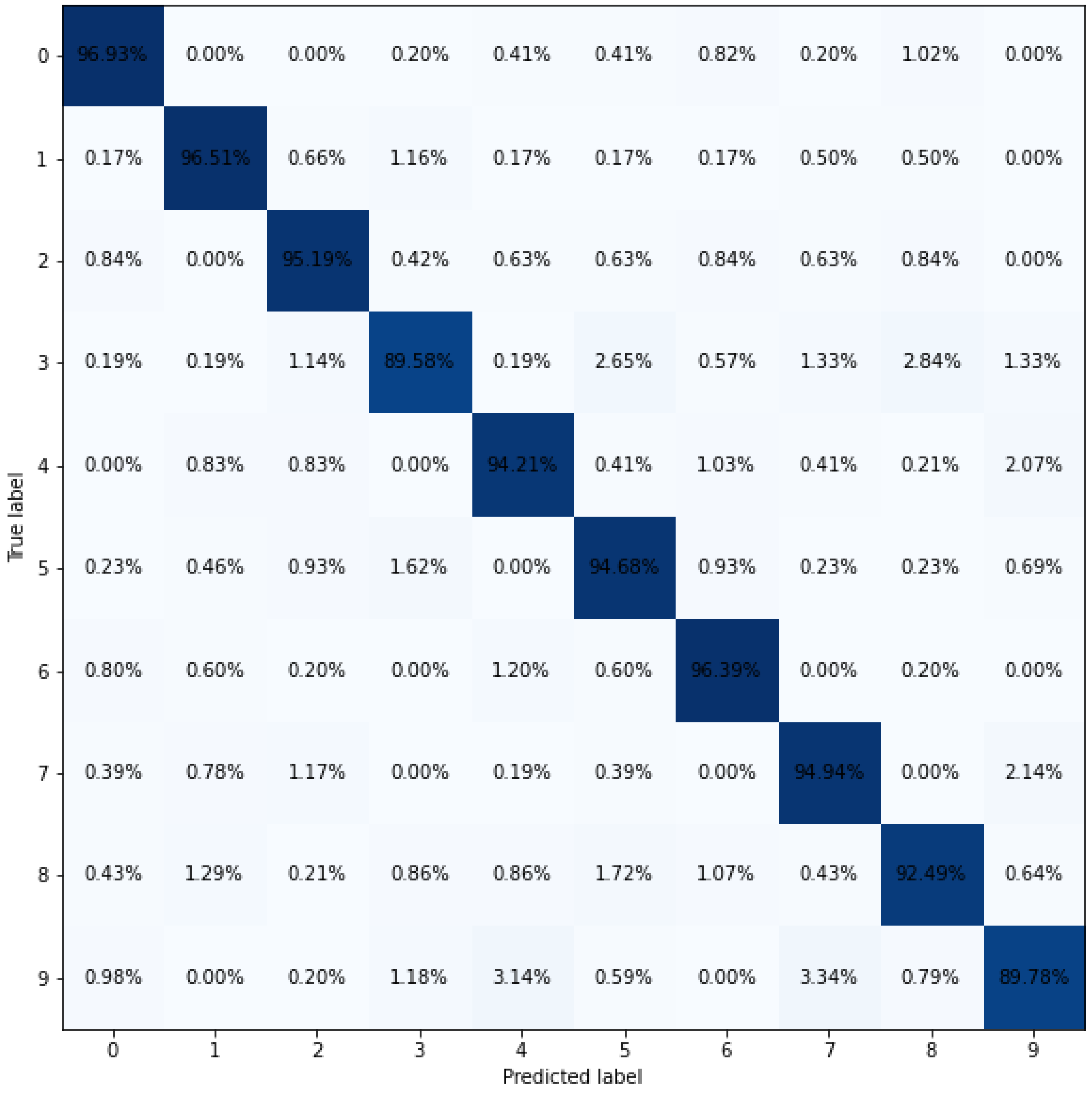
In this section, I trained the network with the training dataset, and test it with the test dataset, the accuracy I got is 0.945, the improvement I did to increase the accuracy is to adjust for an adaptive learning rate, which I found is 0.0005.



```
accuracy for the test dataset is : 0.945
```

## 2.7 Where does it make mistakes?

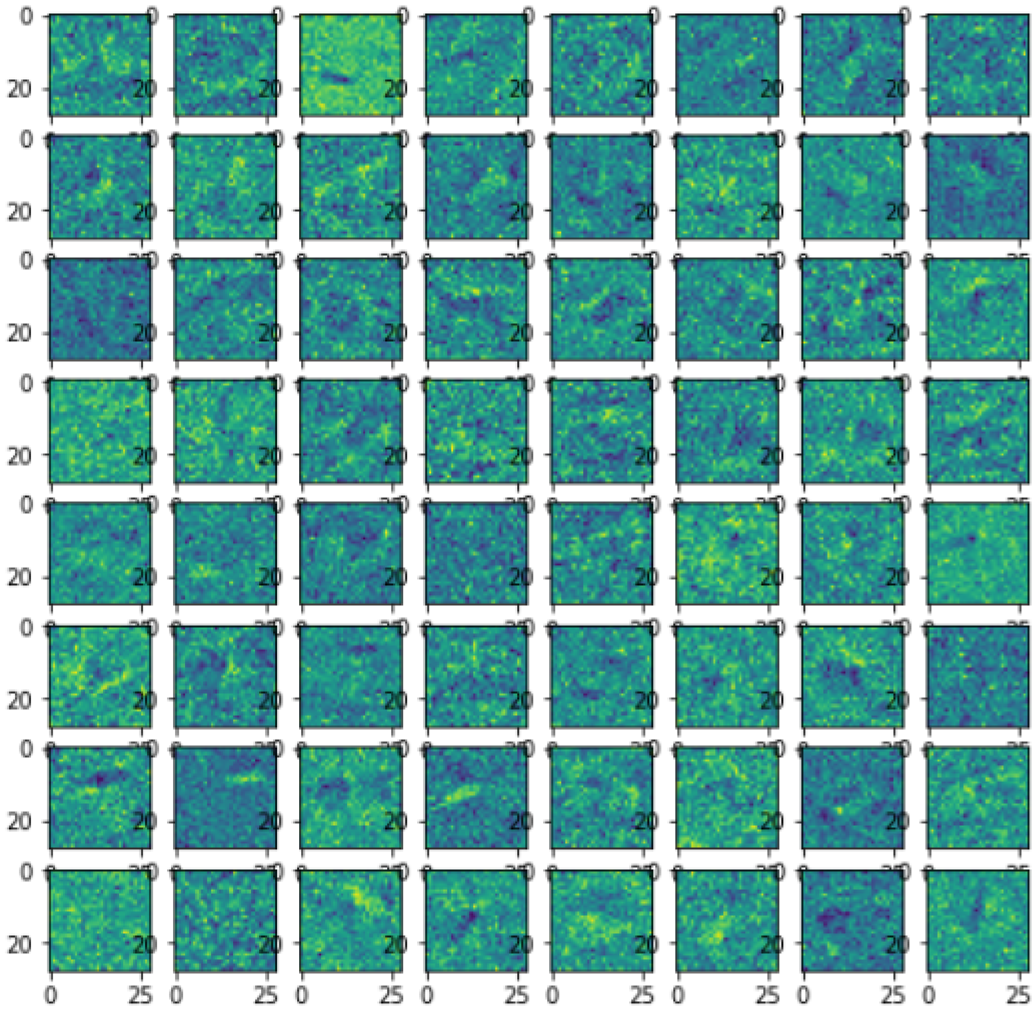
Here is a 10 by 10 confusion matrix. As you can see the overall performance of the network works pretty well. However, number 9 seems is difficult to identify. Confusion matrix note: the darker the color, the higher the accuracy rate.



| True label \ Predicted label | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 96.93% | 0.00% | 0.00% | 0.20% | 0.41% | 0.41% | 0.82% | 0.20% | 1.02% | 0.00% |
| 1 | 0.17% | 96.51% | 0.66% | 1.16% | 0.17% | 0.17% | 0.17% | 0.50% | 0.50% | 0.00% |
| 2 | 0.84% | 0.00% | 95.19% | 0.42% | 0.63% | 0.63% | 0.84% | 0.63% | 0.84% | 0.00% |
| 3 | 0.19% | 0.19% | 1.14% | 89.58% | 0.19% | 2.65% | 0.57% | 1.33% | 2.84% | 1.33% |
| 4 | 0.00% | 0.83% | 0.83% | 0.00% | 94.21% | 0.41% | 1.03% | 0.41% | 0.21% | 2.07% |
| 5 | 0.23% | 0.46% | 0.93% | 1.62% | 0.00% | 94.68% | 0.93% | 0.23% | 0.23% | 0.69% |
| 6 | 0.80% | 0.60% | 0.20% | 0.00% | 1.20% | 0.60% | 96.39% | 0.00% | 0.20% | 0.00% |
| 7 | 0.39% | 0.78% | 1.17% | 0.00% | 0.19% | 0.39% | 0.00% | 94.94% | 0.00% | 2.14% |
| 8 | 0.43% | 1.29% | 0.21% | 0.86% | 0.86% | 1.72% | 1.07% | 0.43% | 92.49% | 0.64% |
| 9 | 0.98% | 0.00% | 0.20% | 1.18% | 3.14% | 0.59% | 0.00% | 3.34% | 0.79% | 89.78% |

## 2.8 Visualize the weights

Here is the visualize of the weights:



## 3.1 Describe the network architecture

The first convolution layer has the size of the input channel of 1, and the output channel is 6, and the kernel size is 5. And for the self.pool has a kernel size of 2 and stride size of 2. The second convolution layer has the input channel size of 6, output channel 16, and kernel size 5. Then the fully connected layer has the input feature size of 16 * 4 * 4, and the output size of 120. The second fully connected layer has the input feature of 120, and the output feature of 84. Lastly, the final fully connected layer has the input feature of 84, output feature of 10(because we only have 10 digits output). All the padding of the kernel is 0.

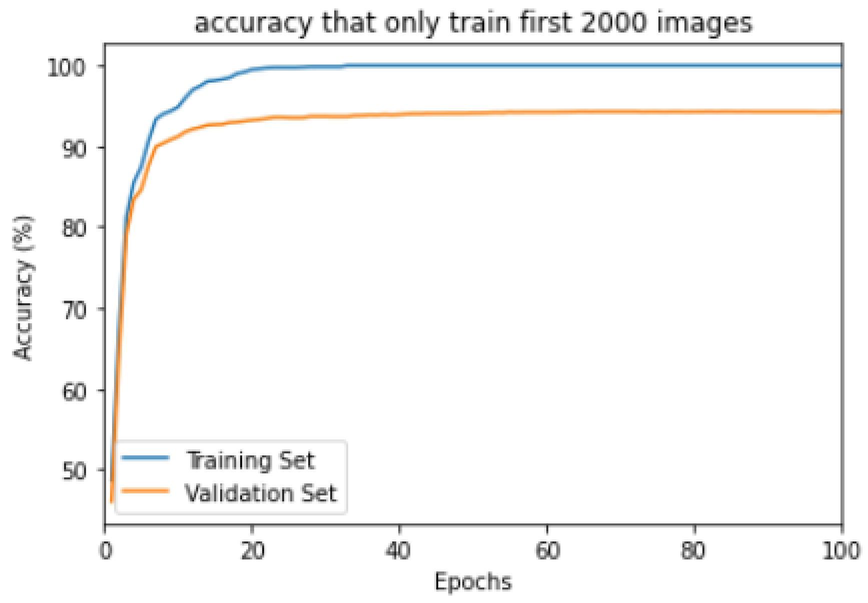
## 3.2 Train CNN

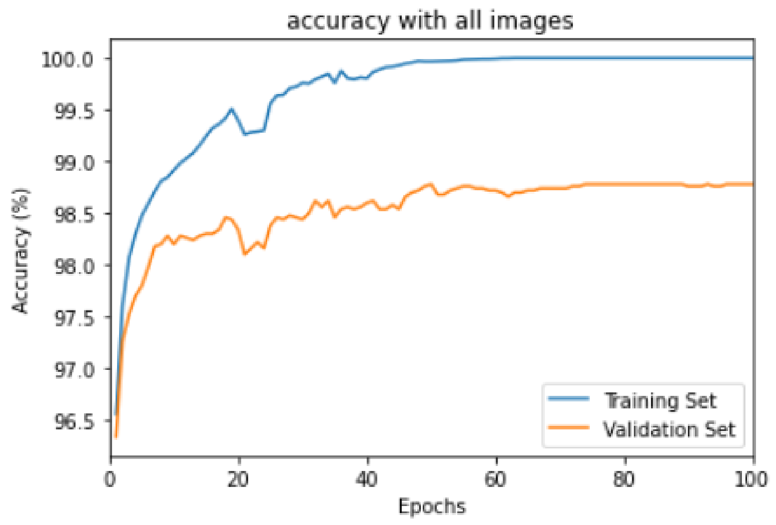
In this section, I trained the CNN using stochastic gradient descent with a batch size of 256, a cross-entropy loss function, and a learning rate of 0.001.

## 3.3 Over-fitting?

This is the same process as section 2.5, except that I use PyTorch CNN instead. I first trained our network with only the first 2000 images from the training dataset, and at the end of each training epoch, I computed the accuracy over the entire 2000 training set and validation set. The first picture is the graphic representation of this accuracy with only trained by 2000 images. The accuracy for the training set is 100%, while the accuracy for the validation set is slightly lower.

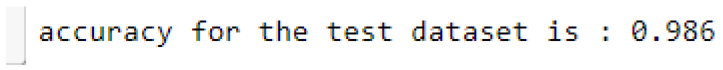
Then, I trained our network with the entire 50000 images training set with the same process described above. We see that the accuracy rate is increasing over time. We can conclude that with more training data, our network performs much better prediction on the validation set with identifying handwritten digit images toward the end.





## 3.4 How well does it work?

In this section, I trained the network with the training dataset, and test it with the test dataset. The accuracy I got is 0.986. Compare to section 2.6. PyTorch CNN gives much better permanence.



```
accuracy for the test dataset is : 0.986
```