CSCI 420 Assignment 3 Report

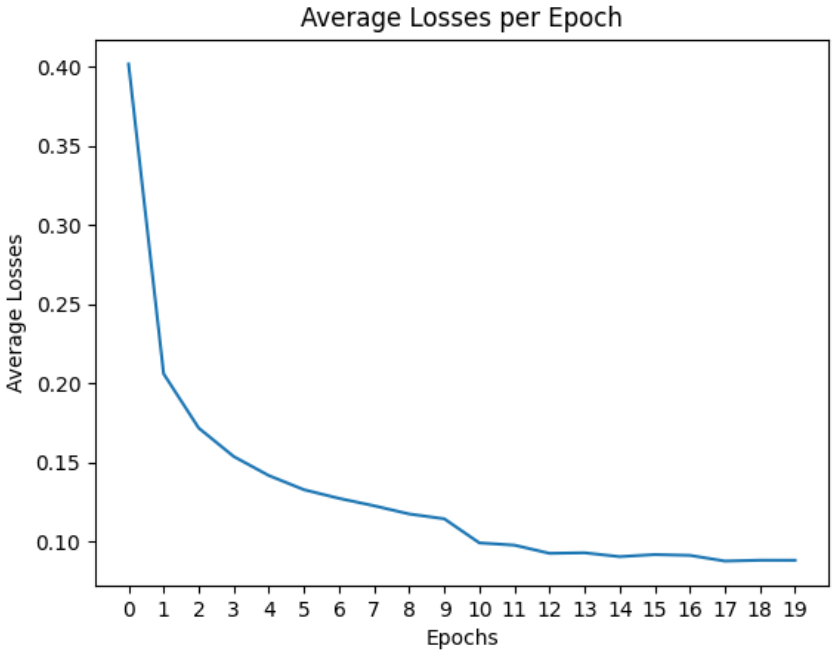
Joanne Lee (ehlee@email.wm.edu) & Justin Park (jmpark@email.wm.edu)

Implementation

- The CNN model code is below where we created the 3 layers. The first layer input is just 1 for a non-colored image. The output is the channel_out1 arg from the main.py arguments. Then we added the ReLU activation function and the dropout to complete the first layer. The second layer input is the output of the first layer and the output for the second layer is the channel_out2 arg. The input for the third layer is the output from the second layer and we chose the output for the third layer to be 10. The kernel size and stride were also taken from the args in main.py

```
self.conv = nn.Sequential(nn.Conv2d(1,args.channel_out1, kernel_size=args.k_size, stride=args.stride),
                          nn.ReLU(),
                          nn.Dropout(args.dropout),
                          # second layer
                          nn.Conv2d(args.channel_out1, args.channel_out2, kernel_size=args.k_size,stride=args.
                          nn.ReLU(),
                          nn.Dropout(args.dropout),
                          # third layer
                          nn.Conv2d(args.channel_out2, 10, kernel_size= args.k_size, stride= args.stride),
                          nn.ReLU(),
                          nn.Dropout(args.dropout)
```

- The final fully-connected layer input we got by finding the dimensions of the layer after we flattened it in the forward function. We defined the fully-connected layer with the dimensions (3610, 10).
- The batch size that we used was the one given in the args in main.py which was defaulted to 100.
- To compute the accuracy, we took the number of correct identifications over the total number of y_labels and multiplied by 100 to get a percentage.
- To calculate the average losses, we made a losses array in each epoch to store the losses with loss.items() and summed up the losses over the number of losses in each epoch. Then we appended it to another array to store each average loss per epoch which we used to graph below.
- Some of the one-line code implementations are explained through comments in the code




Experimental Results:

- We experimented with different numbers of epochs, as well as different number of channels. The test accuracy that we ended up getting after running the code with 20 epochs is 98.39% as shown below.

```
ehlee@bg6:~/cs420$ python3 main.py
cuda
data_path:  ./data/
Epoch: 0, Loss: 0.4321151295552651, Accuracy: 85.77166666666668
Epoch: 1, Loss: 0.22195766703536113, Accuracy: 92.76333333333334
Epoch: 2, Loss: 0.17899663153414924, Accuracy: 94.27833333333334
Epoch: 3, Loss: 0.15591937203270695, Accuracy: 95.1
Epoch: 4, Loss: 0.14060742492166659, Accuracy: 95.55166666666666
Epoch: 5, Loss: 0.13193726035145423, Accuracy: 95.79833333333333
Epoch: 6, Loss: 0.12386210285437604, Accuracy: 96.07833333333333
Epoch: 7, Loss: 0.12130469704978168, Accuracy: 96.13000000000001
Epoch: 8, Loss: 0.1173570070322603, Accuracy: 96.315
Epoch: 9, Loss: 0.11153137181264659, Accuracy: 96.41999999999999
Epoch: 10, Loss: 0.09624145222672571, Accuracy: 96.91
Epoch: 11, Loss: 0.09253252113393197, Accuracy: 97.05333333333334
Epoch: 12, Loss: 0.09166418750149509, Accuracy: 97.04
Epoch: 13, Loss: 0.0902548597784092, Accuracy: 97.16166666666666
Epoch: 14, Loss: 0.09007448458112777, Accuracy: 97.12666666666667
Epoch: 15, Loss: 0.08756491473720719, Accuracy: 97.17333333333333
Epoch: 16, Loss: 0.08730852990256001, Accuracy: 97.15166666666667
Epoch: 17, Loss: 0.08744956871649871, Accuracy: 97.30499999999999
Epoch: 18, Loss: 0.08592868205004682, Accuracy: 97.195
Epoch: 19, Loss: 0.08565845651241641, Accuracy: 97.26333333333334
Testing Accuracy: 98.39
running time:  1.4097732265790304 mins
ehlee@bg6:~/cs420$
```