Homework #3: MNIST classification

Andrew Mueller
2072876

Repository: https://github.com/sansseriff/caltech-ee148-spring2020-hw03

5. Default ConvNet

The accuracy of the default convolutional network was not improved significantly by training it with augmented data. Various augmentation steps were implemented with various degrees of success. Slight changes to the dataset like random crops spanning 2 by 2 pixels seems to improve accuracy on the validation set slightly (from 97.06 to 97.18% accuracy), however almost all more aggressive augmentation schemes reduced accuracy. It is possible the default convolutional network is not large enough to achieve higher accuracy by learning on the effectively larger augmented dataset.

A rather aggressive augmentation scheme was used for the development of the Net() network later on, involving an affine transform with a random angle adjustment of +/-8 degrees, x and y translation of +/- 2 pixels, and sale adjustment between 80 and 110%. The augmentation, when used to train the default ConvNet() resulted in much lower accuracy (from 97.1% to 88.7%). However, when used to train the larger Net() network, this augmentation helped avoid overfitting and added about a half of a percent of accuracy.

6. Best MNIST classifier with Net() network

The fact that augmentation was having an adverse effect on accuracy in the testing of ConveNet() suggested a larger network would be more effective. However, adding more fully connected layers with the same size as following layers did not increase accuracy substantially. So instead of simply adding layers to increase depth, I added a layer of convolution kernels and turned off some of the MaxPools, resulting in a much larger 1st fully-connected layer (15488 inputs and 3000 outputs). With that larger layer, I found that directly connecting it to a much smaller fully connected layer like that right before the final 10 neurons resulted in only a marginal increase in performance. Therefore, I added a sequence of progressively smaller fully connected layers between the last layer and the first fully connected layer. This progression to smaller sizes resulted in the best performance.

Without dropout layers or data augmentation, this large network overfit the training set easily as shown in figure 1a. The data augmentation used to train the Net() network was the affine transformation described above. Dropout layers were used after the first two convolutions, and their dropout probabilities were tuned to avoid overfitting but allow reasonable training convergence time. The convergence shown in figure 1b was achieved with dropout probabilities of 0.4.

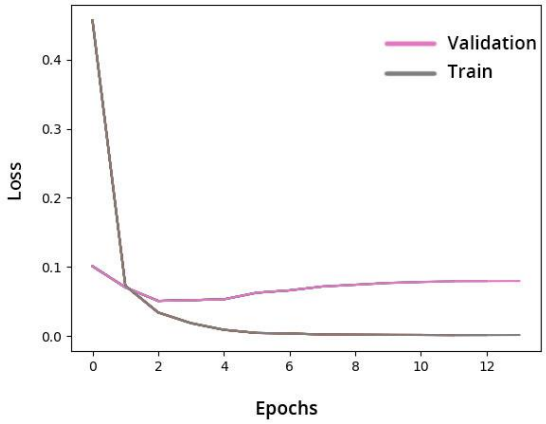


Figure 1a Validation and Train Loss as a Function of Epoch. Overfitting with un-augmented data

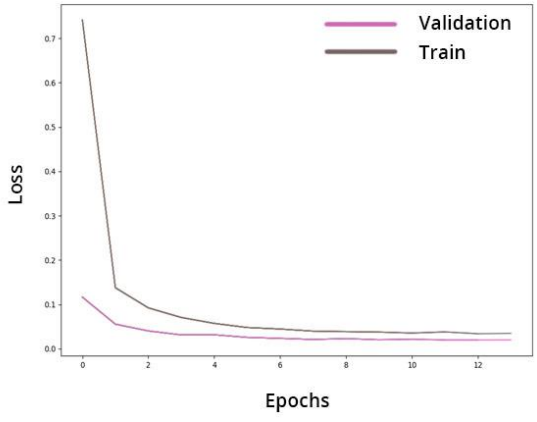


Figure 2b Validation and Train Loss as a Function of Epoch. Overfitting with un-augmented data

7. Loss of Net() network with varied number of train images

Final accuracy on train set: 8944/8995 (99.4330%)

Final Accuracy on the test set: 9936/10000 (99.3600%)

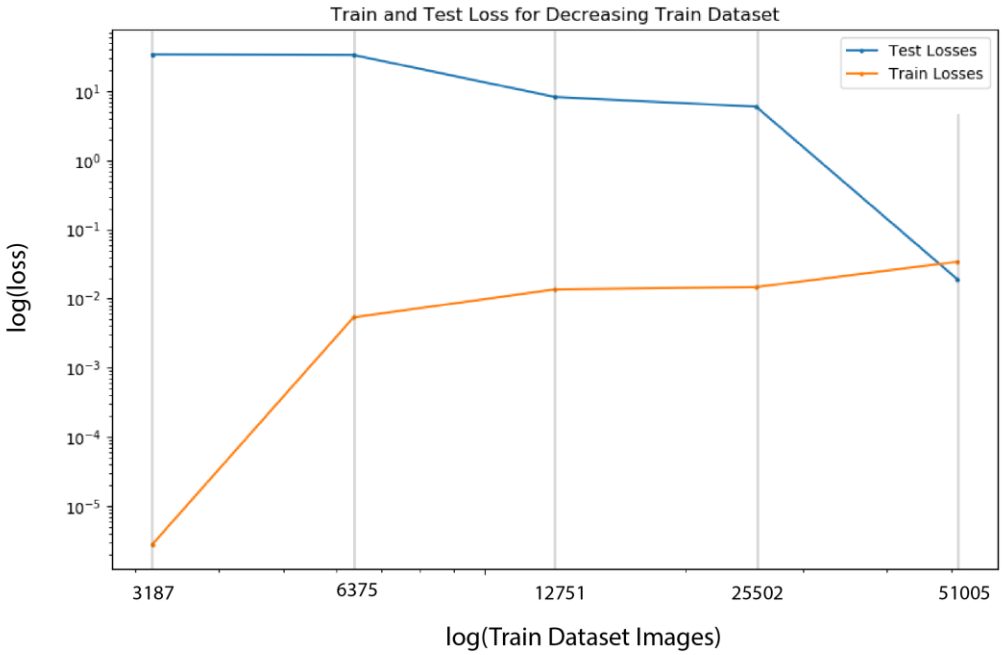


Figure 2 loss as a function of image number used to train network. For all training sets smaller than the original set, the network overfits the data, even with augmentation

Figure 2 shows the tendency of my model to overfit the train data as the size of the train dataset decreases.

8. Network Analysis

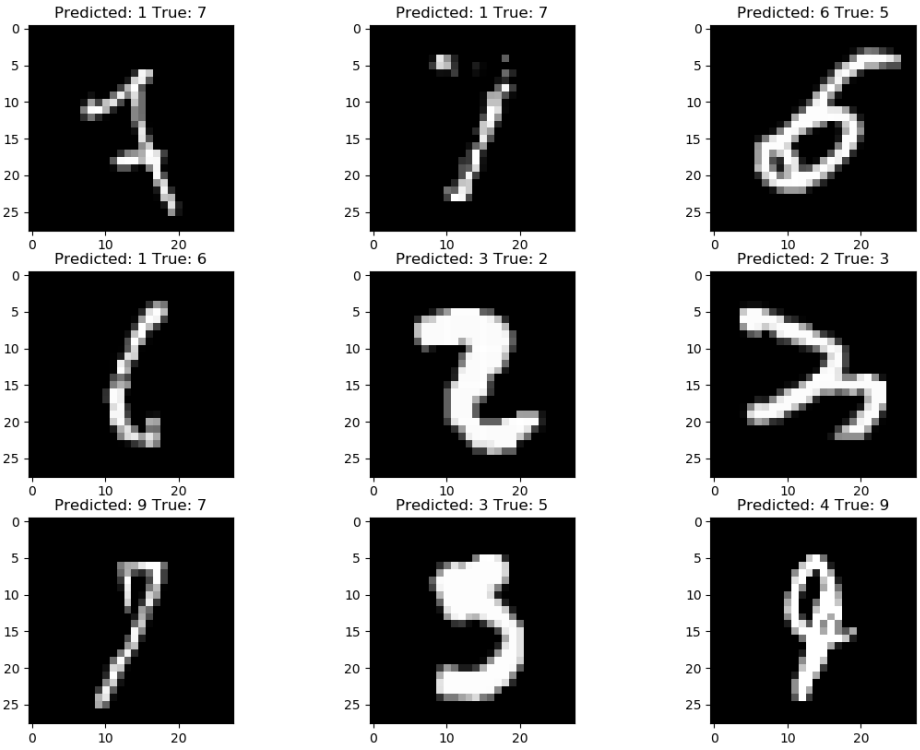


Figure 3: 9 images for which Net() failed to correctly identify numbers

The numbers the network labeled incorrectly could have easily been mislabeled by a human as well.

A couple observations:

- To most people, the image in the upper left-hand corner would appear to show a 7, but only because the cross halfway down. If it wasn't for this cross, it would look entirely like a 1, as predicted by the network. Since not many people cross their sevens, it's likely there were not enough examples of this choice in the training set, so the network wasn't well trained to recognize it.
- Often, just a few pixels allow a human to identify the correct label when the network could not. This is evident in the 6 misidentified as a 1 and the seven (top-middle) identified as a one.

First layer kernels:

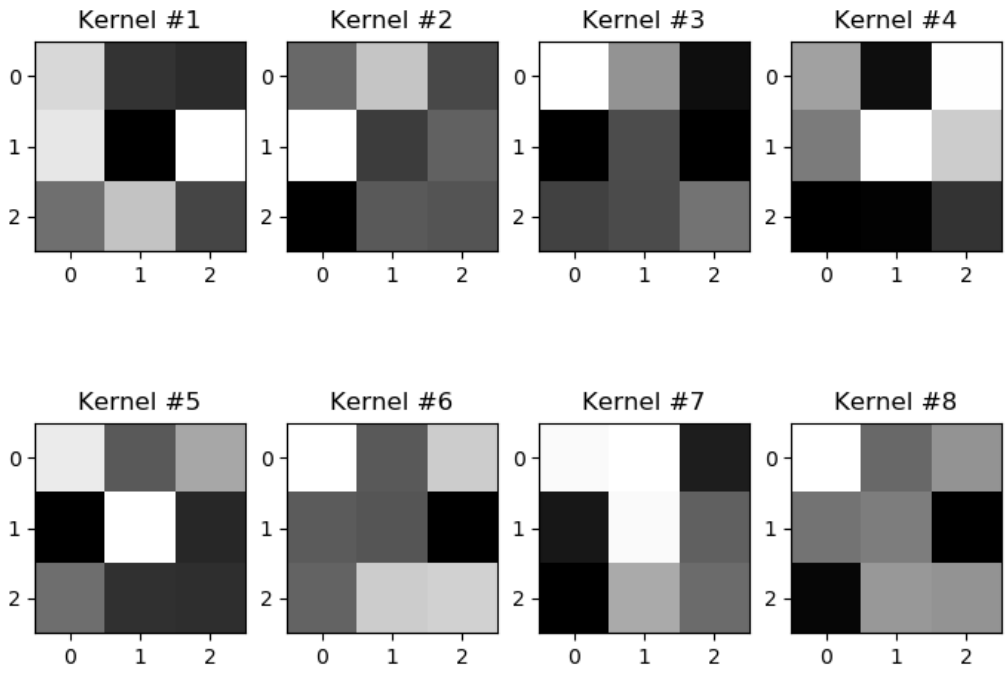
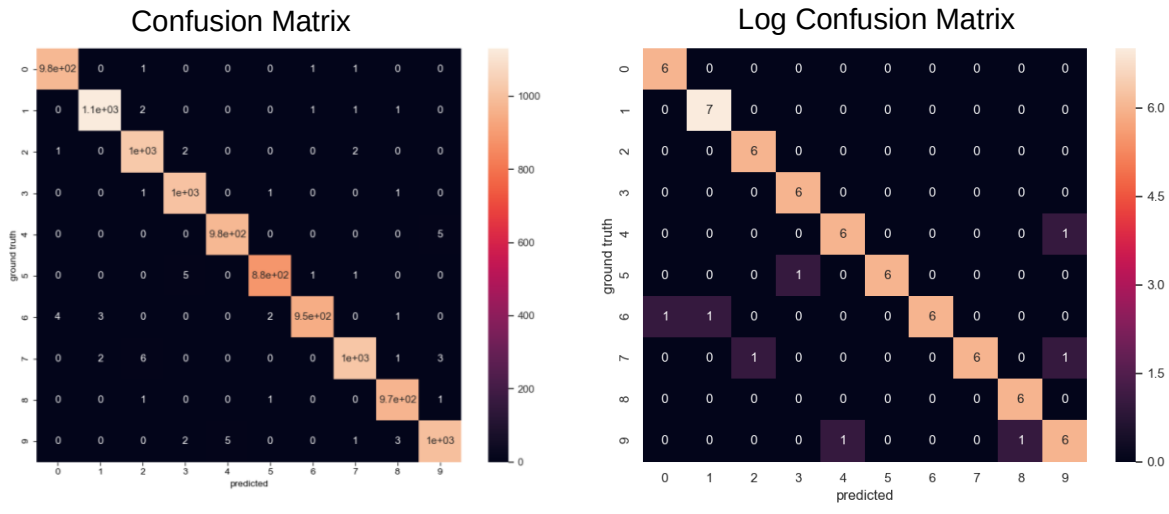


Figure 4: 9 images for which Net() failed to correctly identify numbers

Several of these kernels have two dark pixels separated by a single or a few lighter pixels. That suggests they are activated by lines at different angles, or certain kinds of curves. For example, kernel #7 applied to a sharp left facing corner like that in a handwritten 7 might result in a large activation.

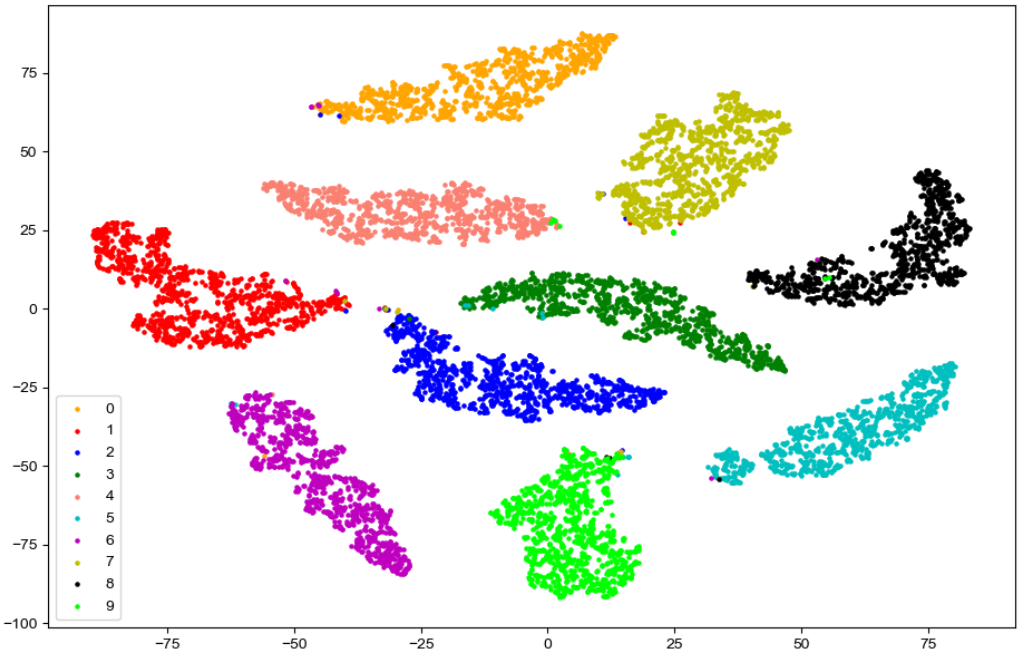
There are also no two kernels that seem too similar, suggesting at least this layer is not significantly larger than it needs to be.

Confusion Matrix:



The Log Confusion matrix is generated by taking the natural logarithm of all nonzero terms in the confusion matrix. It more clearly visualizes what written numbers the network confuses with each other. The numbers most often confused with each other are 5s and 9s. On the test set, Net() mistook 5s for 9s just as often as it mistook 9s for 5s. Net() mistook some 5s for 3s but not vice versa. This probably happened whenever the top half of the 5 was compressed into a single blob of ink, as we seen in the middle-bottom example in figure 3.

Feature vectors and tSNE:



The tSNE view shows how different written numbers are grouped into sets the network can classify. A few miscategorized numbers can be seen. Interestingly, miscategorized points can be found in groups a significant spatial distance from the groups to which they belong. For example, the physical distance between the green group (9) and the pink group (4) is large but green dots can still be seen in the pink group.

Distance Analysis:

The first image of each vertical set is x0, and the following images are the closest in Euclidian distance to x0. The images look similar as you would expect. The main variations between members of the set are tilt angle and line thickness.