COMPSCI 4ML3

Introduction to Machine Learning

Winter 2021

Assignment Four

Tahseen Ahmed

ahmedt26

McMaster University

Revised: April 10th, 2021

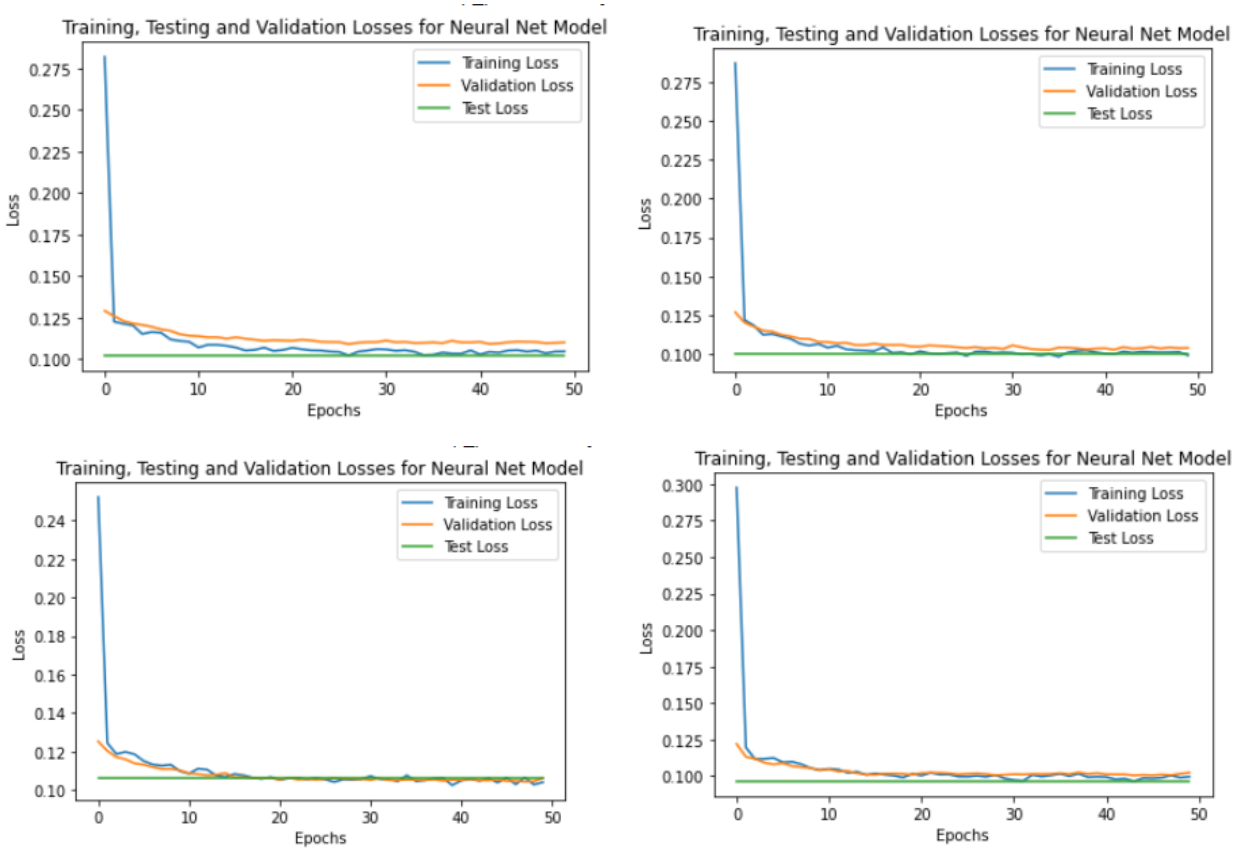
Hello, and welcome to my submission for assignment four. Below you will find my report on my venture into neural networks.

1. **Programming Component: Task One**

We create the function plot_eval_results to plot the training, validation and test losses of our neural net. Here are several outputs of each run of the model:

```
Test loss is 0.11262481839983327 for drop_probability=0.7, epochs=50, batch_size=64, split
Test loss is 0.10924888981185901 for drop_probability=0.7, epochs=50, batch_size=64, split
Test loss is 0.11256843629726179 for drop_probability=0.7, epochs=50, batch_size=64, split
Test loss is 0.10791605649290571 for drop_probability=0.7, epochs=50, batch_size=64, split
```

Each run results in a overall test loss of roughly 0.11. Here's the graphs for each run, from 1 2 3 4 in order of top-left, right and then bottom left and right:



Our best run was the third one. After a little bit of initial underfitting the losses converge, meaning the model working correctly. If validation loss is lower or higher than training loss it indicates under or overfitting respectively. In the majority of the graphs it seems this model bay be overfitting a little bit.

As for number of parameters, we know that these layers are fully connected: each neuron from the previous layer is connected to *every* other neuron in the next layer.

We have two hidden layers of 1000 neurons, and and input/output dimension of 784. In this model, we have bias as well, and these parameters are between the hidden layers, and the last layer and output.

$$1000 * 784 + 1000 * 784 + 1000 + 784 = 1569784$$

We have a total of 1569784 trainable parameters in the neural net. Note that this number usually decreases as training occurs because of dropout. To confirm this, I use a piece of code to get the parameters between the layers, and the bias parameters. The code is found here.

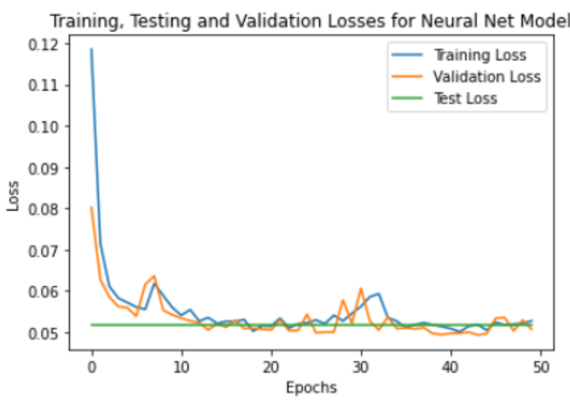
2. **Programming Component: Task Two**

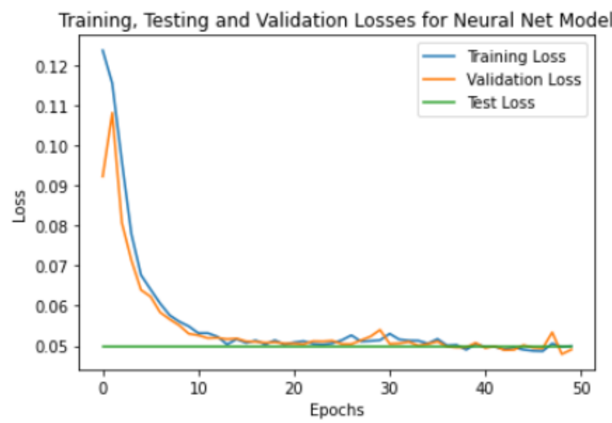
We create create a convolutional neural network for this task and see the differences between the first net.

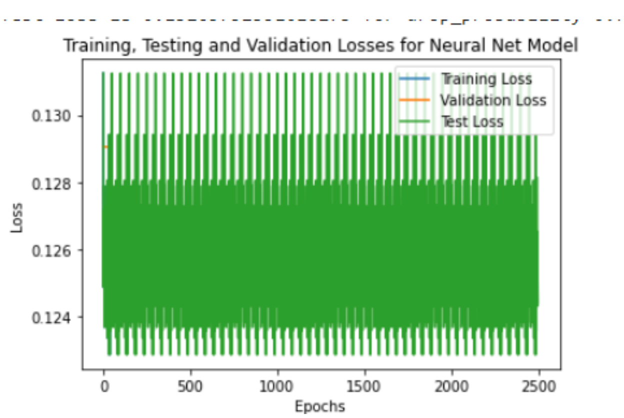
Here's some final outputs from this architecture:

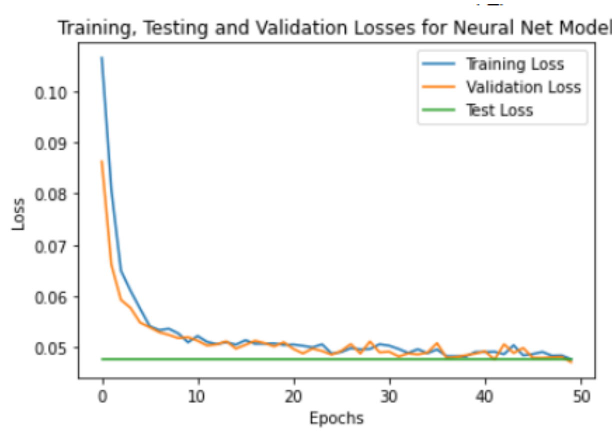
```
Test loss is 0.0518228139514756 for drop_probability=0.7, epochs=50, batch_size=64, split_
Test loss is 0.05020351133718612 for drop_probability=0.7, epochs=50, batch_size=64, split
Test loss is 0.13265752351018273 for drop_probability=0.7, epochs=50, batch_size=64, split
Test loss is 0.04877564208997283 for drop_probability=0.7, epochs=50, batch_size=64, split
```

The graphs are interesting here is the graph for each run









The third run, is quite interesting. The test loss is nearly triple that of the other runs, and the graph might as well be a solid color. We'll ignore it, but this popped up several times when I ran the code so I'd like to report it here.

The training/validation losses jump up and down much more than that of the linear model, but they all seem to converge in the end. This would imply that the MyCNN model might be better than the linear model by about 45 percent.

For number of parameters in CNNs, we use this formula:

$$P = (d^2) * c$$

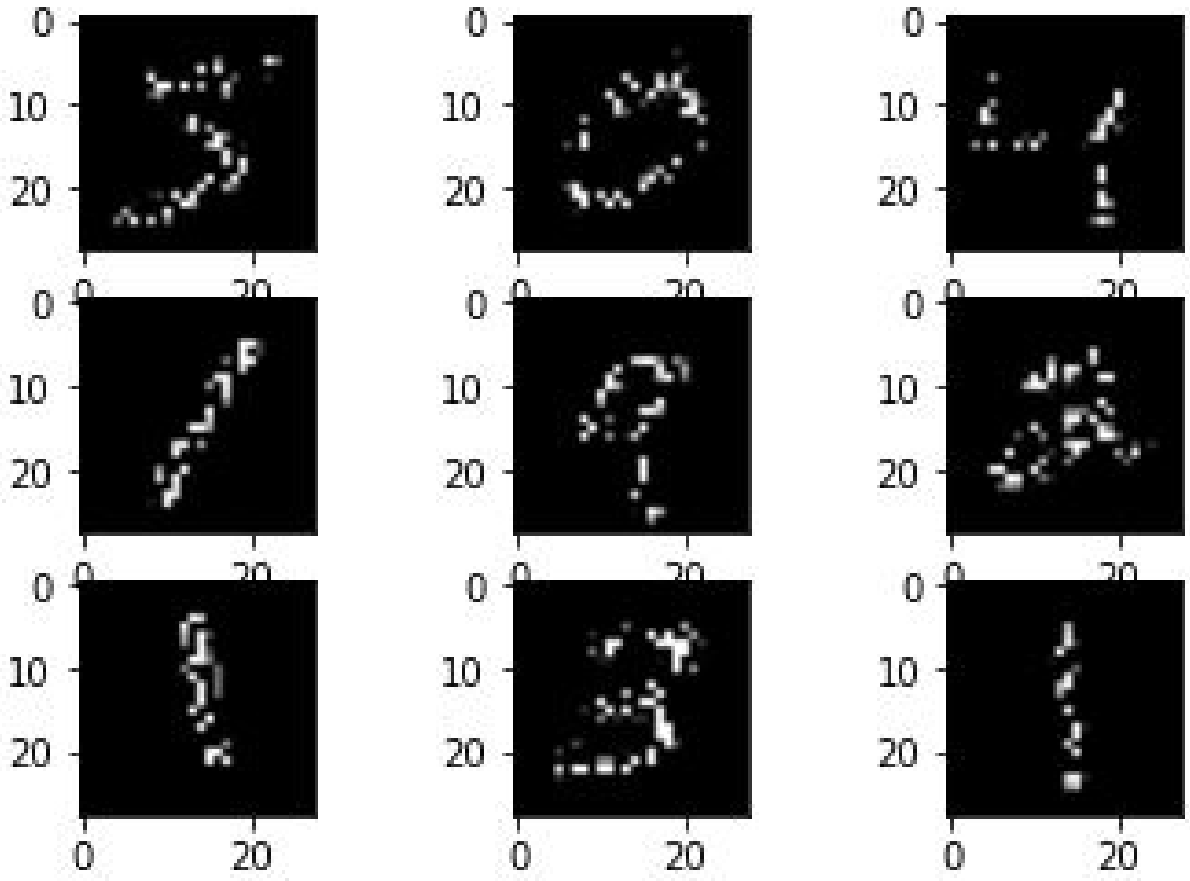
where P is parameters, d is the dimension of the kernel (3 since its square) and c is the number of output channels. We have bias here as well, so the output and input channels each have a parameter, which means 10 bias parameters between the two convolution layers, and then 1 for the last output channel. The formula above does not account for bias values, we would have to add +1 to the d^2 value for this.

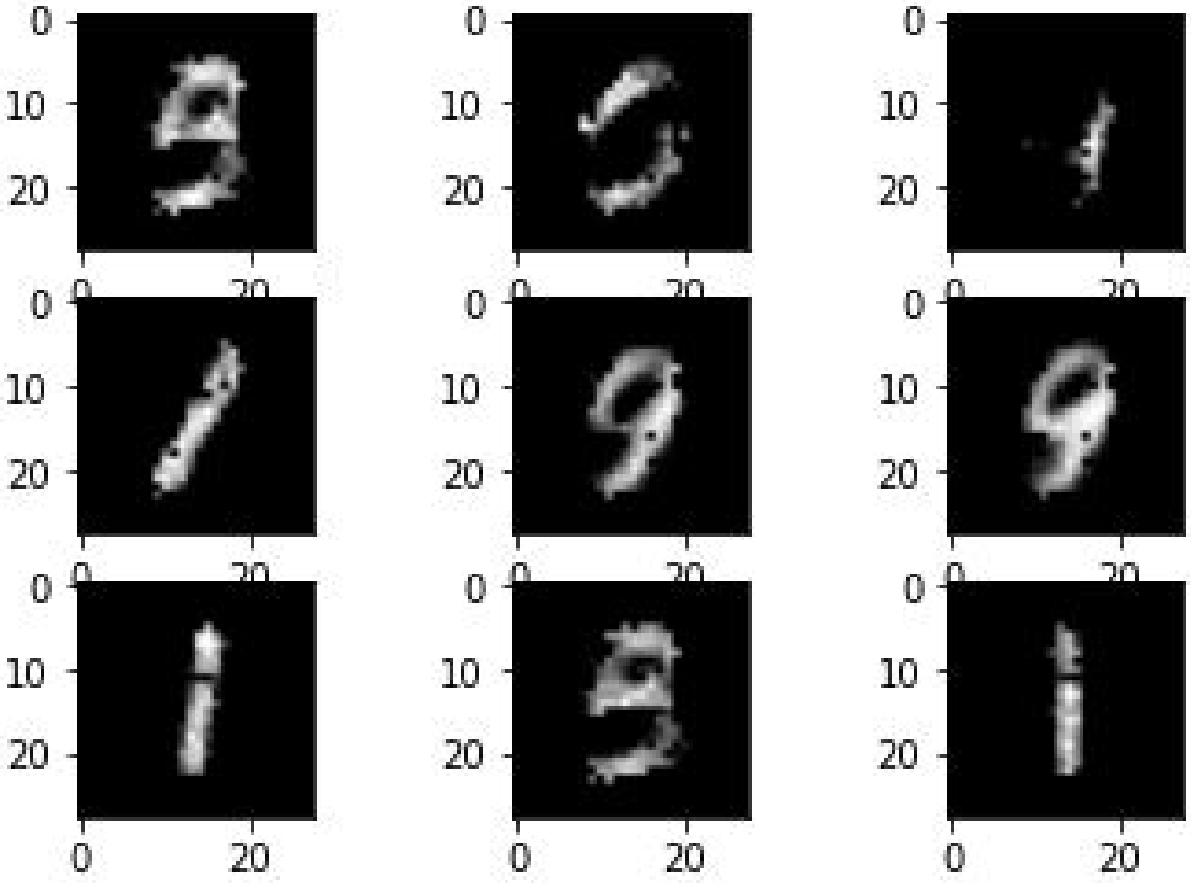
For each convolution layer, we compute the parameter size when $d = 3$, and $c = 10$

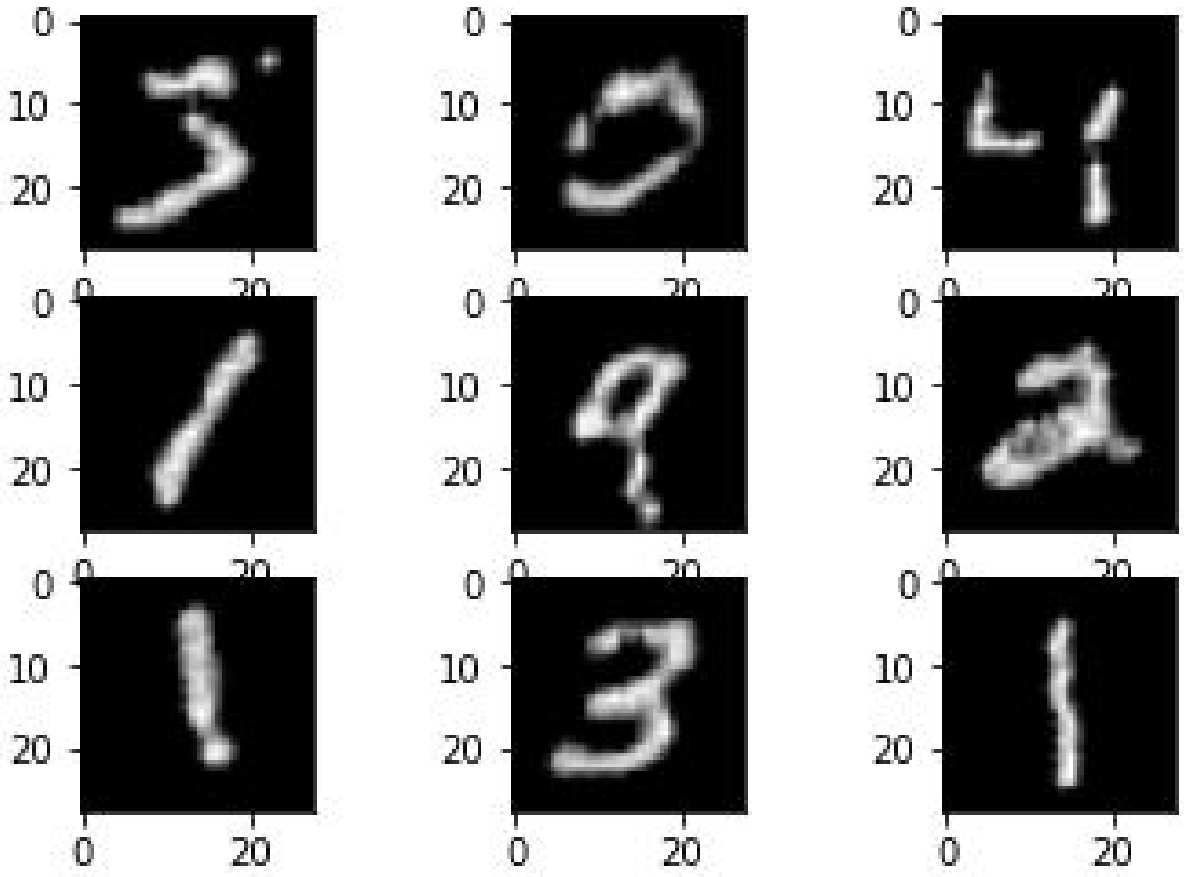
$$(3^2) \cdot 10 = 90$$

This calculation is the same for the second layer. Therefore we have $90 + 90 + 10 + 1 = 191$ total trainable parameters in this network. Our code snippet confirms this.

Here, we denoise 9 (10 doesn't fit right in output) images using the linear and CNN Model. From top to bottom we have the noisy images, the linear denoising and the CNN's denoising:




0
10
20
0
20

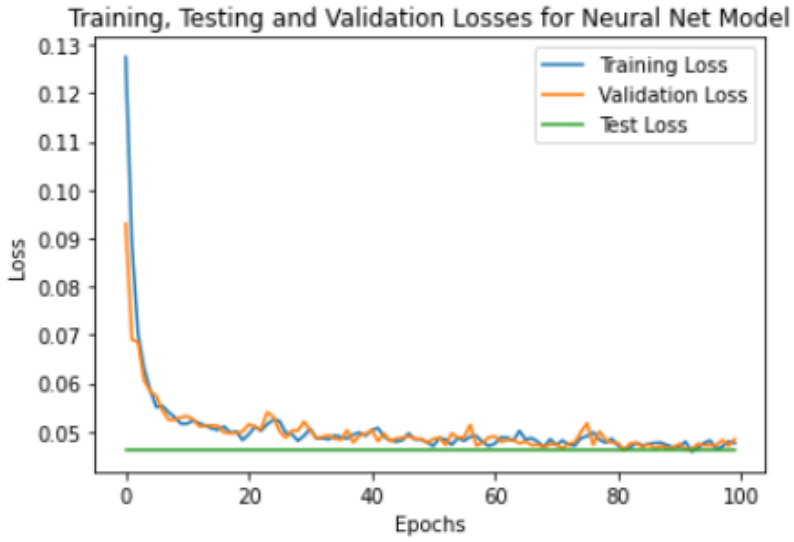


The CNN works much much better at denoising the images, the linear model seems to incorrectly denoise some digits as other digits (the 9 into a 2). The CNN network also 'fills' in the digit much better, making it look more like the digit it is supposed to be.

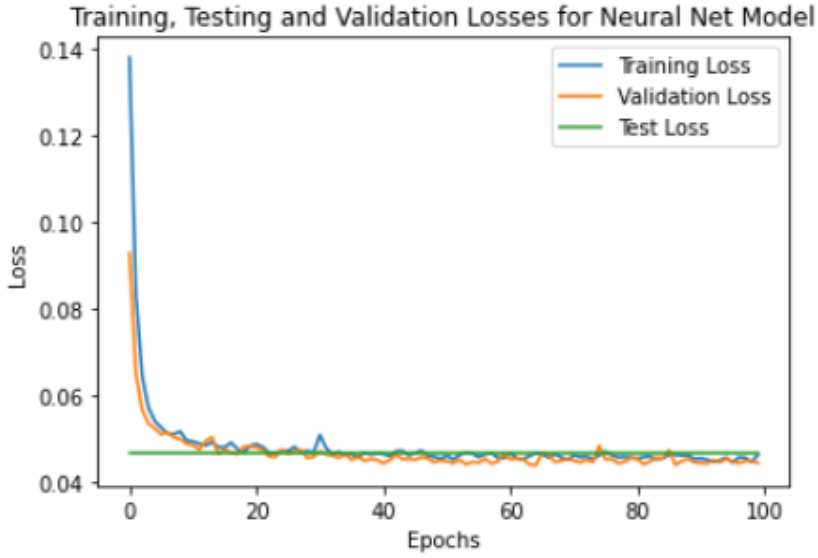
3. **Programming Component: Task Three**

We are trying to make the MyCNN model perform better by either changing the architecture or some of the parameters. The simplest thing we could do is make it run for longer epochs. So we're trying to get the loss to be even lower.

After several runs, we get an ending test loss of 0.04794158046222796 with 100 epochs instead of 50.



Halving the batch size on top of the epochs doesn't seem to do much, we get a test loss of 0.046290556212869316.



Graphically however it looks as if the loss converges much faster. We could possibly reduce epochs to save some time, and instead reduce the batch size. Overall, these small changes seem to slightly improve this model.