CSI 5340 – Deep Learning and Reinforcement Learning

Assignment 2

Student Name: Nathaniel Bowness

Student Number: 7869283

Due Date: October 19th, 2023

Question 1 – Manual Backpropagation and Gradient Descent

Manual Gradient Calculation

To write a program that computes the gradients $\frac{\delta L}{\delta A}, \frac{\delta L}{\delta B}, \frac{\delta L}{\delta C}$ for the function given, I drew out the loss-included forward graph of the equations shown. This forward graph can be seen in Figure 1 below. This was followed by computing and drawing the gradient (derivative) network and, finally, the dualized derivative network. With the dualized derivative network illustrated, I was then able to find a formula for $\frac{\delta L}{\delta A}, \frac{\delta L}{\delta B}$ and $\frac{\delta L}{\delta C}$. The formal for those 3 gradients can be seen in Figure 2 below.

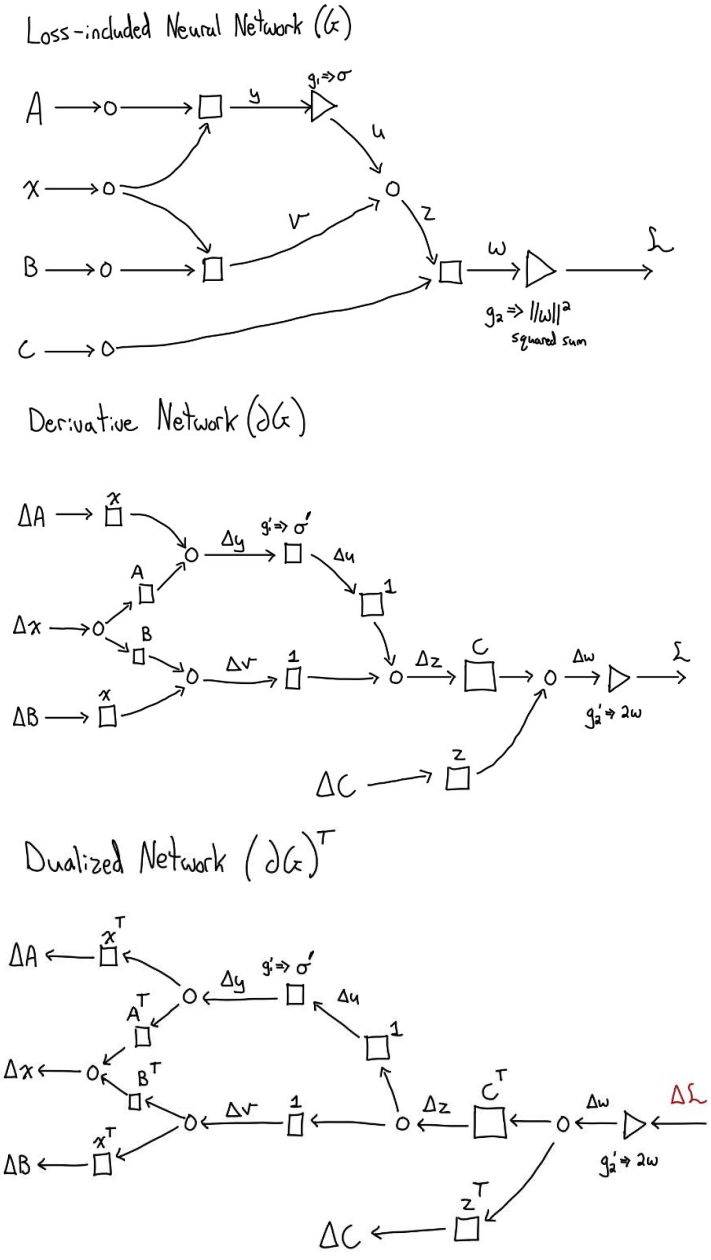

Figure 1: *Loss-included neural network G, derivative network δG , dualized network $(\delta G)^T$ of the 6 equations given, shown in order from top to bottom.*

Gradient Formulas

$$\frac{\partial \mathcal{L}}{\partial C} := \frac{\partial \mathcal{L}}{\partial w} \cdot \frac{\partial w}{\partial C} \Rightarrow \Delta C$$

$$\frac{\partial \mathcal{L}}{\partial B} := \frac{\partial \mathcal{L}}{\partial w} \cdot \frac{\partial w}{\partial z} \odot \frac{\partial z}{\partial v} \cdot \frac{\partial v}{\partial B} \Rightarrow \Delta B$$

$$\frac{\partial \mathcal{L}}{\partial A} := \frac{\partial \mathcal{L}}{\partial w} \cdot \frac{\partial w}{\partial z} \odot \frac{\partial z}{\partial u} \odot \frac{\partial u}{\partial y} \cdot \frac{\partial y}{\partial A} \Rightarrow \Delta A$$

Figure 2: *Computed equations for* $\frac{\delta L}{\delta A}$, $\frac{\delta L}{\delta B}$ *and* $\frac{\delta L}{\delta C}$ *for the 6 equations given, derived from the networks in Figure 1.*

From these equations, I was able to compute the gradients for $\frac{\delta L}{\delta A}$, $\frac{\delta L}{\delta B}$ and $\frac{\delta L}{\delta C}$. The code is included in the zip file with this assignment. I then used the torch python library to re-calculate the gradients on the same x, A, B, and C variables to ensure the computed gradients were accurate. From my testing, I achieved the same numbers as the library. In Table 2 below, I show the calculated gradients $\frac{\delta L}{\delta A}$, $\frac{\delta L}{\delta B}$, $\frac{\delta L}{\delta C}$ for K=2 dimensions and a random seed of 1 for reproducibility. They achieve the same answers for all values of K tested and with any random matrix initialization.

Table 2: *Calculated values for gradients* $\frac{\delta L}{\delta A}$, $\frac{\delta L}{\delta B}$, $\frac{\delta L}{\delta C}$ *when K=2, using the manually computed method and using the torch library.*

Gradient	Manually Computed	Torch
$\frac{\delta L}{\delta A}$	[[0.1132, -0.2149], [0.1088, -0.2066]]	[[0.1132, -0.2149], [0.1088, -0.2066]]
$\frac{\delta L}{\delta B}$	[[0.5969, -1.1332], [0.5020, -0.9530]]	[[0.5969, -1.1332], [0.5020, -0.9530]]
$\frac{\delta L}{\delta C}$	[[-0.3751, -0.2854], [0.1396, 0.1062]]	[[-0.3751, -0.2854], [0.1396, 0.1062]]

Back Propagation using Gradient Descent:

After computing and confirming gradients, they were used to minimize the loss function, $(\hat{A}, \hat{B}, \hat{C}) := \arg\min_{A,B,C} \sum_{i=1}^{N} \mathcal{L}(x_i; A, B, C)$ using gradient descent.

A simple gradient descent algorithm was used, like in Assignment 1. The gradients were calculated using the created algorithm for each input set of data N (the length of the dataset x). From there, the gradients were multiplied by a learning factor and added to the current weights, A, B, and C. After iterating N times, the loss function was minimized. Two different learning rates of 0.001 and 0.01 were used to see how quickly the loss would decrease. Both of these trials for different vector sizes of K=8 and K=20 are shown in Figure 4 below.

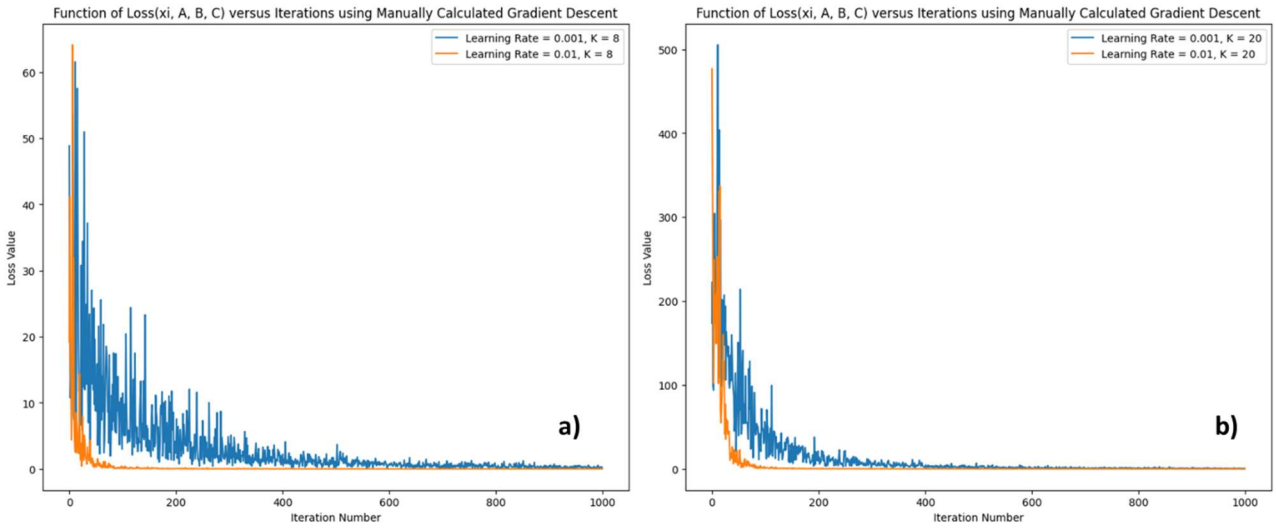

Figure 4: *The loss function graphed over iterations of gradient descent, using variable learning rates and leveraging the manually calculated gradients. Figure 4a) shows the loss for vectors of dimension K=8, which has an initial loss value of around 60. Figure 4b) shows the loss for vectors of dimension K=20, which has an initial loss value of about 500.*

The minimization of loss using the computed gradient makes it apparent that it works for implementing gradient descent.

Question 2 – MNIST Image Classification

This part of the assignment will be broken into 4 sections: details about the different created models, including SoftMax regression, MLP and CNN. Then, one final section compares the 3 models against each other with some final statements.

The following settings were kept constant across the different image classification algorithms, unless explicitly mentioned. They were kept constant to ensure consistency between the initial data comparison of soft-max, MLP and CNN image classifiers.

- Dropout Rate = 0.5
- Epochs = 8
- Batch Size = 500
- Optimizer: Adam (Improved Gradient Descent)
- Learning Rate: 0.001

The 3 image classifiers were all implemented using TensorFlow, using the "keras" library in TensorFlow to add different layers to the model if more than 1 was required. All training data was taken from the MNSIT sample dataset for image classification on different models.

SoftMax Regression Classifier

The SoftMax regression model is used in this assignment as a single-layer image classifier. Since it only contains one layer, it is expected to be the least accurate compared to the others implemented later in

this assignment. The SoftMax regression model I created uses the following additional settings to the ones mentioned:

- Loss Function: Cross Entropy

Using this to create SoftMax regression model and training it on the 60000 inputs from the training portion of the MNIST dataset, I got the following training loss and training accuracy results, shown in Figure 5, over the specified number of iterations. I found it was essential to use batches of the input data for training to speed up the process, particularly for the later image classifier models. As expected, the more iterations through the data, the more accurate the model got until it reached a plateau.

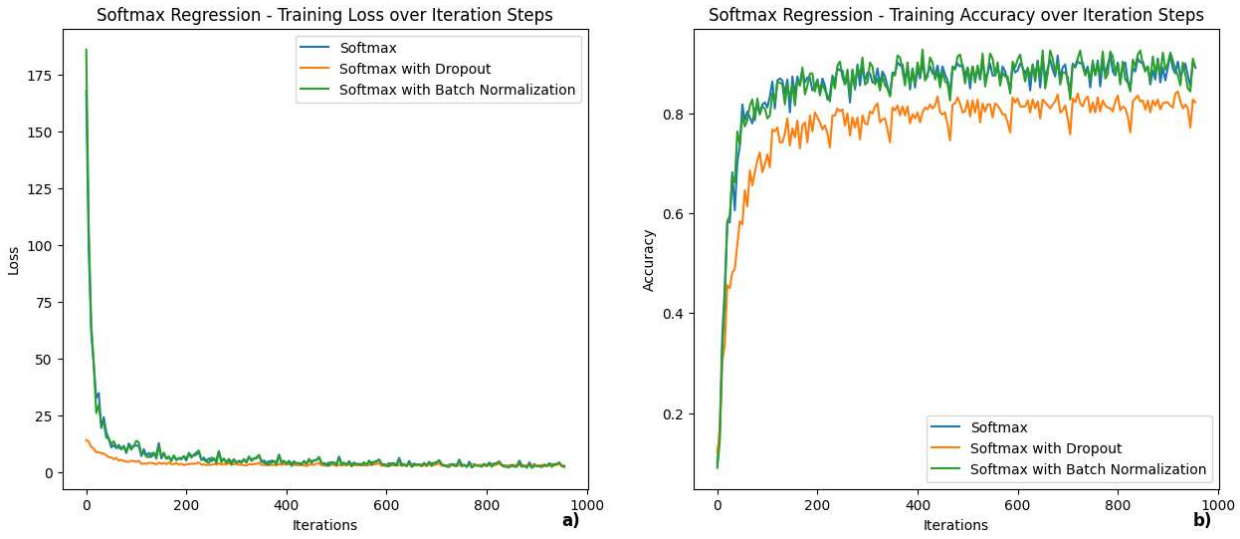

Figure 5*: SoftMax regression training loss a) and training accuracy b) over-training iteration steps of the MNIST training data.*

From Figure 5, we can see that the loss for all the algorithms was minimized over the set of iterations performed. The initial loss with batch normalization was the highest, while the initial loss of the dropout algorithm was the lowest. The training accuracy of SoftMax with and without batch normalization was approximately the same at around 0.85. However, the training accuracy of SoftMax with dropout was less, showing that dropout negatively impacted the accuracy.

In Figure 6 below, we'll see similar results as Figure 5b), which are shown on the training data from the MNIST dataset. Once again, the results of SoftMax with dropout were worse than without, with almost a 10% reduction in accuracy. The batch normalization apparently had minimal effect on the algorithm, with roughly the same results. While not explicitly shown in the graphs here, I did notice that too many iterations on the dataset did result in some overfitting for the model. It was not much overfitting, but enough to see a slight drop in the test accuracy.

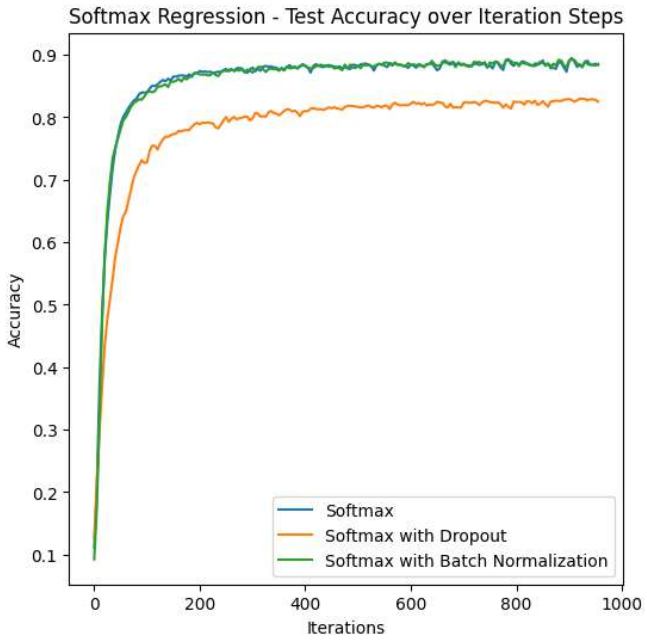

Figure 6: *SoftMax regression test accuracy over the number of training iterations on the MNIST test data. SoftMax with dropout performs worse than SoftMax regression by itself.*

Since dropout greatly affected the results and decreased accuracy, Figure 7 below shows the effect of different dropout rates on the SoftMax regression model and its impact. Figure 7 graphs the effect of dropout values 0.1, 0.5 and 0.8. The data shows that the larger the dropout rate, the less accurate the SoftMax model.

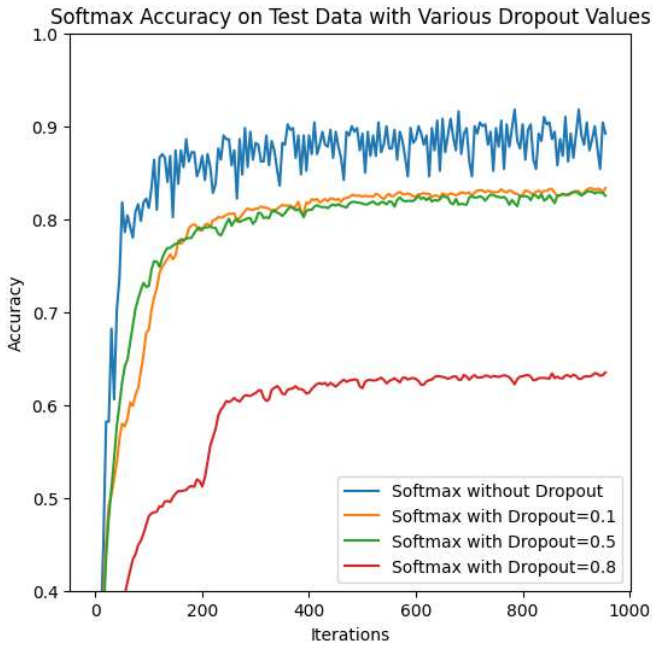

Figure 7: *SoftMax regression accuracy value with various dropout rates, including none, 0.1, 0.5 and 0.8. The higher the dropout rate of the regression model the less accurate it is.*

In conclusion, batch normalization did not impact the softmax regression classifier built for this assignment. Softmax regression without normalization performs slightly better when taking an average of all the loss values, but not by much. SoftMax regression with dropout performs worse than regular SoftMax, decreasing accuracy as the dropout rate increases. I think the main issue is dropping out nodes in the final layer of any classifier will likely reduce the model's accuracy.

MLP Classifier

Multilayer perceptron networks have one or more hidden layers with many neurons in each layer. Due to the added complexity, they tend to do better in image classification. For this experiment, the following MLP network was created with 2 hidden layers:

- **Input Layer:** 784 nodes (28*28)
- **Hidden Layer 1:** 128 nodes, Activation Function: Relu
    - Optional Batch Normalization, Optional Dropout
- **Hidden Layer 2:** 64 nodes, Activation function: Relu
    - Optional Batch Normalization, Optional Dropout
- **Output Layer**: 10 nodes (for 0-9 numbers), Activation function: Softmax

After each hidden layer, batch normalization and dropout were optionally added based on the test run. Figure 8 shows the results of loss and accuracy for the MLP model over the training iterations. From Figure 8a), the loss curves are roughly the same for each model regardless of batch normalization or dropout. However, the model's accuracy on the training data is increased both with dropout and when batch normalization is used. Both options appear to improve the model's accuracy on the training data by a similar amount.

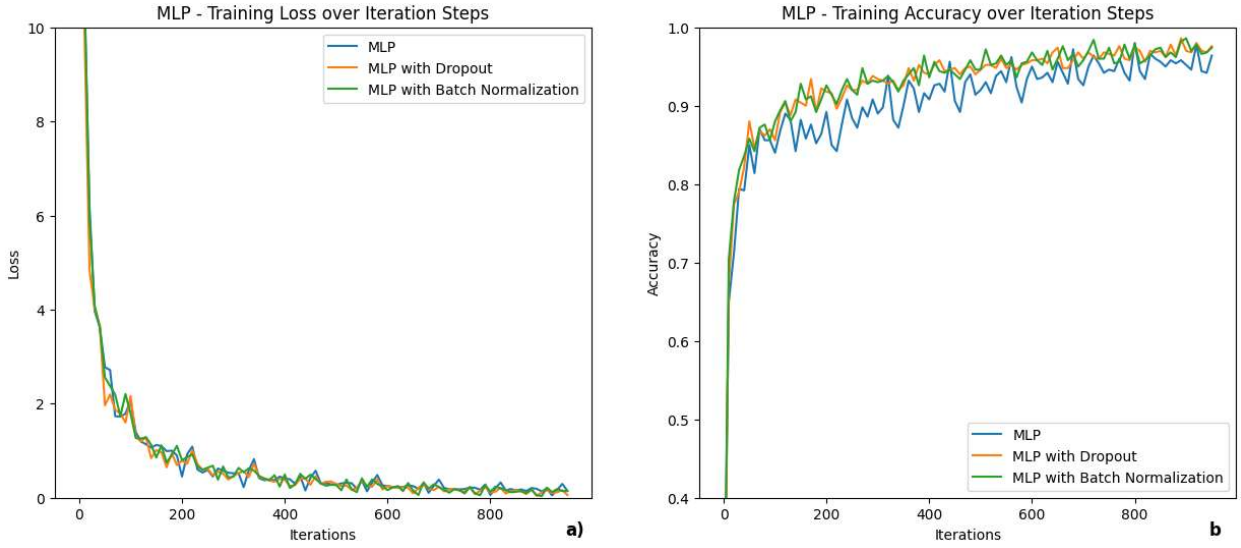

Figure 8: *MLP training loss a) and training accuracy b) over the training iteration steps of the MNIST training data.*

Figure 9 below shows the accuracy of the MLP model on the test MNIST set as the model was trained. Once again, the accuracy of MLP by itself was not as accurate. It appears that MLP with dropout performed slightly better than MLP with batch normalization, but not by much.

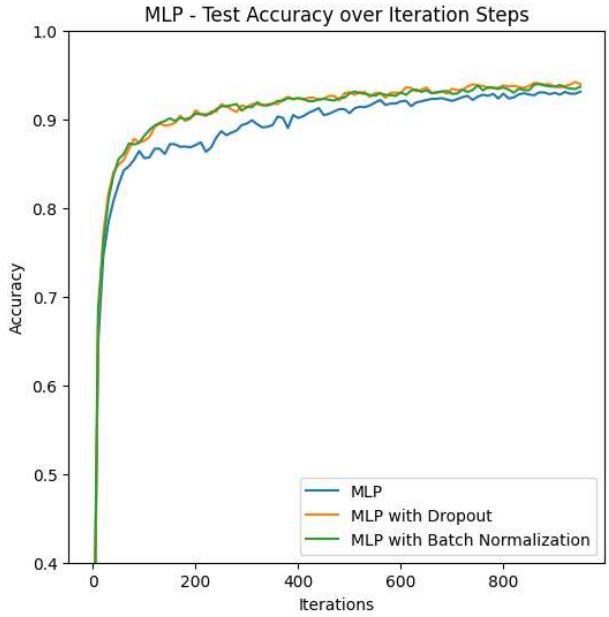

Figure 9: *MLP test accuracy over the number of training iterations on the MNIST test data*.

In conclusion, both dropout and batch normalization improved the performance of MLP by a few percentage points. Some testing was done to only add dropout and batch normalization after 1 of the hidden layers rather than both. The results were close, but the accuracy was higher after adding them to both layers.

CNN Classifier

CNN classifiers are the most complex model of the 3, with different types of layers that tend to help with image classification, so it is predicted to have the best results from theory. There are various ways to implement a CNN. For this assignment, the following CNN architecture was used:

- **Layer 1:** Spatial Convolution, Activation: Relu
  - Optional batch normalization
- **Layer 2:** Max Pooling
- **Layer 3:** Spatial Convolution, Activation: Relu
  - Optional batch normalization
- **Layer 4:** Max Pooling
- **Layer 5:** 128 Nodes, Activation: Relu
  - Optional dropout
- **Layer 6:** 10 Nodes, Activation: SoftMax

Some experiments were trialled, like with the MLP model used, where batch normalization and dropout were added after each layer when possible. I found the results were not improved when dropout was used before a max pooling layer. It decreased the model's accuracy and slowed training. Similarly, adding the batch normalization after Layer 5 appeared to have a minimal effect, so it was removed. The final model used is shown in the bullet points above.

Figure 10 below shows the loss and accuracy of the CNN model as it was trained. The results of the CNN were great. It was accurate within the first 200 iterations and approached 100% accuracy towards the end of the training. It is hard to see much difference on the graphs, but adding either dropout or batch normalization did improve the results, lower loss, and improve the training data accuracy.

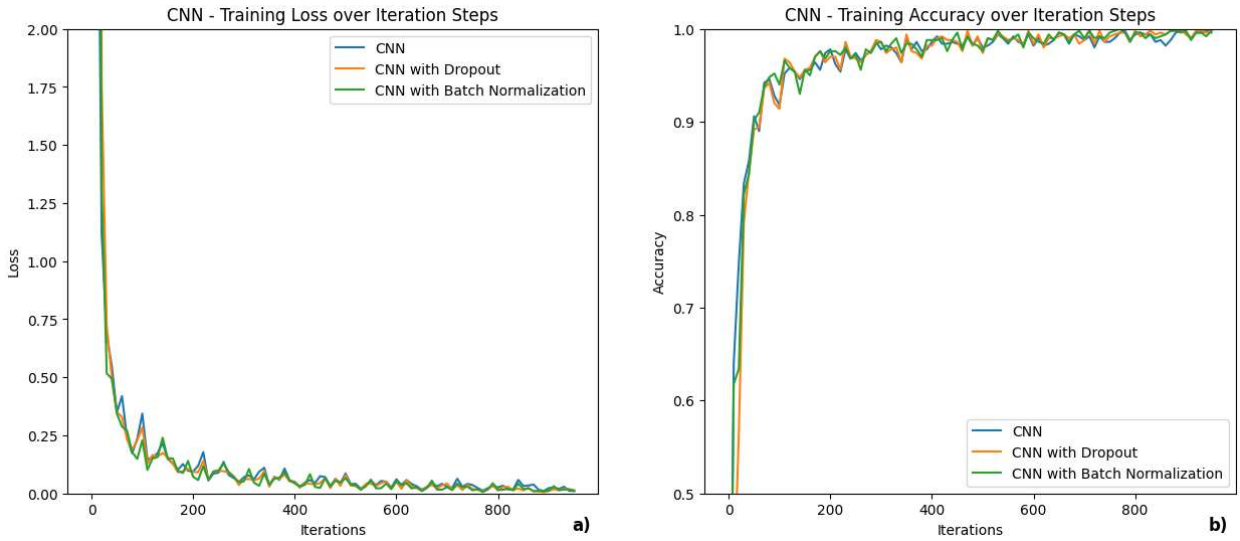

Figure 10: *CNN training loss a), and training accuracy b) over iteration steps of the MNIST training data.*

Figure 11 below shows the CNN model on the MNIST test data over the training iterations. It's easier to visualize on this figure that both dropout and batch normalization improved the algorithm. The model appears to be the most accurate when only using dropout and least accurate without either. It reaches an accuracy of approximately 98%.

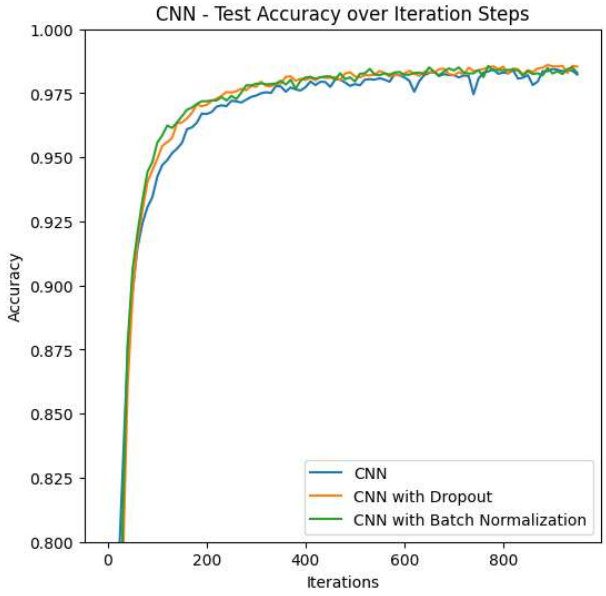

Figure 11: *CNN test accuracy over the number of trainings iterations on the MNIST test data.*

In conclusion, the CNN is a very accurate model for this simple image classification task. However, It is the most complex model, and it takes the longest to train and run an accuracy test on. So, the complexity does have a cost to run. During testing, I found that more layers in the CNN quickly performed better on the test data, but once again, the increased complexity increased the training time.

Classifier Comparison and Thoughts:

Figure 12 compares the different image classifier's loss and accuracy as trained. The figure shows different axis scales for the loss and accuracy to give the reader a "closer" look at the trends toward 100% accuracy and 0 loss. CNN classifiers perform the best on this image classification task on the MNIST dataset. CNNs take the least number of iterations to train and are the most accurate quickly. It's apparent from the data that after around 200 iterations, the CNN does not become much more accurate.

The second most accurate model, with the least amount of loss, is the MLP algorithm. The least accurate model is the SoftMax regression model. As mentioned earlier, the more complex models perform much better for image classification as they are more oriented to it.

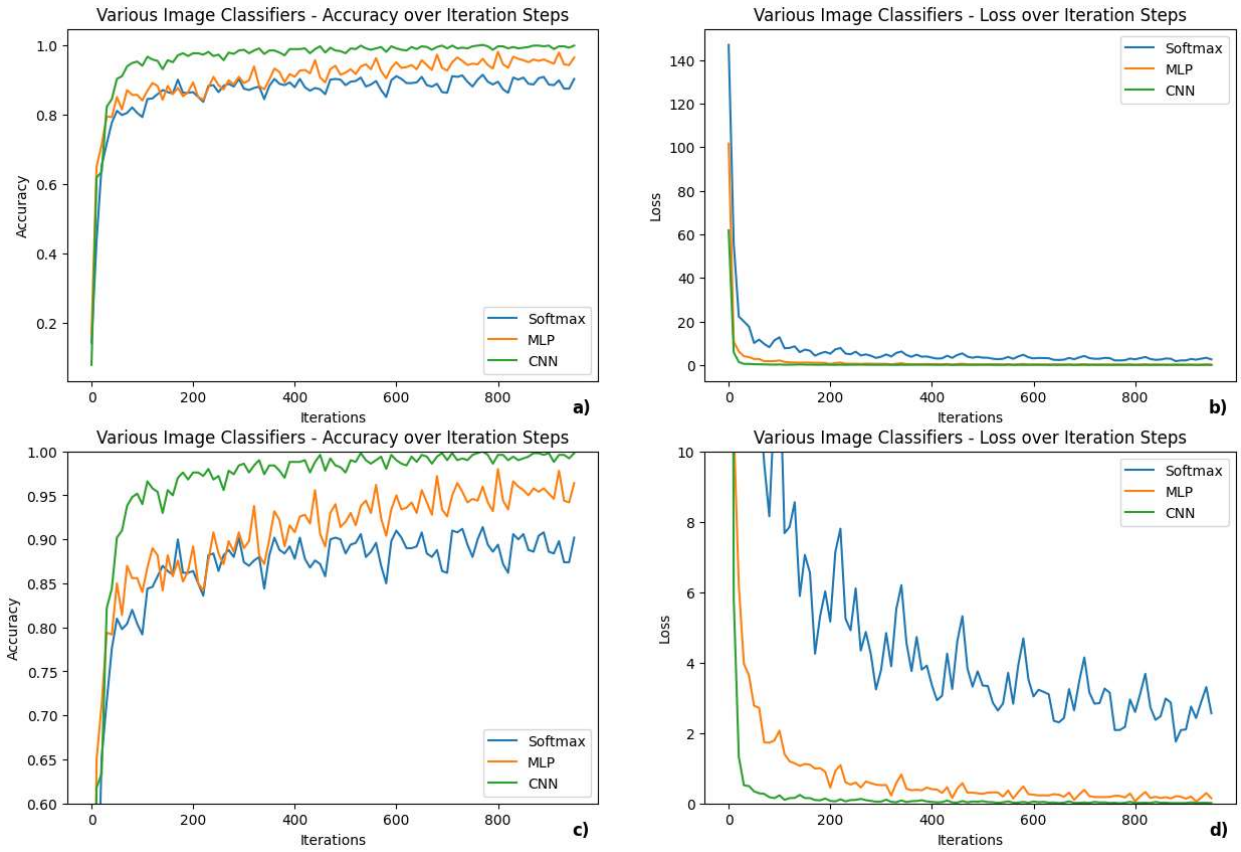

Figure 12: *Accuracy and loss of various image classifiers on the training data. A), c) show the accuracy of the image classifiers on different scales. All classifiers were quite accurate, so c) shows the top end of the accuracy results. B), d) show the loss of the image classifiers for the same data on different y scales.*

I found it quite interesting to see the difference in the starting loss values for the image classifier algorithms. The starting loss for the CNN network is around 60, whereas the starting loss for the MLP is

140. I had expected CNN to initially have a higher loss because of the complexity. Still, the multilayer design, including max pooling between some layers, appears to help reduce loss even at the start.

Figure 13 below shows the accuracy of the different classifiers on the test MNIST data over the iterations as the model is trained. Based on the results, the CNN generalized the best for image classification, with only a drop of under 2% between the training and test data accuracy. Both other algorithms see a drop of about 5% when comparing the accuracy of the training versus test data.

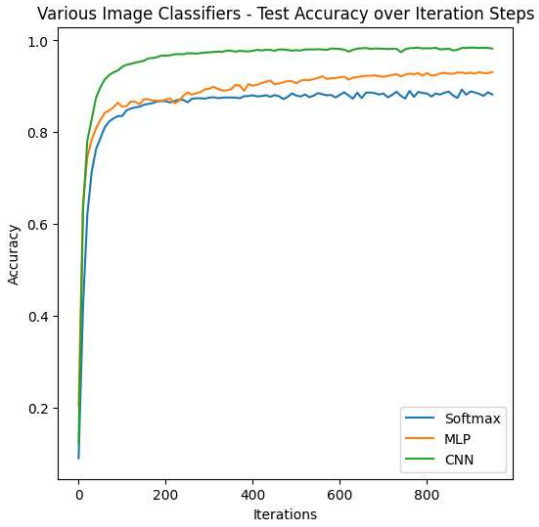

Figure 13*: Test accuracy of various image classifiers over the iterations as the model was trained.*

One lesson learned from the experiment is that batching the data when training is vital; otherwise, the models take an extremely long time to train. Also, I found that computing the accuracy of the training data and test data each iteration is expensive and takes a long time to compute, especially on more complex networks like CNN. I eventually only started to compute the accuracy every 20 training intervals to speed up the training, while still being able to graph the results. I can see why loss is normally the function graphed because it does not take additional steps to compute while training the model. After training the 3 classifier models, the final observation is that for more complex models like the CNN, each training interval takes longer compared to the simple SoftMax model. However, the CNN becomes accurate quickly and does not require as many intervals. So, moving forward, when using more complex classification models, I will know that fewer training iterations may be required for complex models and that it is important to look at the loss when training to find the ideal point to stop.