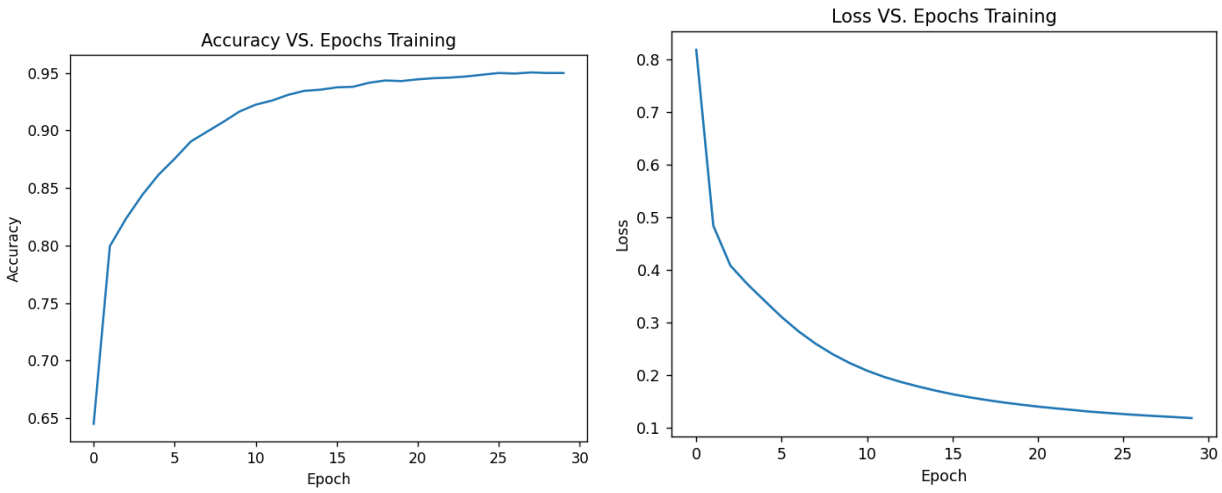
Homework 3 Writeup
CS 349 - Machine Learning
Spencer Rothfleisch, Louie Shapiro, Max Ward

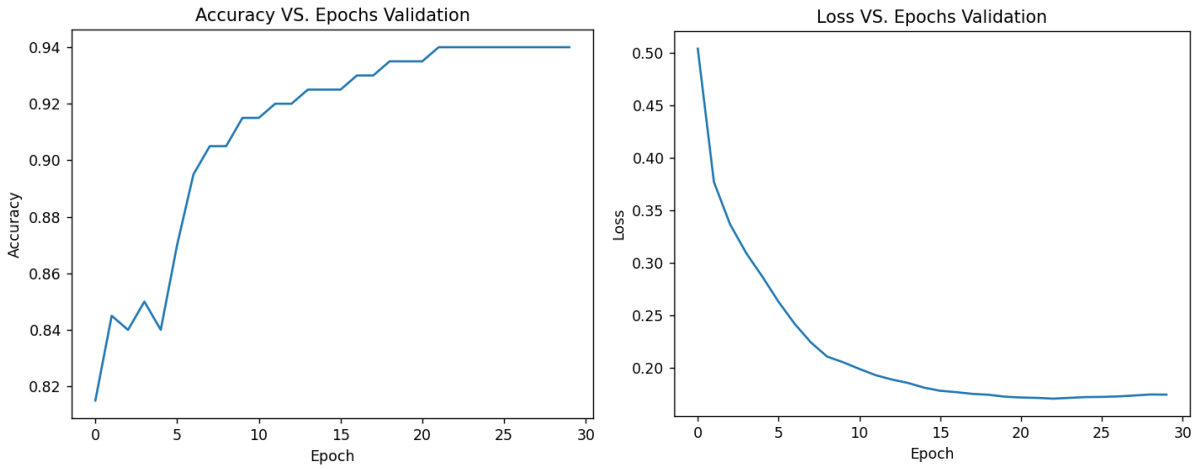
1. (8.0 points) Implement a simple feed-forward neural network in PyTorch to make classifications on the “insurability” dataset. The architecture should be the same as the one covered in class (see Slide 12-8). You can implement the neural network as a class or by using in-line code. You should use the SGD optimizer(), and you must implement the softmax() function yourself. You may apply any transformations to the data that you deem important. Train your network one observation at a time. Experiment with three different hyper-parameter settings of your choice (e.g. bias/no bias terms, learning rate, learning rate decay schedule, stopping criteria, temperature, etc.). In addition to your code, please provide

(i) learning curves for the training and validation datasets

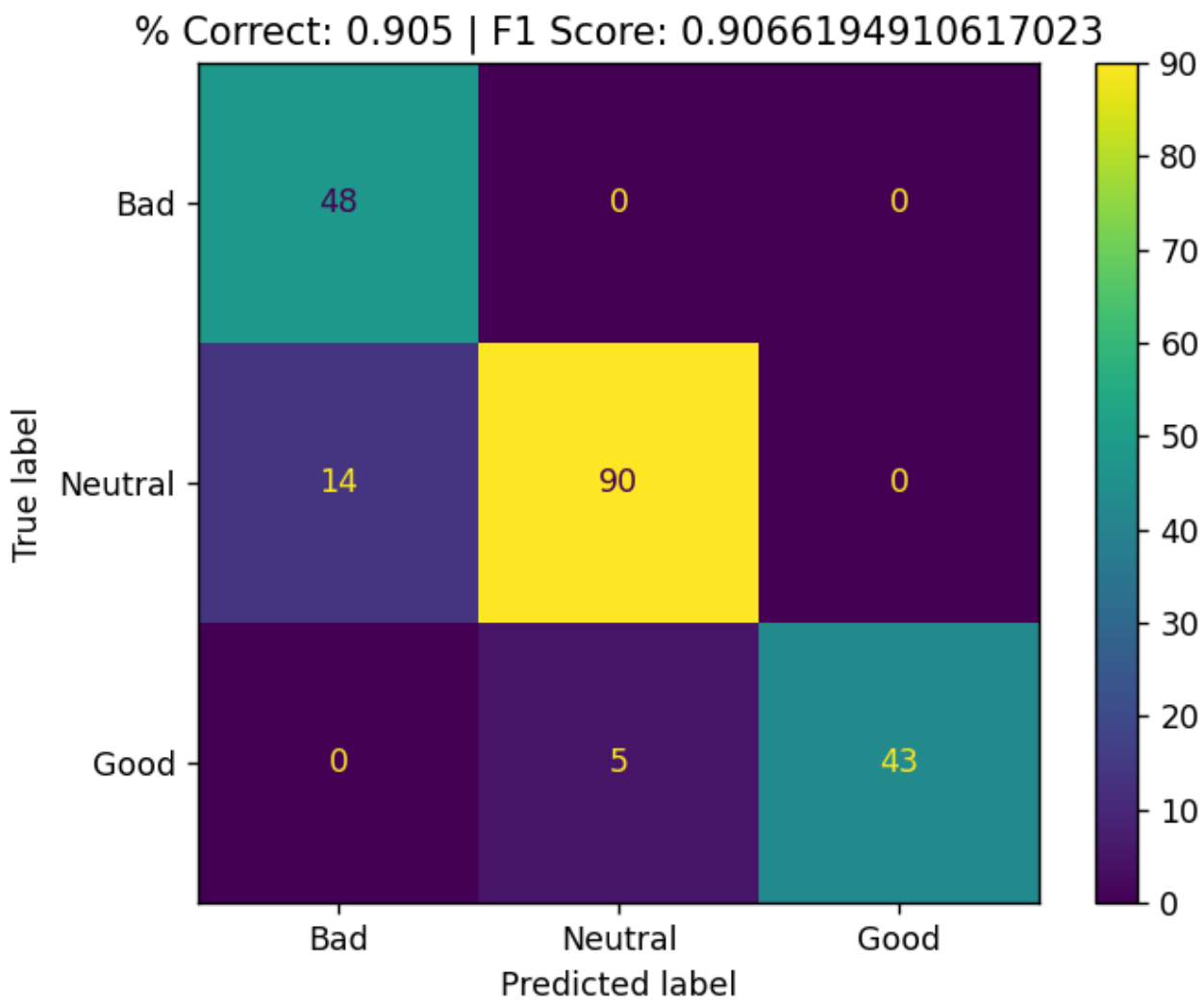
Training data learning curves:



Validation data learning curves:



(ii) final test results as a confusion matrix and F1 score **for test data set:**



(iii) a description of why using a neural network on this dataset is a bad Idea

For starters, neural networks work better on large data sets that have many different variables. The dataset being used for this case simply has three variables (Age of person, hours of exercise, and amount of cigarettes smoked) to determine 3 possible outcomes (Bad, Neutral, or Good health), indicating it may not be best for a neural network. Additionally, using a neural network to determine the health of a person based on 3 factors is incredibly problematic. Health is based on way more variables than just how old you are, how much you exercise, and how much you smoke. Genetic variation can lead to predisposed health issues, illnesses and viruses can lead to health issues, and just general variability of the human population can explain why some people smoke every day but are in good health while others only smoke a little bit are in bad health. None of this is accounted for in the data set, especially since there are only 200 entries for each grouping (training, test, validation) it is impossible to include every possibility with only three features. Finally, there are ethical concerns in using AI within the medical world, specifically with health predictions, as there are serious consequences in the medical world for incorrect predictions. It is very difficult to ensure accuracy with neural networks in this context.

(iv) short discussion of the hyper-parameters that you selected and their impact.

The first design choice that we would like to mention is a dataset transformation. We converted good, neutral, and bad to the values 0, 1, and 2 respectively. We made this change because the model requires operating on numerical data for training the network and examining the loss. Secondly, as for the hyper-parameters, we experimented with three different hyperparameters settings. The first was the learning rate. We settled on a learning rate of 0.02. A learning rate of 0.02 means that, with each update, the weights of the model are adjusted by a step proportional to 2% of the gradient's magnitude. When we decreased or increased the learning rate we noticed that the accuracy decreased and that the loss function on the training data wasn’t always decreasing, and often jumped up and down drastically. The second hyperparameter we tested different values with was the stopping criteria. We set the stopping criteria to be after a set number of iterations—labeled in the graphs as epochs—of training and evaluating the neural network. We settled on a value of 30 iterations. As seen in the training and validation graphs above, the accuracy and loss begins to plateau in the range of 25-28 epochs, so we chose 30 to fully illustrate the plateau. Additionally, when we tested other forms of stopping criteria, such as by examining delta increase over each iteration, we found that the criteria would stop the iterations too early as there would be an occasional iteration that did not increase as much as expected. The final hyperparameter we selected was our choice of the number of neurons within the hidden layer. We chose to go from 3 input neurons to 8 neurons with a singular hidden layer. We chose 8 because in our testing this number of neurons appeared to be the perfect combination of having high accuracy whilst avoiding overfitting the data through our brute force testing method of trying various values for the hidden layer.

2. (8.0 points) Implement neural network in PyTorch with an architecture of your choosing (a deep feed-forward neural network is fine) to perform 10-class classification on the MNIST dataset. Apply the same data transformations that you used for Homework #2. You are encouraged to use a different optimizer and the cross-entropy loss function that comes with PyTorch.

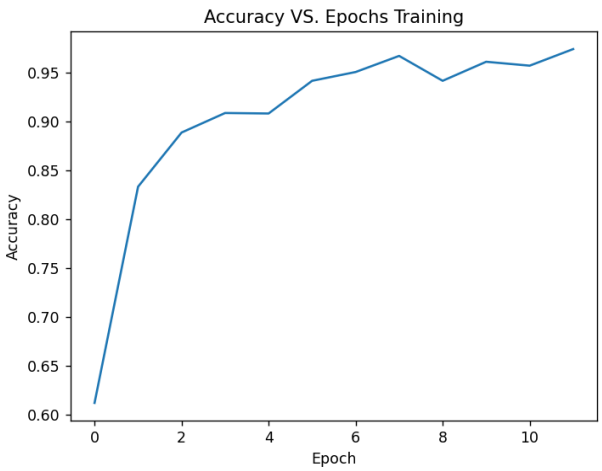
- Describe your design choices, provide a short rationale for each and explain how this network differs from the one you implemented for Part 1.
- Compare your results to Homework #2 and analyze why your results may be different.
- In addition, please provide the learning curves and confusion matrix as described in Part 1.

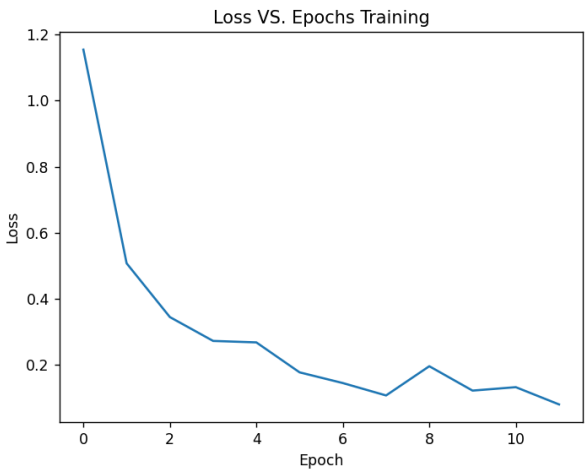
For the neural network we implemented in this assignment for the MNIST dataset, we did not have to scale the output values for the loss function, in the same way that we did for the insurability dataset. This is because the desired classification outputs are between 0-9 and do not include negative values.

The hyperparameters we decided to tweak were the number of hidden layers (1, 2, or 3), the learning rate, the stopping criteria, and the number of neurons in the hidden layers. For the number of hidden layers, we found that 2 hidden layers was ideal, as 1 hidden layer tended to cause the model to overfit, and 3 hidden layers didn't offer a notable improvement, while causing the overall runtime to increase. Increasing the number of hidden layers with this dataset seemed to improve performance because it is a significantly more complicated dataset. When adjusting the learning rate, we saw similar impacts on performance as what we observed in part 1 (accuracy decreased and the loss function on the training data wasn't always decreasing, instead jumping up and down), again deciding to use a learning rate of 0.02. For stopping criteria, we tested hard coded numbers of iterations (epochs), and noticing that the accuracy began to plateau around 10-15 epochs, settling on 12. For the number of neurons in the hidden layers we decided on 24 per layer, which was determined empirically based on improvement with overfitting and loss.

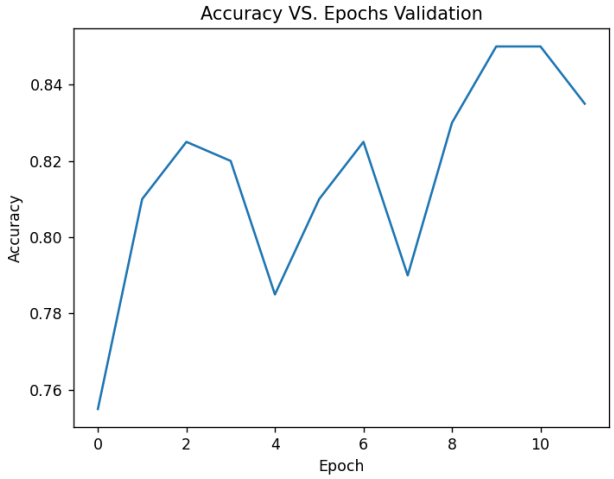
Compared to homework 2, our results here were significantly better. Our accuracy for k-nearest was ~80% and for k-means was ~56%, whereas here, we were able to achieve accuracies around 87%. This is because a neural network is better suited for a complicated dataset like MNIST. This could be due to a number of reasons, but the most apparent is likely the curse of dimensionality that occurs with KNN, as the MNIST dataset is a 784-dimensional space. Additionally, neural networks are generally better suited for dealing with non-linearities in the data, for use cases like image recognition.

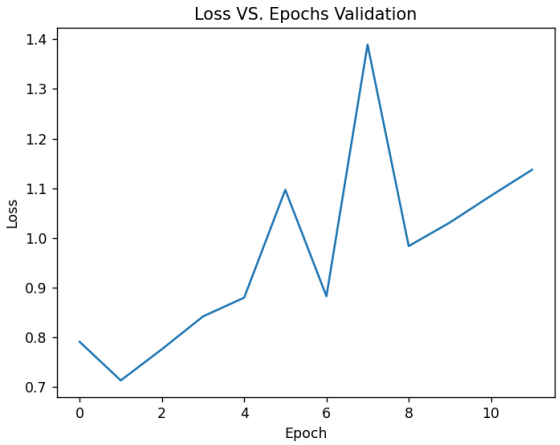
(i) learning curves for the training and validation datasets
Training data learning curves:





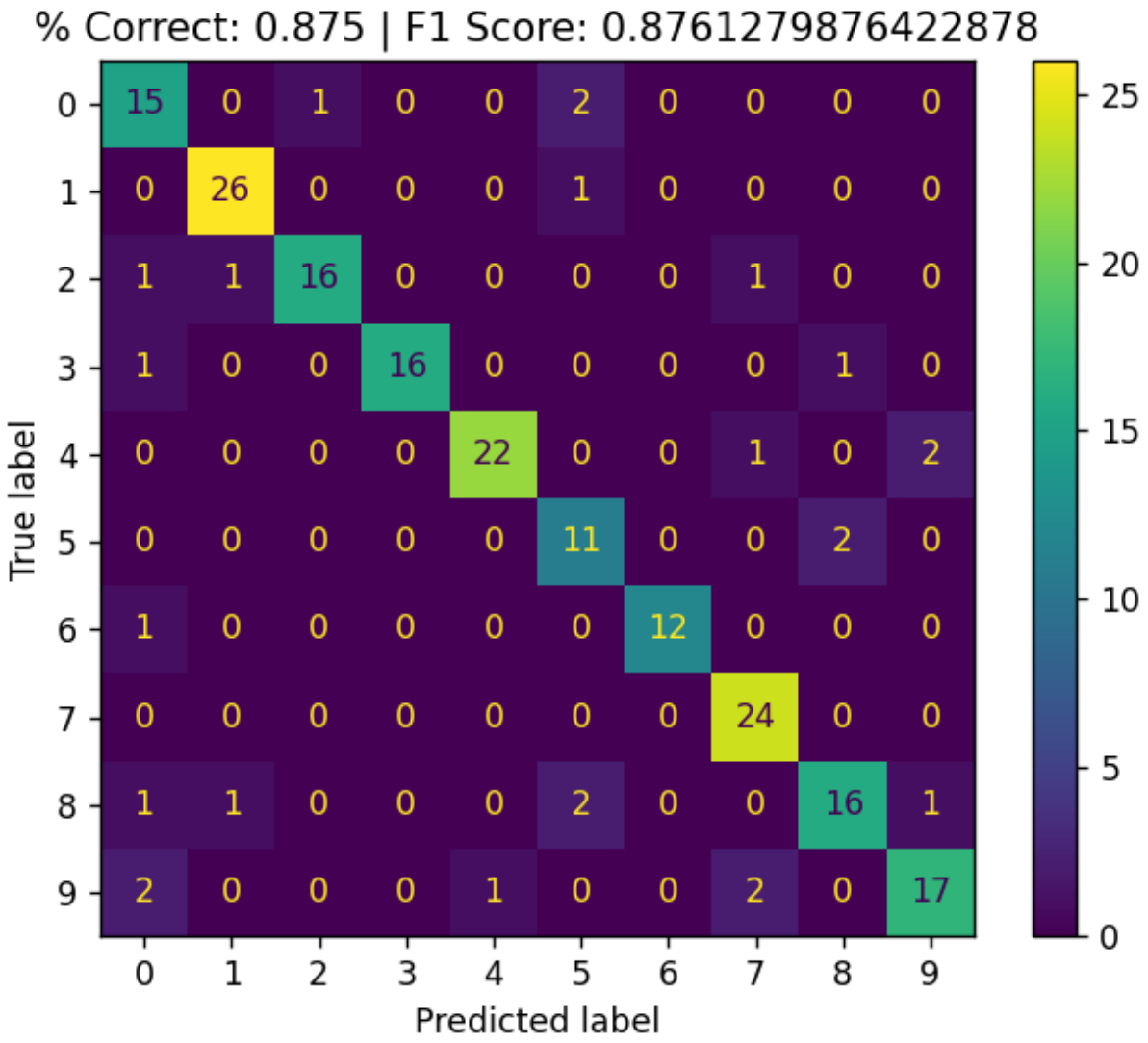
Validation data learning curves:





We would like to note that we are aware of the issues regarding the loss graph increasing; however, when tweaking hyperparameters we realized that the test data set actually has a higher accuracy with the current settings, even though the validation loss graph appears to indicate an issue with our process. Additionally, using a regulator in question 3 helped with the oddities with our validation loss.

(ii) final test results as a confusion matrix and F1 score **for test data set:**



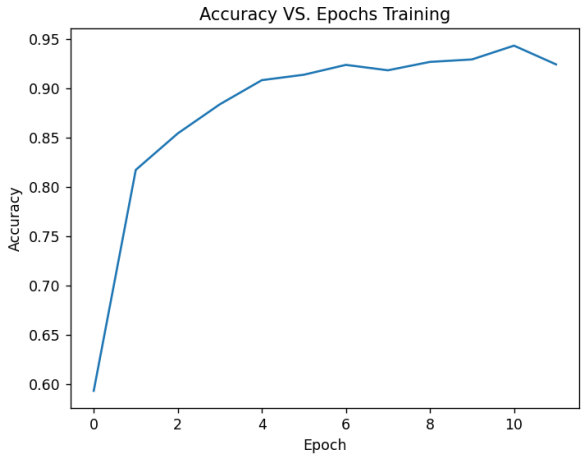
3. (i) Describe your regularization, the motivation for your choice, and analyze the impact on the performance of the classifier.

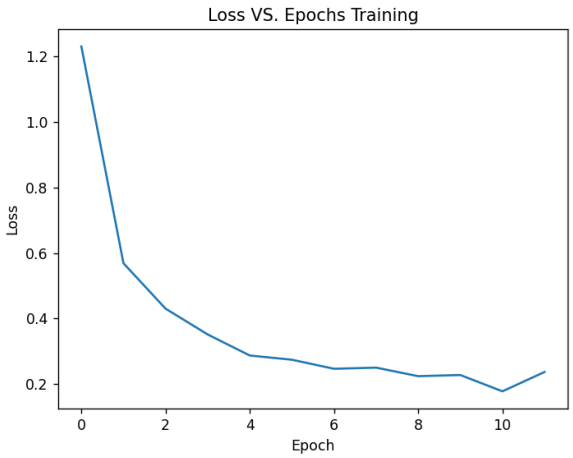
We chose to implement L2 regularization, as it was provided by PyTorch in the parameters of the optimizer, and it seemed to perform adequately. L2 regularization will remove small percentages of weights at each iteration, which will improve performance with regard to overfitting. While the overall performance of the classifier actually decreased slightly with regularization, the loss performed better, tending to decrease across iterations (loss tended to increase without regularization). It is important to note that the loss would jump up and down throughout, with and without regularization. For hyperparameters, we changed the number of nodes per hidden layer and also tweaked the weight decay. The rest of the hyperparameters did

not appear to greatly affect the performance of the classifier. We found that 16 nodes per layer tended to perform better with regularization, whereas 26 nodes per layer was more effective without regularization, although we're not necessarily sure why this is the case. For weight decay, we found that 0.005 was ideal, as it tended to balance accuracy and loss quite well.

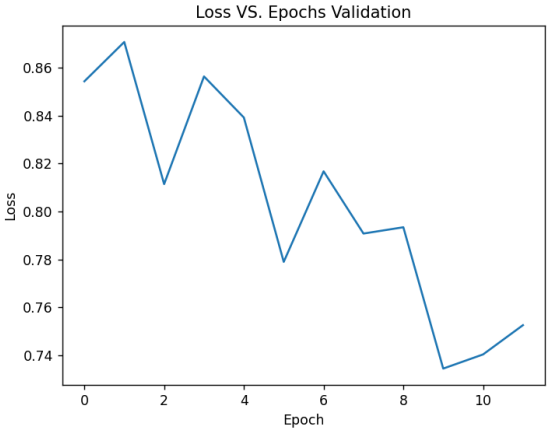
(ii) Provide graphs

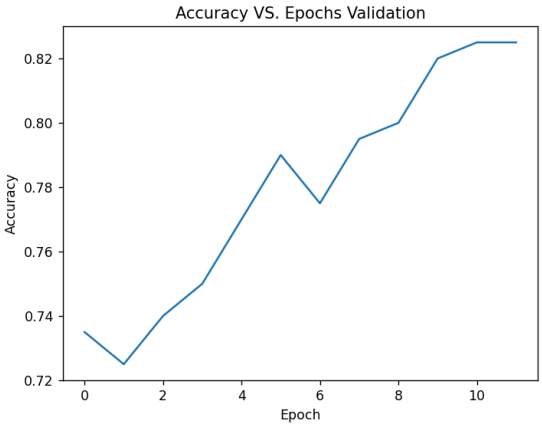
Training data learning curves:



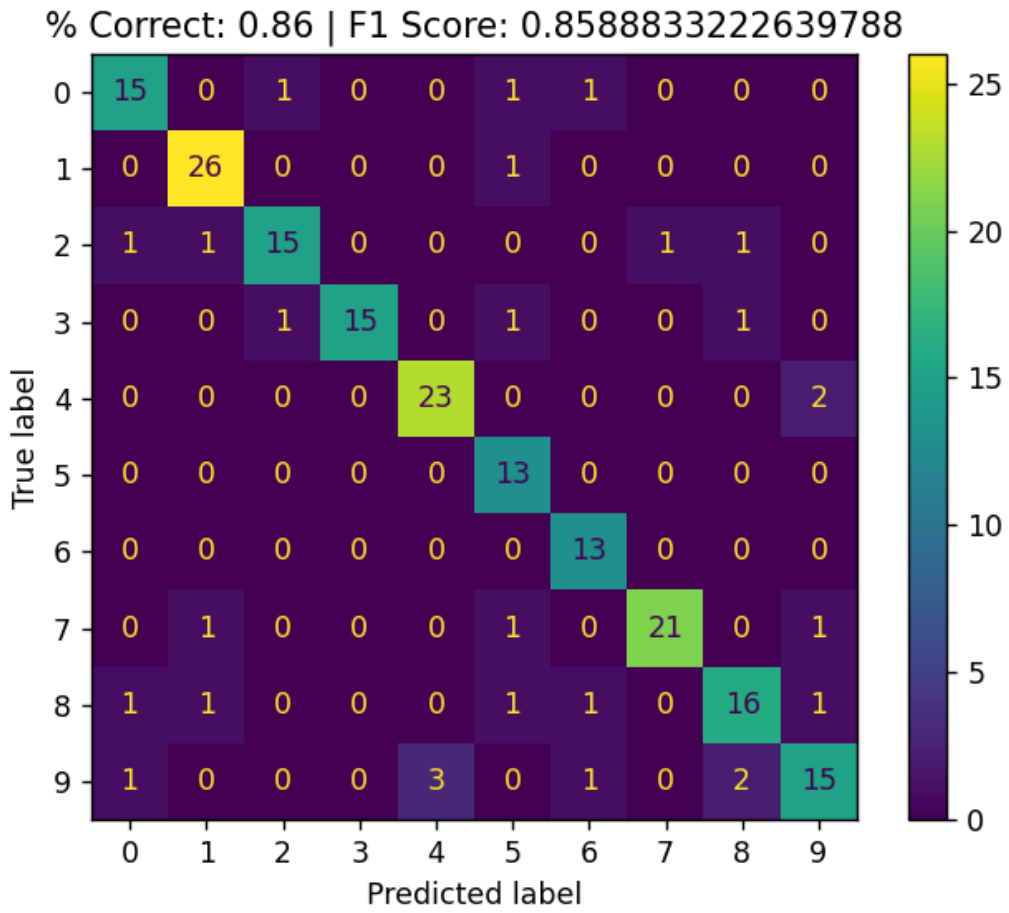


Validation data learning curves:





Final test results as a confusion matrix and F1 score **for test data set:**


% Correct: 0.86 | F1 Score: 0.8588833222639788
True label
Predicted label
0 15 0 1 0 0 1 1 0 0 0
1 0 26 0 0 0 1 0 0 0 0
2 1 1 15 0 0 0 0 1 1 0
3 0 0 1 15 0 1 0 0 1 0
4 0 0 0 0 23 0 0 0 0 2
5 0 0 0 0 0 13 0 0 0 0
6 0 0 0 0 0 0 13 0 0 0
7 0 1 0 0 0 1 0 21 0 1
8 1 1 0 0 0 1 1 0 16 1
9 1 0 0 0 3 0 1 0 2 15
0 1 2 3 4 5 6 7 8 9
25 20 15 10 5 0