Homework 4
601.482/682 Deep Learning
Fall 2023

Due 11:59pm on Oct 11, 2023

Please submit the following to Gradescope (Entry Code: BBVDNN):

- "Homework 4 - report" – Your written LaTeX generated report (.pdf)
- "Homework 4 - notebook" –
  (1) Your notebook file (.ipynb)
  (2) Exported PDF version of notebook file (.pdf)
  (3) function file (hw4_utils.py)

Note 1) It is important to be cognizant of model training time for this homework. The MLP models you design in Problem 1 likely won't take longer than a few minutes to train at maximum for each run. However, in Problem 2, you will implement two convolutional neural networks *and* experiment with hyperparameters/architectures. For context, our proposed designs require 40-60 seconds per epoch, and we ask you to train your models for at least 50 epochs. We recommend you design and test your model at a small scale (e.g. 5-10 epochs) before scaling up for the final run.

Note 2) You will be using PyTorch for the majority of this assignment. Using packages other than those specified in the notebook is not accepted for this homework.

Note 3) The function file is generated at the end of the jupyter notebook. Important functions and class will be exported and we will run the autograder on this file. This is intended to help you identify the buggy segment of your code for easier debugging. Please check the notebook file for more information.

Questions

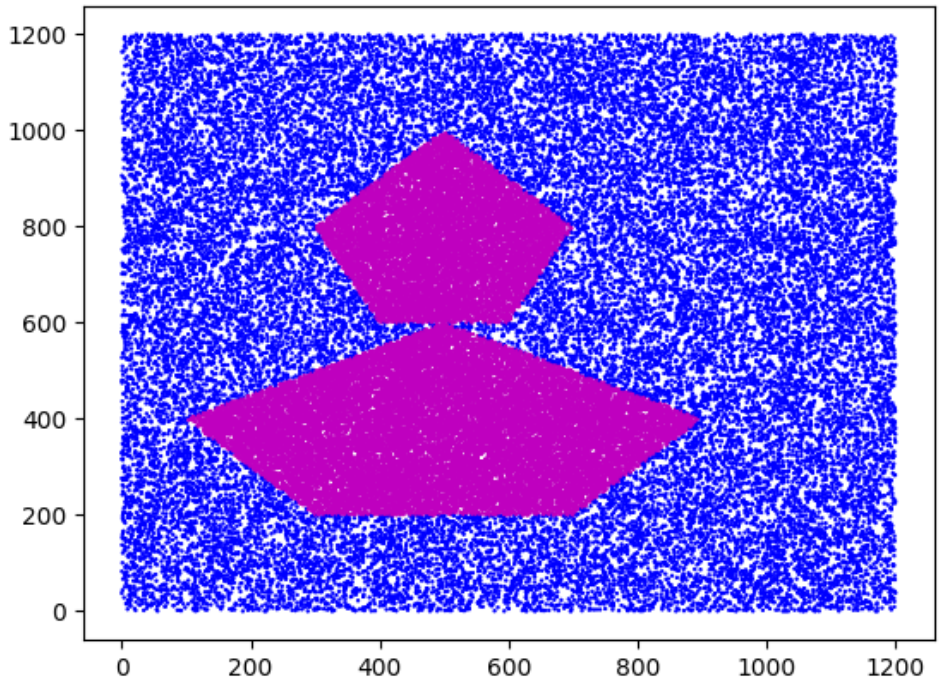
1. We learned that a two-layer MLP can model polygonal decision boundaries. You are given data sampled from a "two-pentagons" decision boundary. Data points within the pentagons are considered "true" (labeled as 1) while data points outside are considered "false" (labeled as 0).

   Please load the data via `numpy.load()` function and check that you obtain a numpy array with a shape of $(60000, 3)$. The first two columns are coordinates (x, y) and the third column represents the labels.

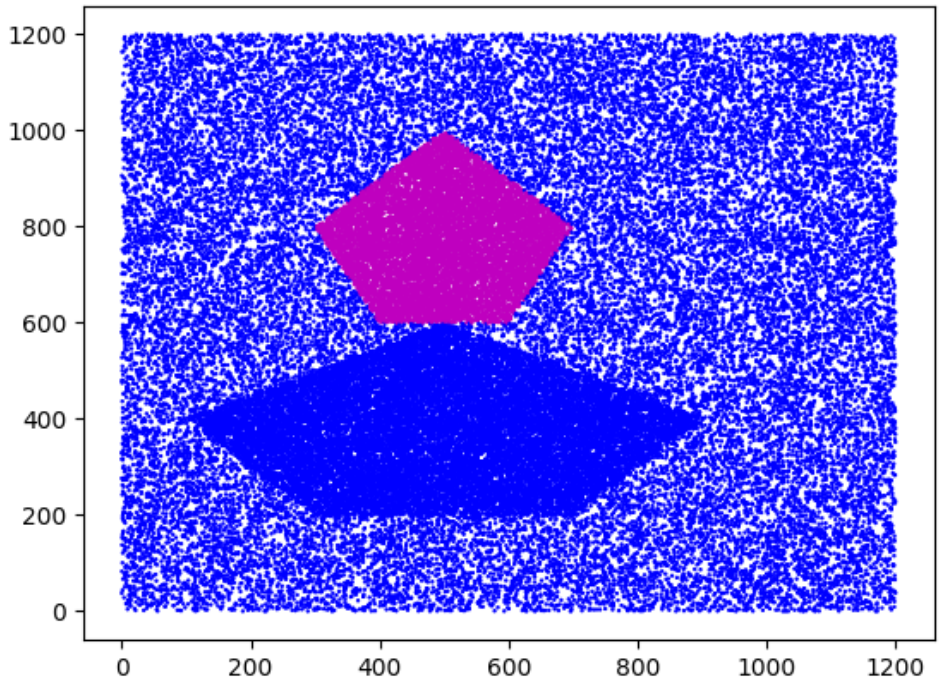
   The vertices of the 2 polygons are (500, 1000), (300, 800), (400, 600), (600, 600), (700, 800), and (500, 600), (100, 400), (300, 200), (700, 200), (900, 400). We have provided some visualization functions in the Jupyter notebook. The decision boundary can be modeled with a total of $2 \times 5 + 2 + 1$ neurons with threshold activation functions.

   (a) We have set up an MLP with threshold activation by analytically obtaining weights that can model the above decision boundary.

      i. Implement the "AND gate" and "OR gate" and the unit step function to predict all points within the 2 polygons as the positive class. Use the `predict_output_v1()` to test your implementations. Attach the visualization to your LaTeX PDF report.



ii. Modify the code in `predict_output_v2()` to predict only the points in the first polygon as the positive class. Attach the visualization to your report. *Note: You should only need to update the application of the gates themselves.*



Note: We reverse-engineered the weights from the known decision boundaries but in the real world, we do not know the decision boundaries. In a more realistic scenario, we would thus need to discover (learn) the decision boundaries from training data which we will explore next.

(b) Build an MLP using sigmoid activation functions to replace the "AND/OR" gates. Implement the function "preprocess_data". Implement the "training" function with mini-batch support. Implement visualization functions for your results. You should be

using PyTorch for this question and all remaining parts of the assignment.[1]

Data preprocessing:

i. Please use the first 50000 points as your training set and the remaining 10000 as your test set. (test_ratio is set to $\frac{1}{6}$)

ii. You may also want to perform other reasonable data preprocessing methods based on your observation of the given data prior to training your model.

Model hyperparameters:

i. Use 2 hidden layers, and 1 output layer with binary-cross-entropy loss [2]

ii. Use 10 nodes in the first hidden layer, 2 nodes in the second hidden layer, and 1 output node (binary classification), which mirrors the capacity of the model in 1a)

iii. Initialize the weights using Xavier initialization with uniform distribution. Initialize the bias with a uniform distribution.

iv. You may use any combination of batch size, training epochs, and learning rate. (Note that a larger batch size typically corresponds to a larger learning rate)

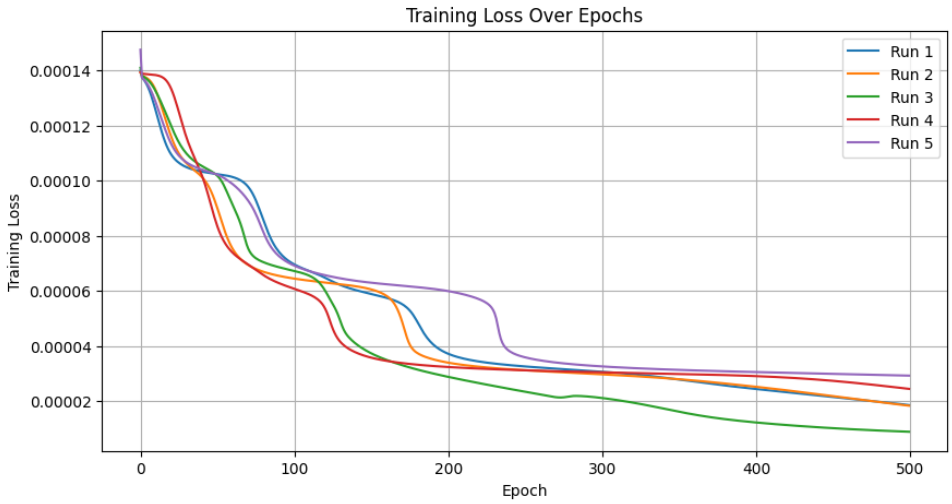
v. Do not use any regularization for this problem

Use gradient descent to optimize the parameters. You should expect an average accuracy greater than 90%.

TODO:

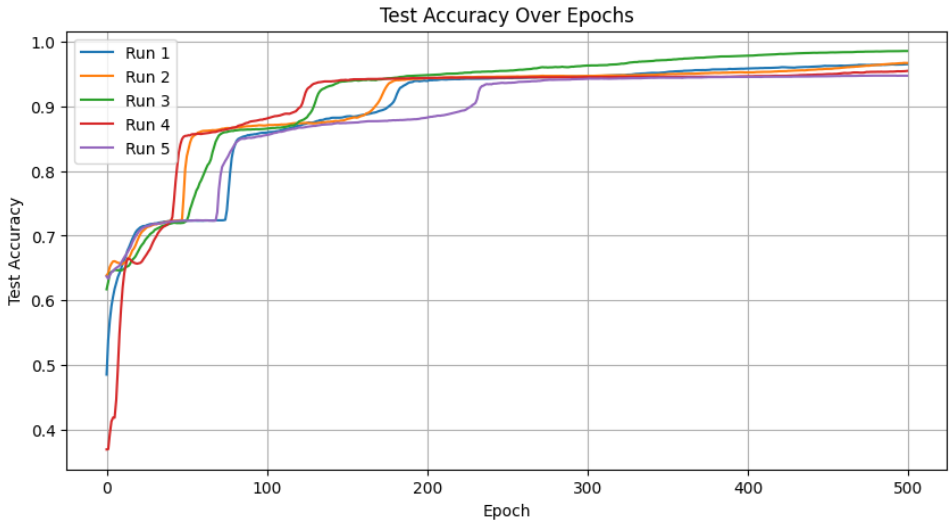
i. Train your model using 5 random seeds and report the mean and standard deviation of the train and test accuracy at the end of 500 epochs. Attach the figures of training loss and testing accuracy change over epochs (a total of 2 figures is expected, please combine the results over 5 runs in the same plot).

Mean of Train Accuracy: 96.60%. Mean of Test Accuracy: 96.41%. Standard Deviation of Train Accuracy: 1.35%. Standard Deviation of Test Accuracy: 1.29%

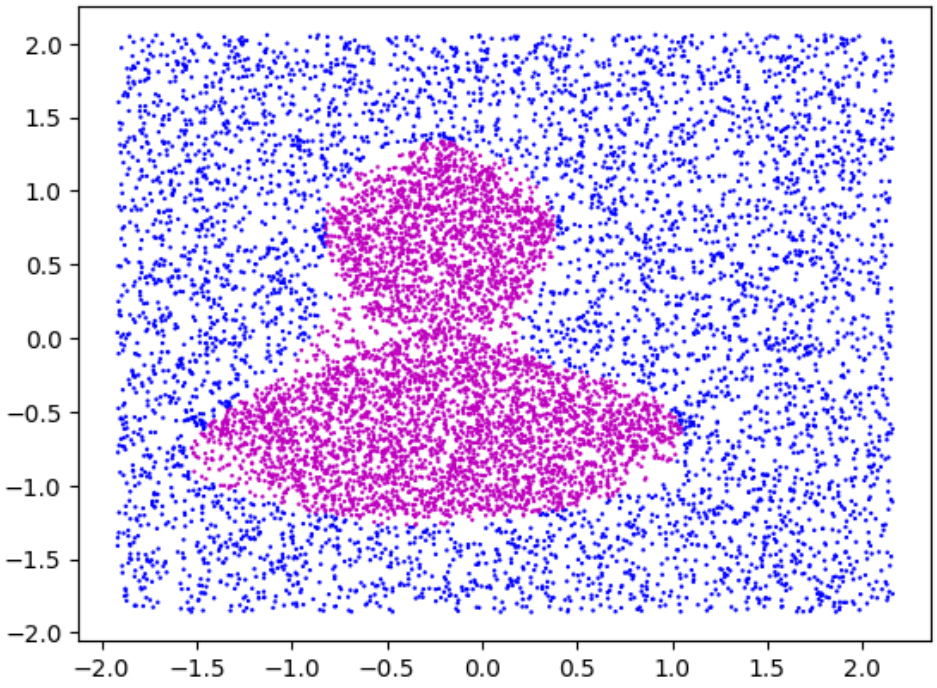


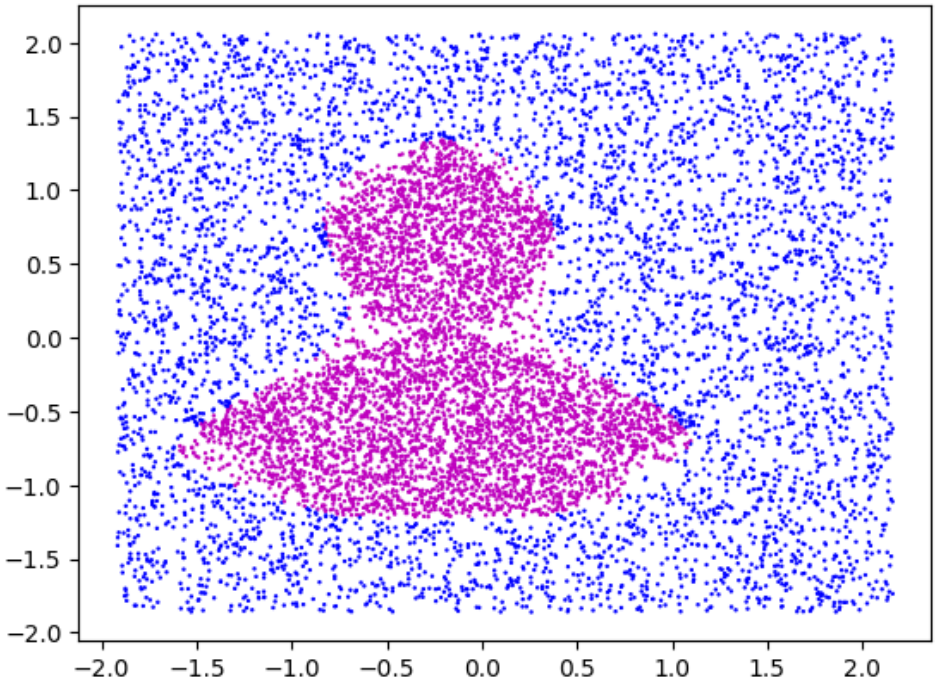
[1] `https://pytorch.org/`, `https://pytorch.org/get-started/locally/`. If you have not programmed with PyTorch before, give yourself some time to go through some online tutorials. There are many on the internet so it is hard to recommend a specific one, it would really depend on your level of comfort. You may also find it helpful to review the PyTorch resources on Piazza (`https://piazza.com/jhu/spring2023/cs482682/resources`).

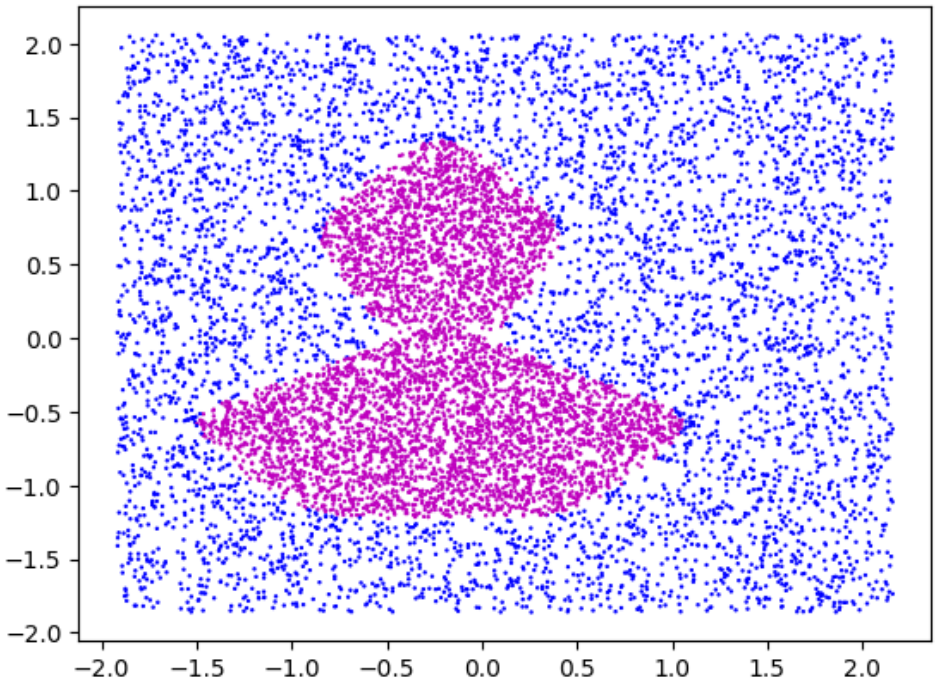
[2] See discussion here https://stackoverflow.com/questions/53628622/loss-function-its-inputs-for-binary-classification-pytorch

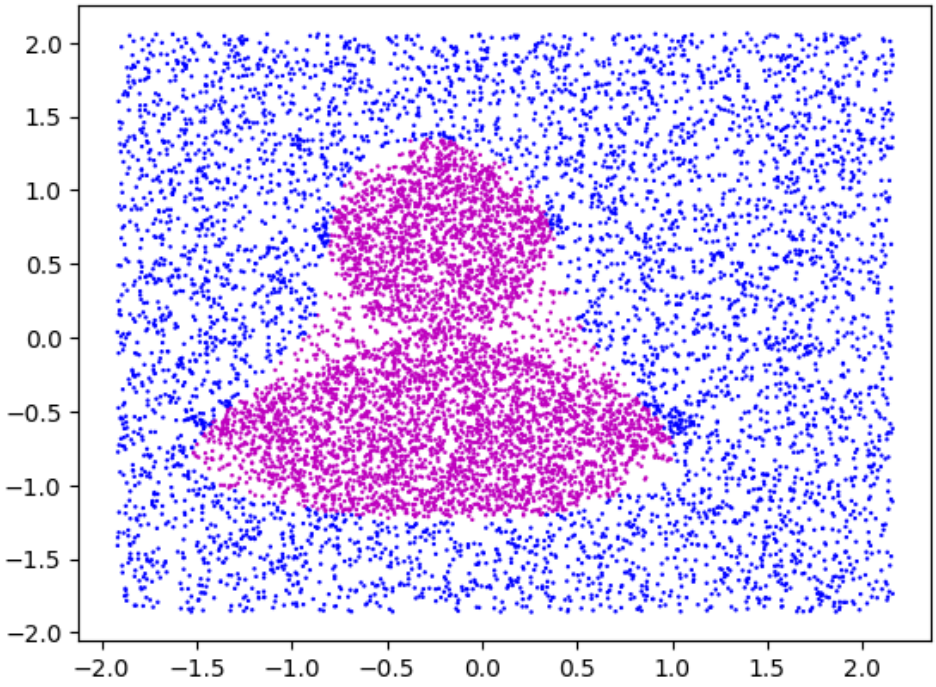


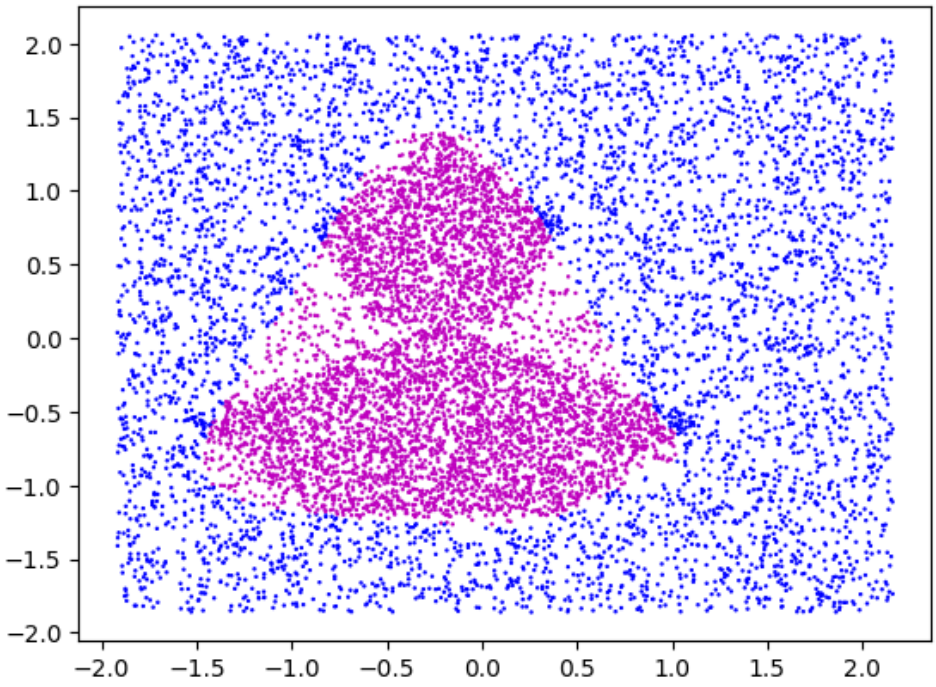
ii. In your report, attach the visualizations of the prediction of polygon figure (5 figures).




2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0





iii. Brief discussion. Explain whether you are able to find a solution that performs almost as well as the "manual" solution in (a) and why or why not.

I was able to find a solution that was almost as good as the manual solution. The polygons were successfully classified, the edges were a little rough and the model had a hard time distinguishing between whether the points were in the polygon or not.

(c) Build an MLP with a larger capacity (increase the depth and width). *Note: Since this is a toy dataset, you might not actually observe differences in performance compared to your 1b solution. We do expect a reasonable accuracy of the modified model. However, you should be able to reason about expected behavior based on our lecture discussions.* You may increase the epoch number if you observe that your model does not converge by 500 epochs due to the additional parameters.

i. What do you expect to happen to your model's ability to learn the appropriate

decision boundaries when you increase the depth? Increase the width?

I expect the accuracy to go up by increasing the depth and the width. When you increase the depth, you are able to capture more complex patterns in the data. When you increase the width, you are able to capture more features. With these two changes, I expect a better classification.

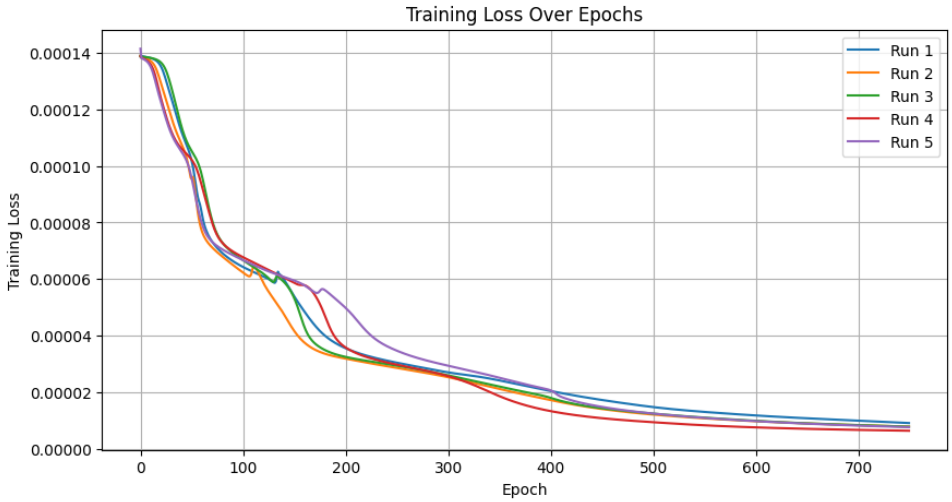
ii. Like before, train your model using 5 random seeds and report the mean and standard deviation of the train and test accuracy at the end of 500 epochs.

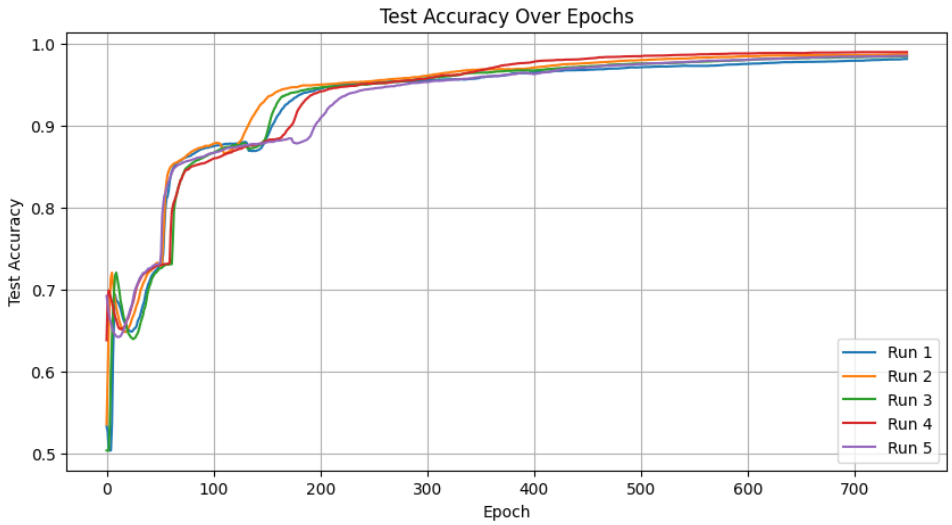
Mean of Train Accuracy: 99.08%. Mean of Test Accuracy: 98.56%. Standard Deviation of Train Accuracy: 0.14%. Standard Deviation of Test Accuracy: 0.28%

iii. Report what depth and width you used.

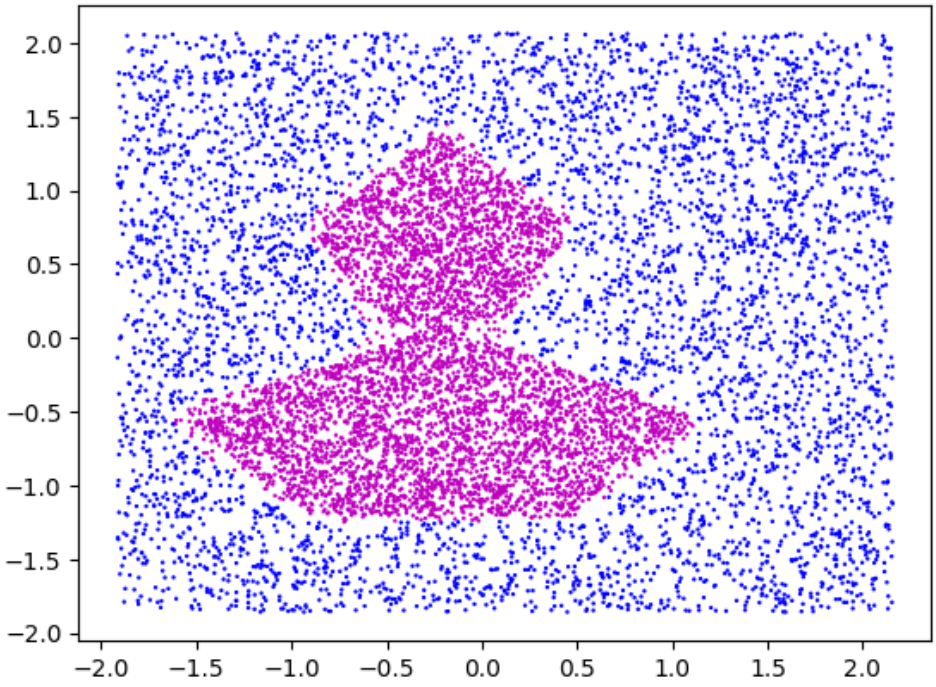
My layer dimensions were [32, 16, 8]. I changed my depth from 2 to 3 and my width for each layer is 32 neurons for the first, 16 neurons for the second, and 8 neurons for the third layer.

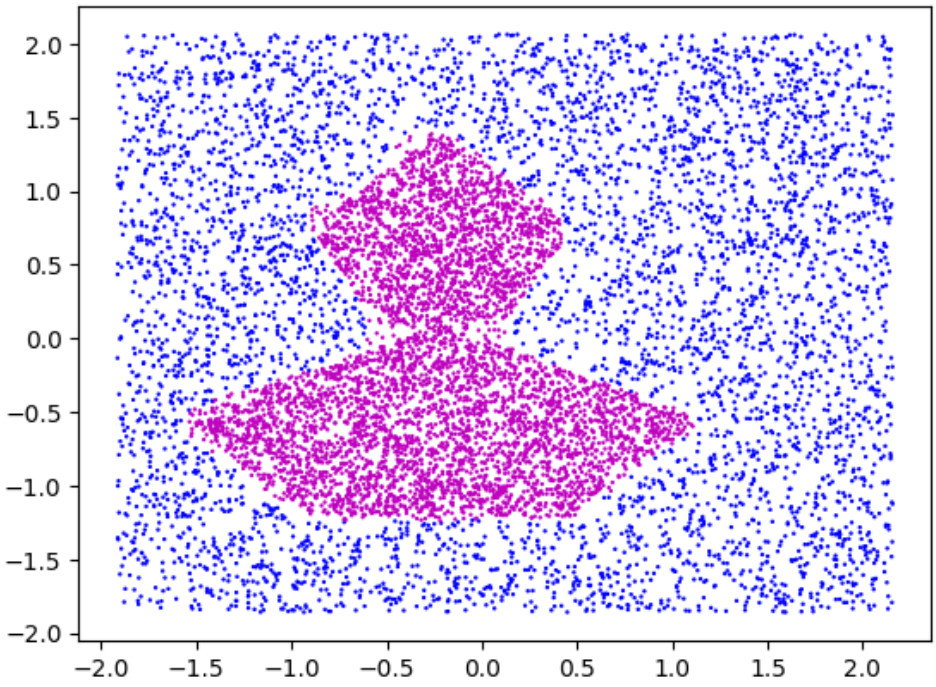
iv. Attach the plots of training loss and testing accuracy change over epochs.

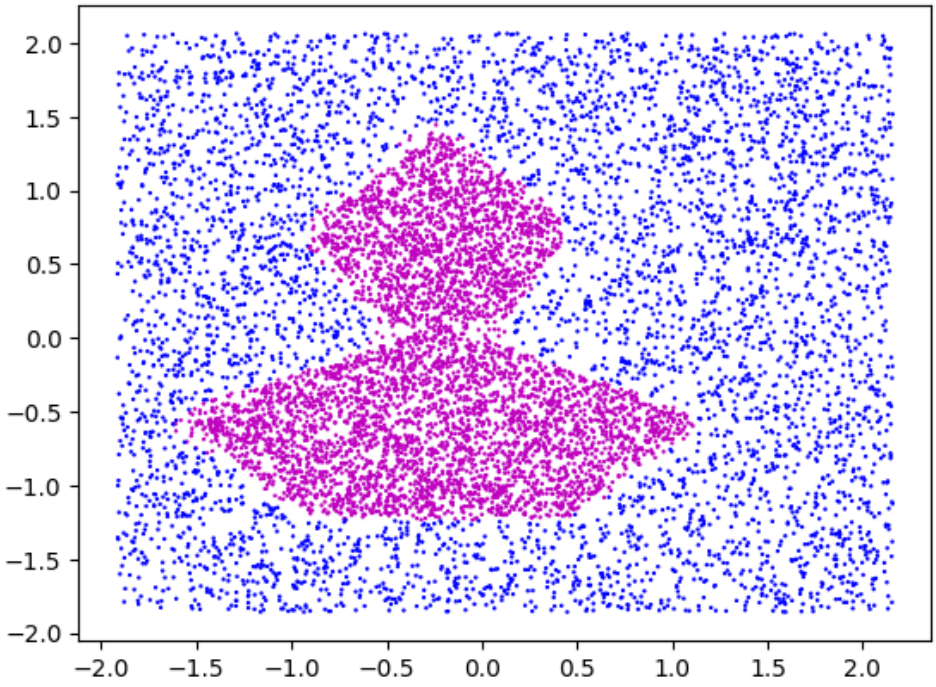


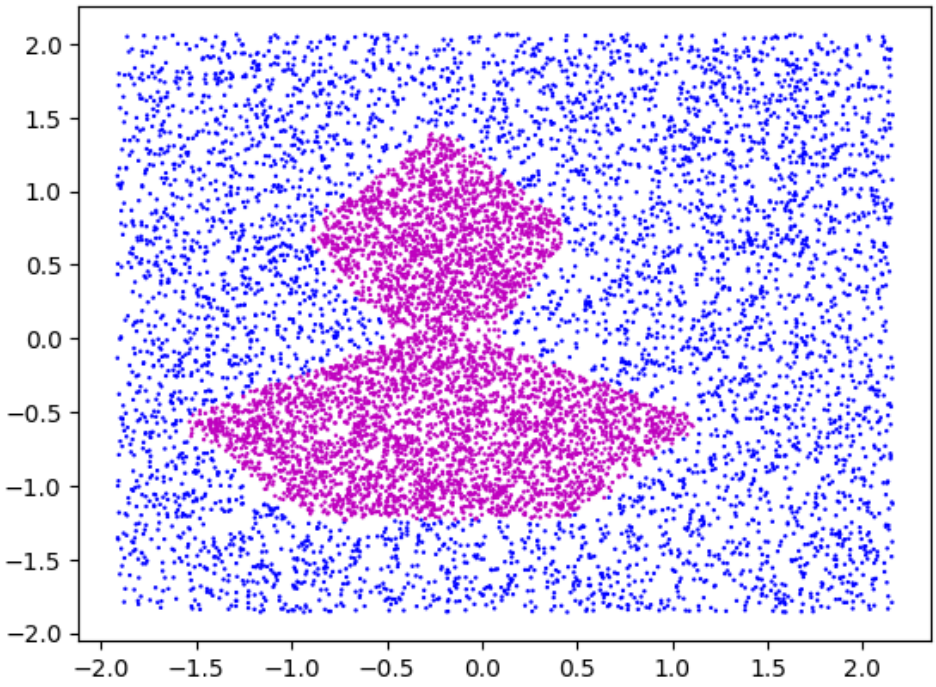


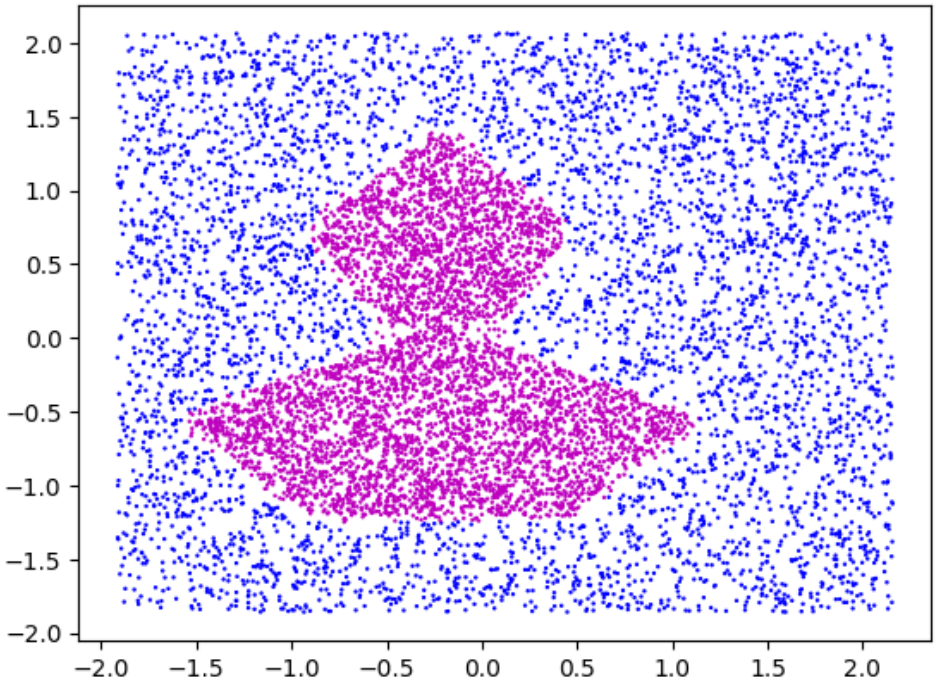
v. Attach the visualization of the prediction for the polygon figure.


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0



vi. Briefly discuss what you notice about your model performance as you adjust the depth and width of the model. *Hint: Think about our discussions from the lecture regarding model capacity.*

My model's performance did increase with the change of width and depth. The lines of the polygons are more crisp. This makes sense since with adding width and depth we are able to make more complex decisions and capture more features.

(d) In part (b) we had fixed the hyperparameter choices, but in part (c) you adjusted those choices for the MLP in terms of depth and width of the network. Would you consider your test results to be valid and generalizable? Please briefly discuss.

I would consider my test results valid and but not generalizable. I think my model does very well on this data set with polygons, but if you introduce different shapes I am not sure how effective my model would be. I would need to train and test using additional data sets.

2. In this problem, we will explore convolutional neural networks (CNNs). We will be working with the FashionMNIST dataset.[3] This dataset only has a train-test split, therefore we will be using the last 10000 training instances as the validation set. Please use PyTorch for this assignment. Because training times can be quite long, you *do not* need to train your model with multiple random seeds — 1 is enough, but of course feel free to expand if you have time and are confident in your model. You should train your models for at least 50 epochs.

(a) Design a small CNN (e.g. 3-4 hidden layers) using the tools we learned in class such as

i. convolution and pooling layers

ii. activation functions (other than sigmoid)

TODO:

i. **Briefly** describe your architecture in your report and explain your design choices (e.g. I used two-layer-network/ReLU activation because...) **Please design the network architecture by yourself. This is not about performance.**

For this problem, I decided to use 2 convolution layers, 1 pooling layer, and 2 fully connected layers. In total, I have 4 hidden layers with one of the fully connected layers being the output layer. I used two

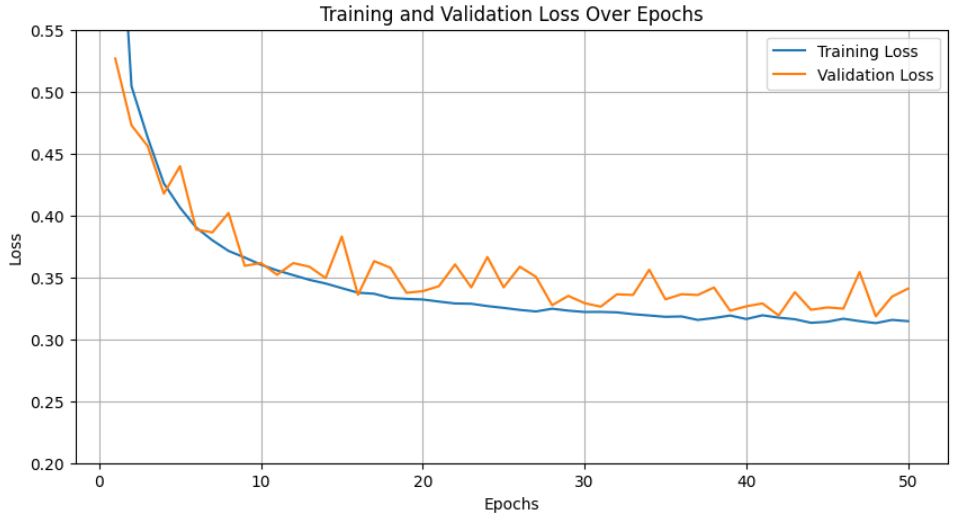
[3]The full FashionMNIST dataset can be downloaded from the official website here: `https://github.com/zalandoresearch/fashion-mnist`. We provide a starter code for loading and splitting the dataset in the notebook.

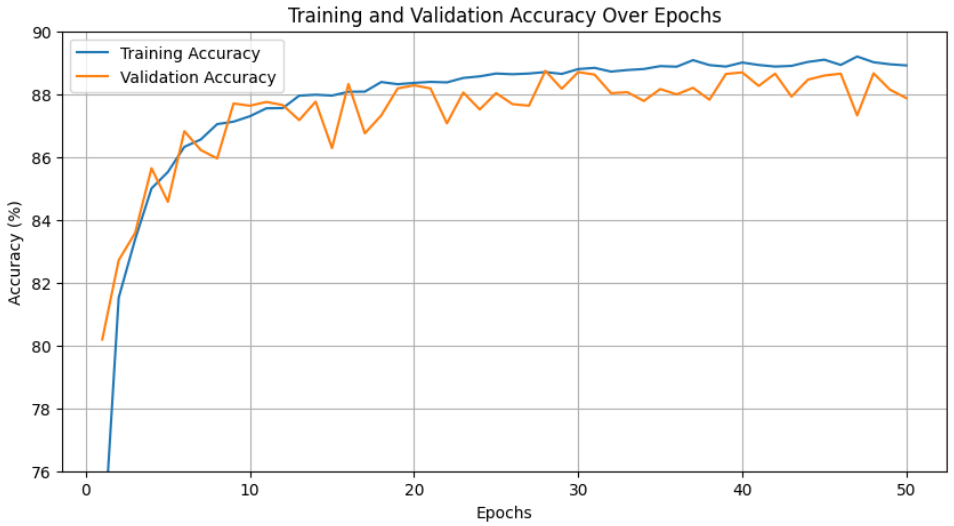
convolution layers because I thought that would be the optimal number for training this data set. The pooling layer connects the two convolution layers together and also connects the convolution layer to the fully connected layers at the end. Pooling layers are responsible for decreasing dimensions in the layers which helps to capture the most dominant features in the data set. The fully connected layers help us to make classifications. I used a relu activation function because I wanted to use something I was comfortable with for this problem. The optimizer I chose was Adam with a weight decay of 0.005 and I am using cross entropy loss. I wanted to experiment using different optimizers and using Adam with a weight decay paired with cross-entropy loss gave me the best accuracy.

ii. Report the train, validation, and test accuracy of your best model (e.g. if the best model was obtained at epoch 50, report the train, validation, and test accuracy at epoch 50).

My best train accuracy was 88.20%, my best validation accuracy was 88.74% and my best test accuracy was 87.56%.

iii. Attach the plots of training loss and validation accuracy change over epochs.





Note: If you achieve more than 85% accuracy on the test set, move on to the next subproblem.

(b) Now, try to improve your model's performance by including additional architecture elements. You should implement at least two of the following:

i. dropout

ii. batch normalization
iii. data augmentation
iv. different optimizers (other than vanilla stochastic gradient descent)

Try to improve your accuracy over the model in 2(a). Your new model should perform at least as well as your previous model.

TODO:

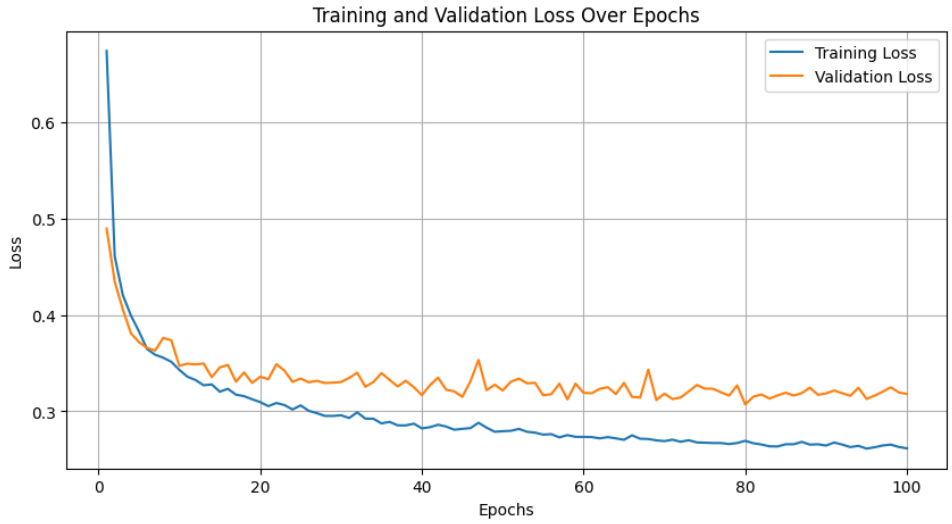
i. Describe your new model additions and explain your design choices (e.g. I added dropout because...).

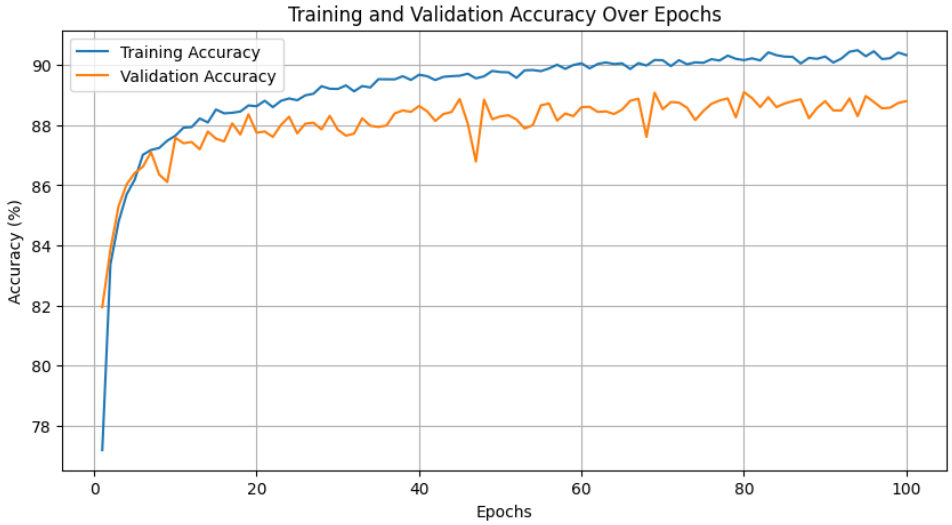
I decided to add some data augmentation because my model was plateauing around 86% so I thought adding some random rotations to the data set would allow for better classifications. I added batch normalization because I felt the model wasn't converging fast enough. I also changed the optimizer and used stochastic gradient descent with momentum because in training I found that it updated my weights better. I also decided to train for more epochs (100) and added a scheduler to help with the plateauing issue.

ii. Report the training, validation, and test accuracy of your best model.

My best train accuracy was 90.49%, my best validation accuracy was 89.10% and my best test accuracy was 88.08%.

iii. Attach the plots of training loss and validation accuracy change over epochs.





iv. Briefly discuss if your choice of additional elements achieves the results you expected and why. Is it possible to distinguish the effects of the different elements you modified in your model? Briefly explain why or why not.

The additional elements I added did do what I expected them to do. My second model did not work as well as I would've liked it to though. I was still plateauing but around 88% this time. Adding the batch normalization and the scheduler really helped to increase my accuracy and decrease my loss. I noticed when my accuracy and loss were too similar after a few epochs, the learning rate would decrease allowing the model to continue to improve.

(c) Search the internet for models that others have built on FashionMNIST. This can be from research papers or even something like Kaggle. Briefly describe one technique or architecture decision that you found interesting (remember to cite your sources). It can be an extension of an existing method, and you don't need to cover the paper's results for that method. As a rough guide, anything from approximately 50-100 words of prose is sufficient.

The paper I researched using LeNet-5 Architecture for the Fashion MNIST data set. The LeNet-5 architecture has three convolution layers, two pooling layers, and two fulling connect layers. I thought this architecture was interesting because I did something similar in 2b without realizing and didn't get the same accuracy they did in the paper. I maxed out at 88% accuracy while their architecture achieved at least 99% accuracy for all labels.

References

[1] M. Kayed, A. Anter, and H. Mohamed, "Classification of garments from fashion mnist dataset using CNN lenet-5 architecture," 2020 International Conference on Innovative Trends in Communication and Computer Engineering (ITCE), 2020. doi:10.1109/itce48509.2020.9047776