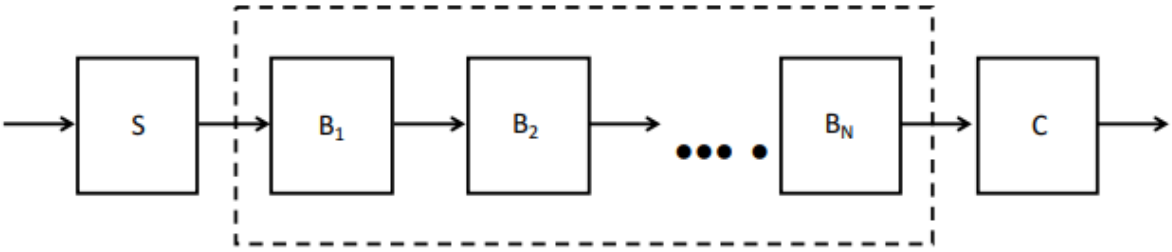
ECS659U/P: NEURAL NETWORKS & DEEP LEARNING

Aris Christofides – 210911099

For this coursework I had to create a classification model for the Fashion-MNIST dataset that classifies every image in terms of 1 out of 10 classes. This report will include many models that have been tested as well as the different parameters and inputs used to find the optimal model. The code is well documented and filled with comments to briefly explain what every part does.

The model consists of a **Stem**, a **Backbone** with **2 Blocks** which contain 2 MLPs each **(Total 4 MLPs)**, and a **Classifier.**




```
### MLP 1  SETUP ###
self.Block1_Linear1 = nn.Linear(output1, output2)
self.Block1_Linear2 = nn.Linear(output2, output3)

### MLP 2 SETUP ###
self.Block1_Linear3 = nn.Linear(output4, output5)
self.Block1_Linear4 = nn.Linear(output5, output6)

#### BLOCK 2 ####
self.Block2_ReLU1 = nn.ReLU()
self.Block2_ReLU2 = nn.ReLU()
self.Block2_ReLU3 = nn.ReLU()
self.Block2_ReLU4 = nn.ReLU()
### MLP 1  SETUP ###
self.Block2_Linear1 = nn.Linear(output6, output7)
self.Block2_Linear2 = nn.Linear(output7, output8)
### MLP 2 SETUP ###
self.Block2_Linear3 = nn.Linear(output8, output9)
self.Block2_Linear4 = nn.Linear(output9, output10)
```

To test out my models and compare them, I tested out different outputs which are defined after the neural network **class**. In addition to changing the outputs, I tried out different optimizers, such as Adam, RAdam and SGD and altered their learning rate. Also tested 1, 2 and 3 blocks. I left the **patch size = 16** because it seemed to get me the better results.

```
optimizer = torch.optim.RAdam(net.parameters(), lr=0.001)
```

Code Resources:

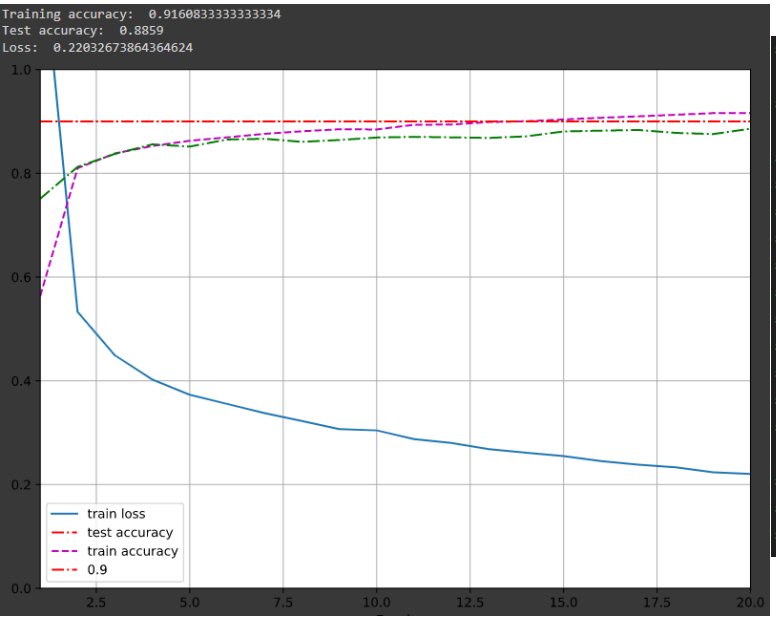
https://pytorch.org/docs/stable/optim.html

https://pytorch.org/docs/stable/nn.html

https://towardsdatascience.com/build-a-fashion-mnist-cnn-pytorch-style-efb297e22582

https://pytorch.org/docs/stable/index.html

Best Model

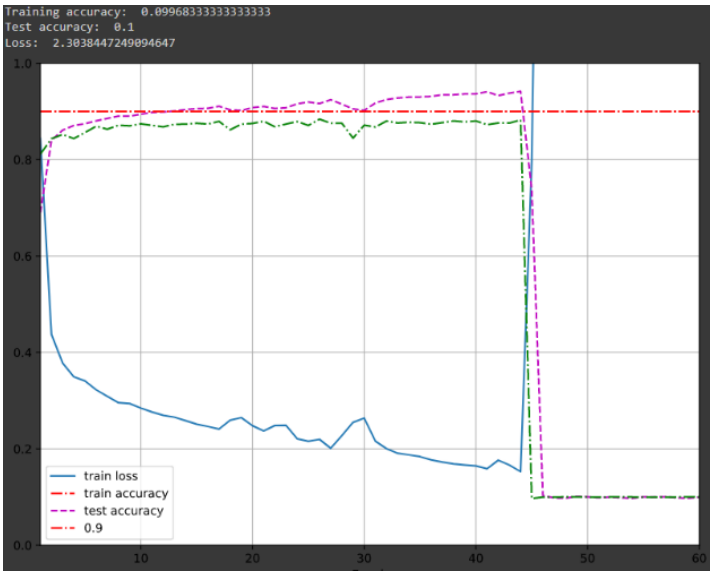



```
# Test(16) with 2 blocks 4 MLP - Optimizer RAdam (lr=0.001) - Epochs = 20

# patch size = 16

# output1 = 49
# output2 = 256
# output3 = 128
# output4 = 49
# output5 = 80
# output6 = 60
# output7 = 32
# output8 = 128
# output9 = 100
# output10 = 50
# classes = 10

# Training accuracy:  0.9160833333333334
# Test accuracy:  0.8859
# Loss:  0.22032673864364624
```

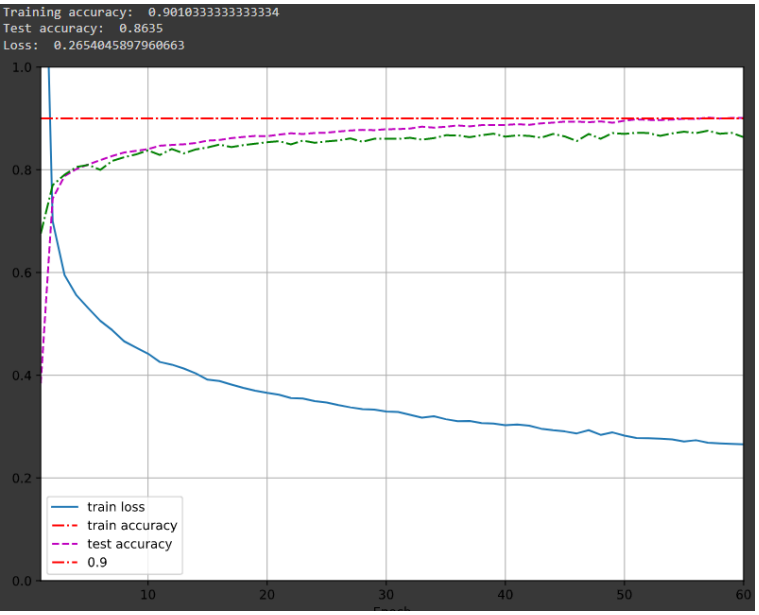
This model got me the highest accuracy. I tried increasing the epochs, different learning rate, as well as different optimizer, but this combination got me the best result.

Tested Models

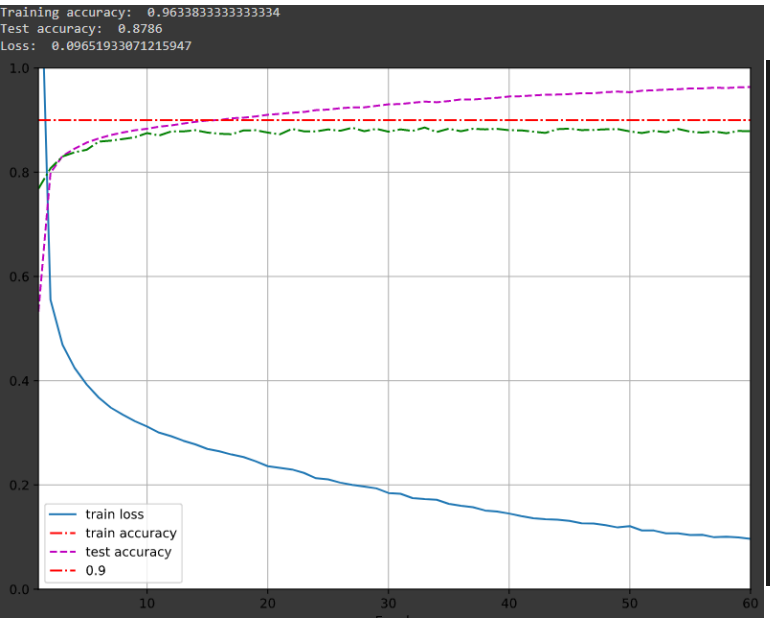
Here are some models that I have tested to achieve my best model.



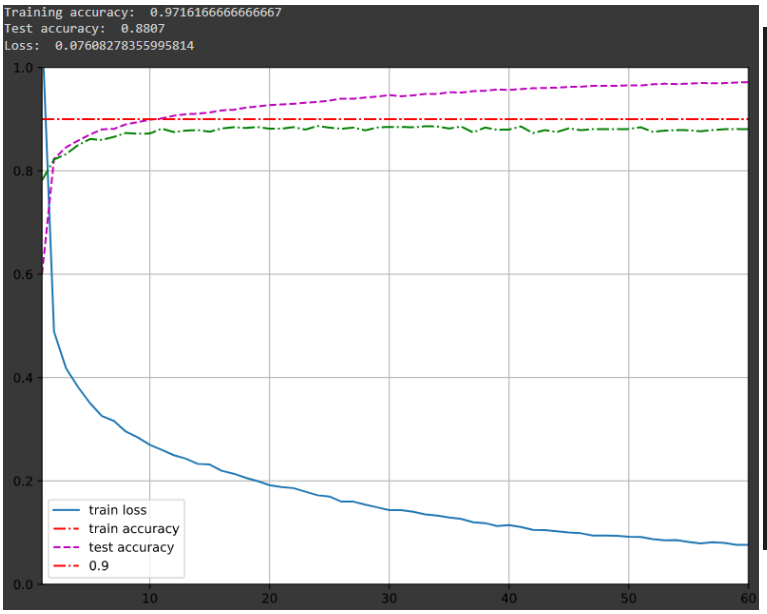

```
# Test(1) with 2 blocks 4 MLP - Optimizer RAdam (lr=0.01) - Epochs = 60

# patch size = 16

# output1 = 49
# output2 = 256
# output3 = 128
# output4 = 49
# output5 = 64
# output6 = 32
# output7 = 16
# output8 = 128
# output9 = 64
# output10 = 32
# classes = 10

# Training accuracy:  0.09968333333333333
# Test accuracy:  0.1
# Loss:  2.3038447249094647
```




```
# Test(2) with 1 block 2 MLP - Optimizer RAdam (lr=0.001) - Epochs = 60

# patch size = 16

# output1 = 49
# output2 = 250
# output3 = 200
# output4 = 49
# output5 = 150
# output6 = 100
# classes = 10

# Training accuracy:  0.9010333333333334
# Test accuracy:  0.8635
# Loss:  0.2654045897960663
```




```
# Test(6) with 2 blocks 4 MLP - Optimizer RAdam (lr=0.001) - Epochs = 60

# patch size = 16

# output1 = 49
# output2 = 256
# output3 = 128
# output4 = 49
# output5 = 64
# output6 = 32
# output7 = 16
# output8 = 128
# output9 = 64
# output10 = 32
# classes = 10

# Training accuracy:  0.9633833333333334
# Test accuracy:  0.8786
# Loss:  0.09651933071215947
```




```
# Test(12) with 2 blocks 4 MLP - Optimizer RAdam (lr=0.002) - Epochs = 60

# patch size = 16

# output1 = 49
# output2 = 256
# output3 = 128
# output4 = 49
# output5 = 80
# output6 = 60
# output7 = 32
# output8 = 128
# output9 = 100
# output10 = 50
# classes = 10

# Training accuracy:  0.9716166666666667
# Test accuracy:  0.8807
# Loss:  0.07608278355995814
```

All tests are included in the Jupyter Notebook.