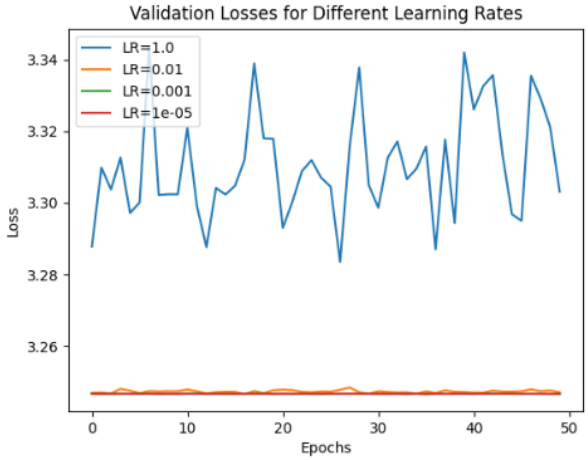
ML Methods Ex4 Report – Yehuda Frist

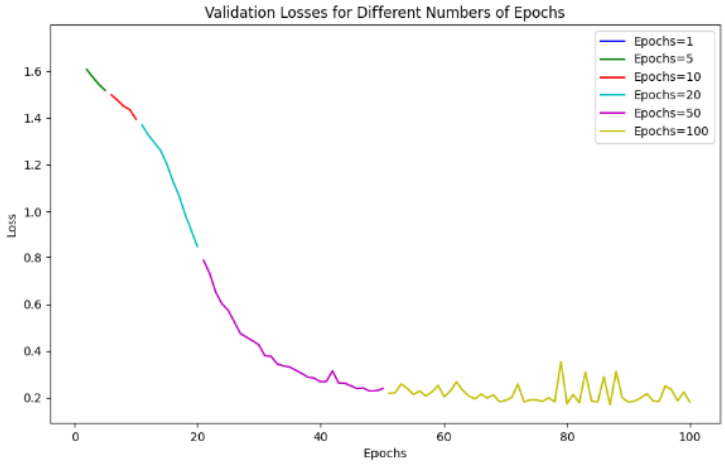
6 Multi-Layer Perceptron's –

6.1.2 Optimization of an MLP - Questions:

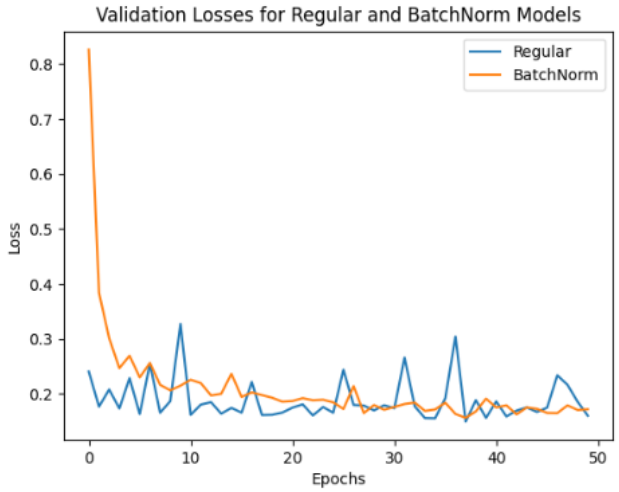
1. A too high learning rate can lead to oscillations of the loss. In our case, the validation loss moving around 3.3 suggests that the model's weights are being updated too drastically, leading to instability. The lower learning rates allow for more stable learning as we can see in the graph.



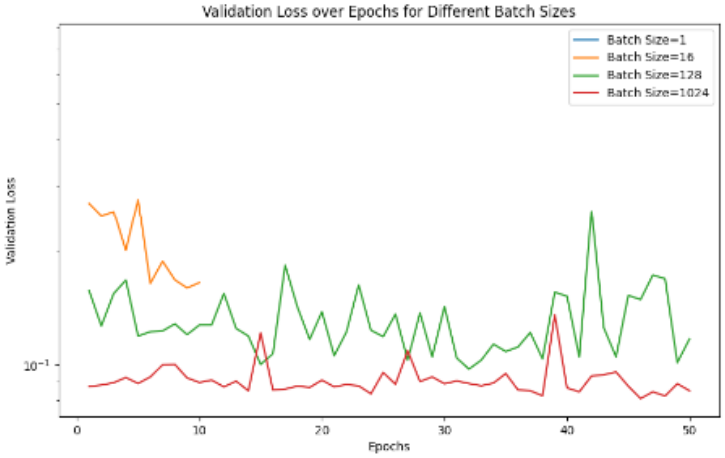
2. The effect the number of epochs has is that for too few epochs the model won't have enough "time" to learn the patterns in the data and the model's losses will be relatively high. On the other hand, after too many epochs we can expect the training to lead to overfitting (as seen in our graph at approximately the 80-epoch mark). In general, increasing the number of epochs will allow the model to decrease the loss and improve its performance.



3. As seen in our graph, although the BatchNorm graph's validation losses start higher, we can see that batch normalization helps stabilize the training process. This can lead to faster convergence than the Normal model which can help the model learn more effectively.

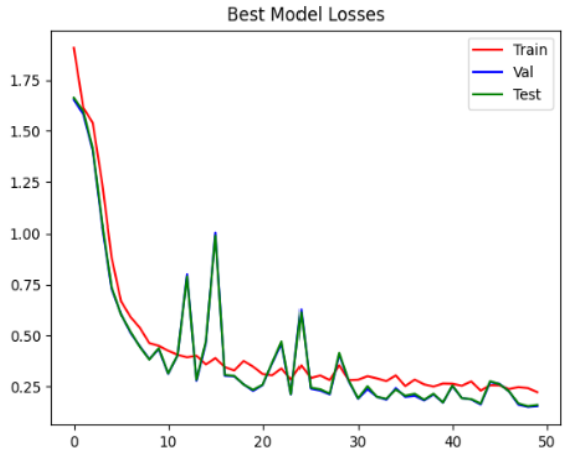


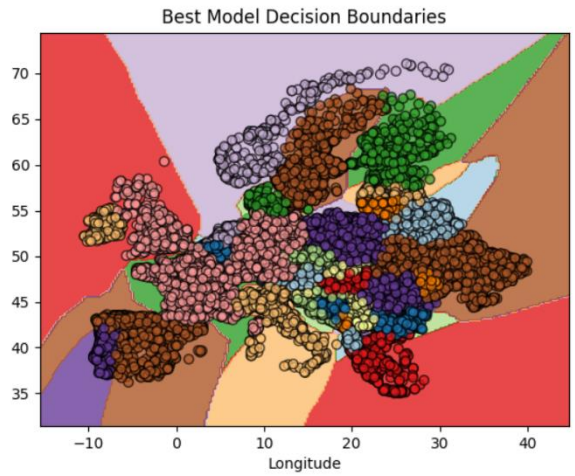
4. Test Accuracy – Generally, larger batch sizes achieved higher test accuracy.
   Training Speed – Generally, smaller batch sizes resulted in slower training speeds per epoch.
   Stability – As we can see in the graph, smaller batch sizes result in less stable loss as apposed to higher batch sizes which perform a more stable training process with more stability in their losses.
   - *I have attached below the test results.*



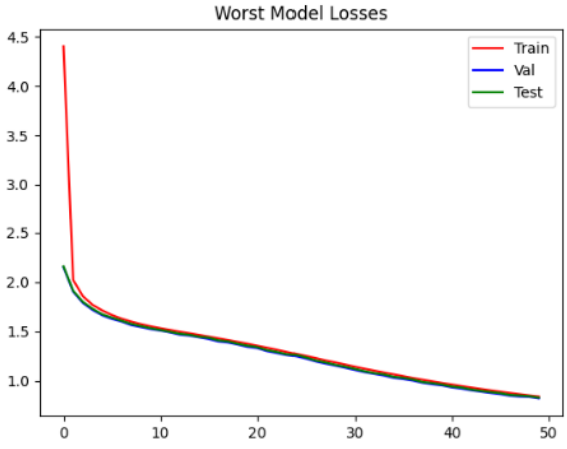
6.2.1 Evaluating MLPs Performance - Questions:

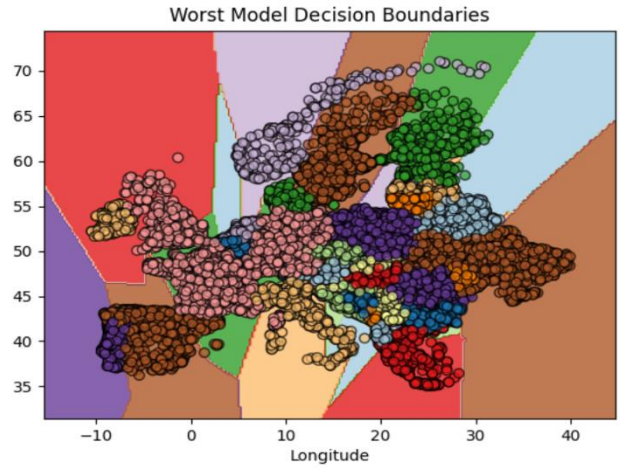
1. Below are the training, validation, and test losses. As we can see the losses converge to a rather low number (around 0.25) indicating that the model generalized well.



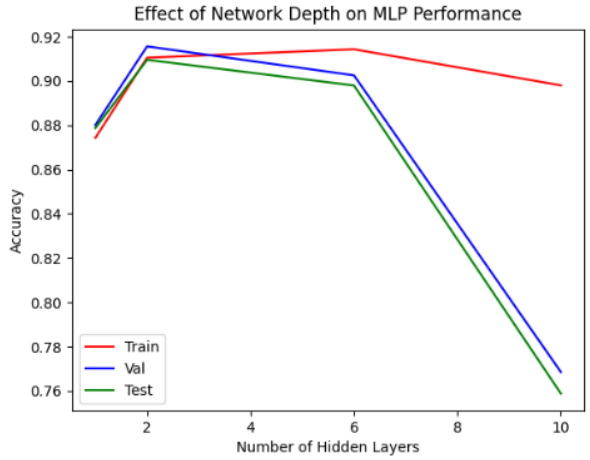


2. This model also generalizes well but as we can see from the graph, its losses converge to a higher number (1.0) than the best model indicating that it doesn't generalize as well as the best model.

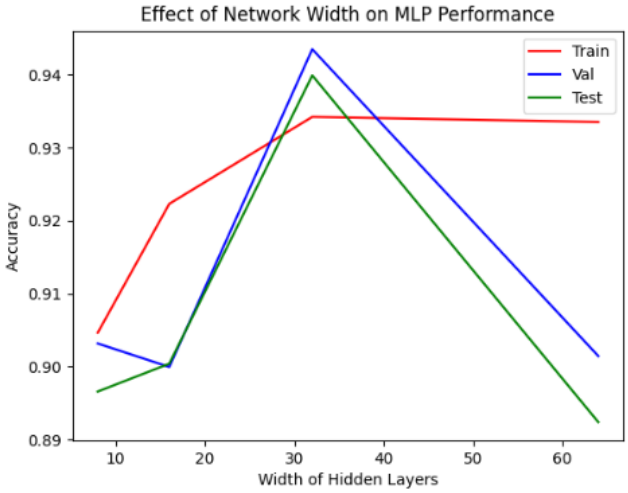




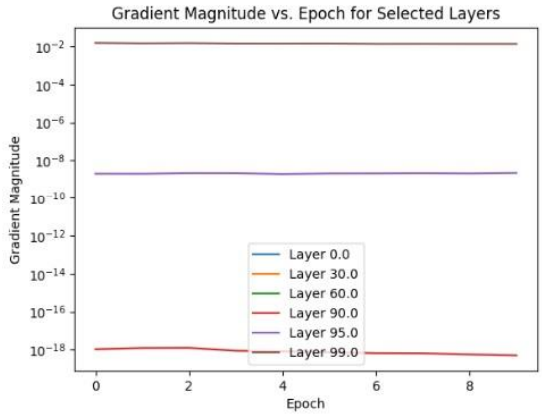
3. In our graph we can see that the accuracy peeks at 2 hidden layers and then drops. This suggests that adding hidden layers past that point is harmful to the model's performance. This phenomenon is known as "The vanishing gradient problem".



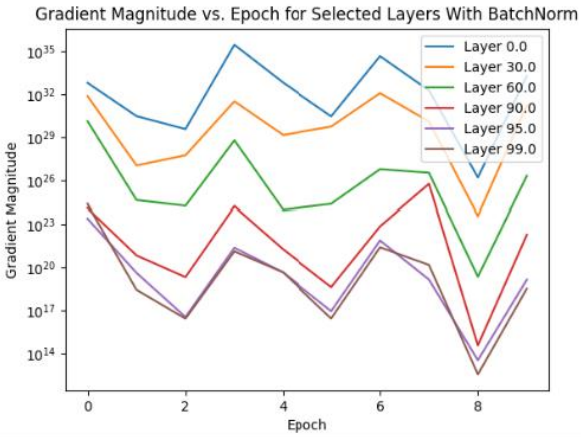
4. As we can see in our graph, the Validation and test accuracy peek at about 30 hidden layers. This suggests that with a width of around 30 neurons per hidden layer, the model generalizes best. We can also observe the steep drop in accuracy fro more or less neurons per hidden layer. Indication that around 30 is the optimal number of neurons to achieve the best generalization.



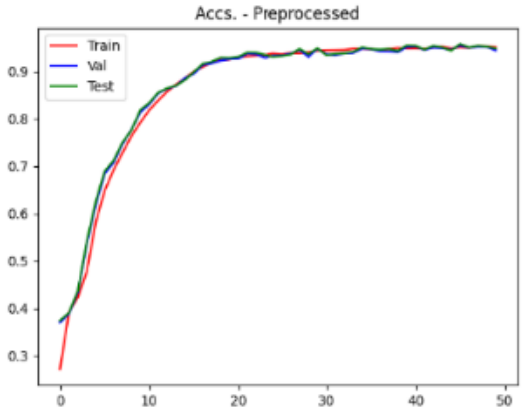
5. We can observe that vanishing gradients occur as expected, in the first layers the gradients magnitude “vanishes”. To solve this problem, we can add batch normalization layers to normalize the inputs to each layer. This can help stabilize the gradients through its training.

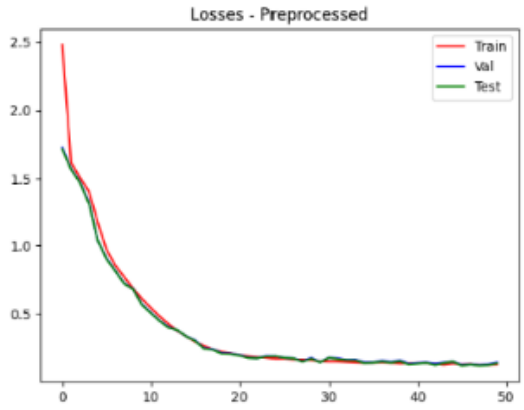


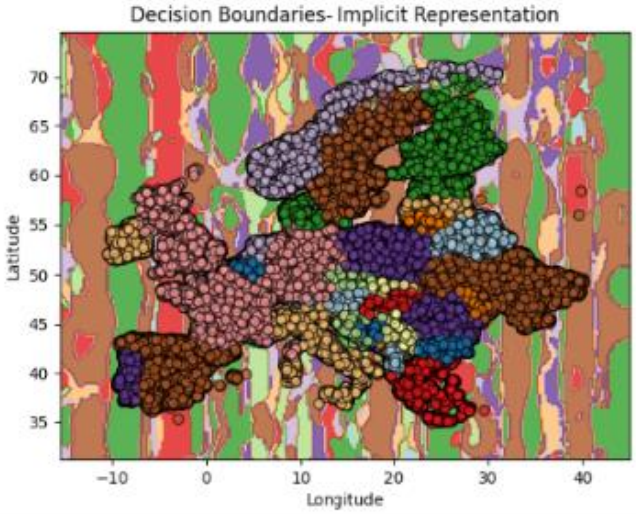
6. After implementing my suggestion of adding batch normalization layers we can see that the gradients no longer vanish rather they “explode” which also indicates issues with the training process.



7. As we can see from both plots, the model's losses converge to 0 and its accuracies to 1. This indicates that that model is effectively learning the underlying patterns in the data. This can also be seen in the intricate decision boundaries the model has decided. By representing the data with sin and cosines, the model can learn more complex patterns than a model with similar architecture on the standard form of the data. This can be seen clearly by the decision boundaries plotted by the implicit representation.







7 Convolutional Neural Networks –

7.2 Task

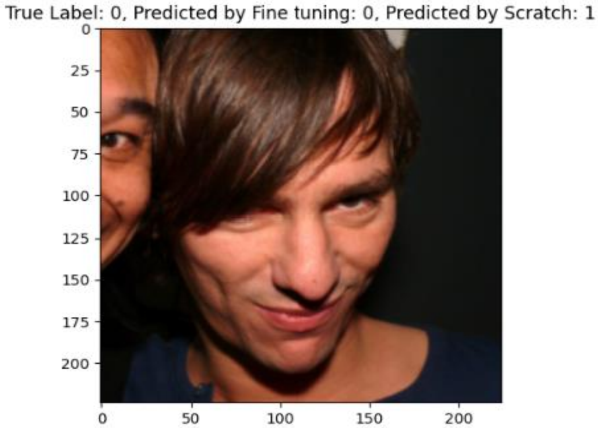
```
Q1 - XGBoost with default params
Accuracy: 73.5%

Q2 - training from scratch
Epoch 1/1, Loss: 0.7758, Val accuracy: 0.5250
Test Accuracy: 0.5225
Learning Rate: 0.01

Q3 - linear probing
Epoch 1/1, Loss: 0.6795, Val accuracy: 0.6950
Test Accuracy: 0.7250
Learning Rate: 0.01000


Q4 - linear probing based on sklearn (lr =0.01)
Epoch 1/1, Loss: 0.6795, Val accuracy: 21.2800
Train Accuracy: 0.7557
Val Accuracy: 0.6750
Test Accuracy: 0.7300

Q5 - fine tuning
Epoch 1/1, Loss: 0.6244, Val accuracy: 0.7400
Test Accuracy: 0.7750
Learning Rate: 0.00100
```

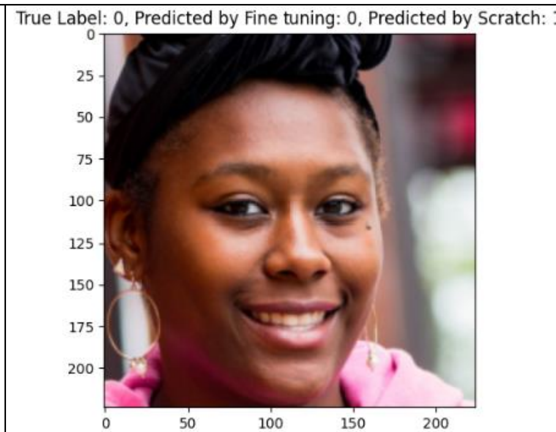
7.4 Questions

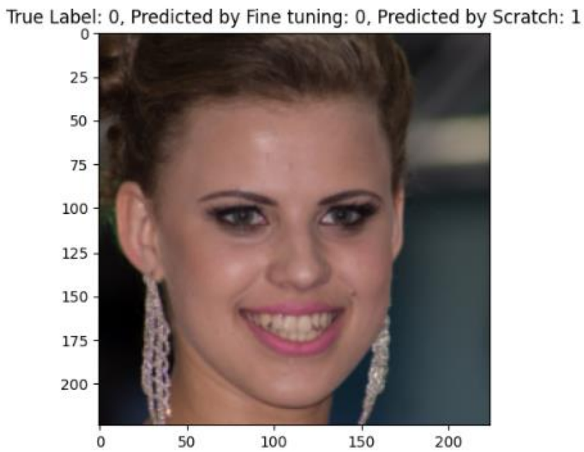
1. The learning rates I used for the PyTorch base line were lr = [0.00001, 0.0001, 0.001, 0.01, 0.1].
   The 2 best models are:
   - Fine tuning with lr=0.001 resulting in a test accuracy of 0.755 after a single epoch.
   - Linear probing based on sklearn with lr=0.01 resulting in a test accuracy of 0.73 after a single epoch.

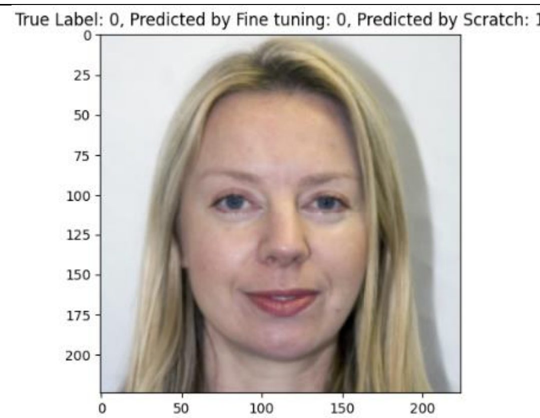
   The worst model was the model training from scratch which achieved its highest test accuracy of 0.5225 with a lr=0.01.

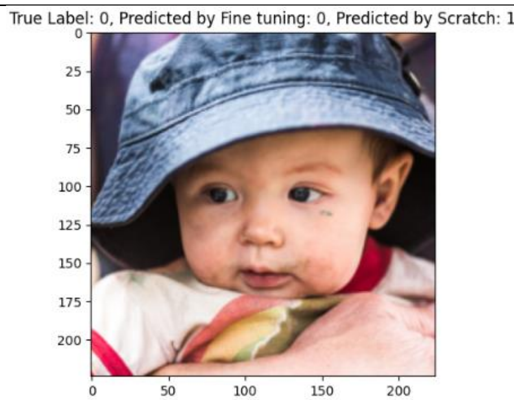
2. **5 samples correctly classified by the Fine-tuning model and misclassified by the model trained from scratch:**











Appendix:

6.1.2.4 – Batch Size -

```
Epoch 1, Train Acc: 0.622, Val Acc: 0.686, Test Acc: 0.678
Batch Size: 1, Time: 152.70 seconds

Epoch 10, Train Acc: 0.915, Val Acc: 0.934, Test Acc: 0.928
Batch Size: 16, Time: 165.87 seconds

Epoch 50, Train Acc: 0.955, Val Acc: 0.950, Test Acc: 0.948
Batch Size: 128, Time: 179.67 seconds

Epoch 50, Train Acc: 0.967, Val Acc: 0.969, Test Acc: 0.967
Batch Size: 1024, Time: 111.76 seconds
```