Report for exercise 4

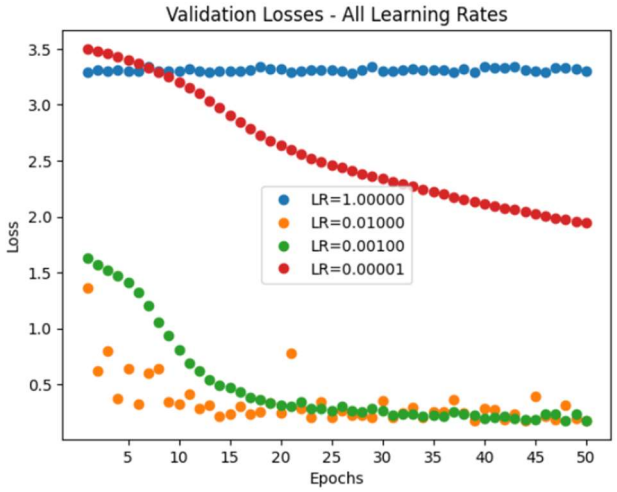
Dana Aviran

Optimization of an MLP

Question 1:

Train the network of depth 6 and width 16 on the given dataset (longitude and latitude for classifying to countries) with learning rates of 1, 0.01, 0.001, 0.00001. What does a too high learning rate do? What about a too low one? Explain.

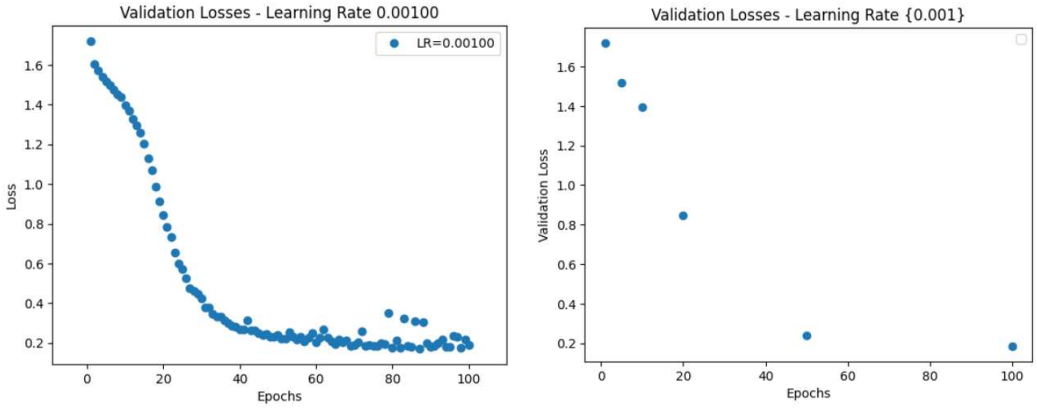
Answer:



It appears in the graph that the learning rate of 1 causes the validation loss to stay the same during epochs, keeping the initial high validation loss of about 3.4. Thus, it looks like the gradient descent optimization overshoot the minimum of the loss function, taking too big steps at each epoch while keeping the validation loss value high. Generally, the learning rate determines the step size the model takes when updating its weights based on the error it makes. Therefore, high learning rates can cause the model parameters to update too aggressively, potentially causing them to overshoot optimal values or even diverge to infinity. On the other hand, it appears that the low learning rate of 0.00001 did not lead to convergence to minimal validation loss during the 50 epochs. In this case, the gradient descent process progresses very slowly since each step is very small, so it takes many epochs to converge. Generally, too small learning rates might never lead to convergence because they might get the algorithm to converge to a local minimum which they are too small to recognize as noise in the data. In conclusion, there is a tradeoff between too high and too low learning rates, where we strive to pick a learning rate that is high enough so it can navigate more significant changes in the landscape and potentially reach the global minimum, but also not too high so it doesn't overshoot the correct path.

Question 2:

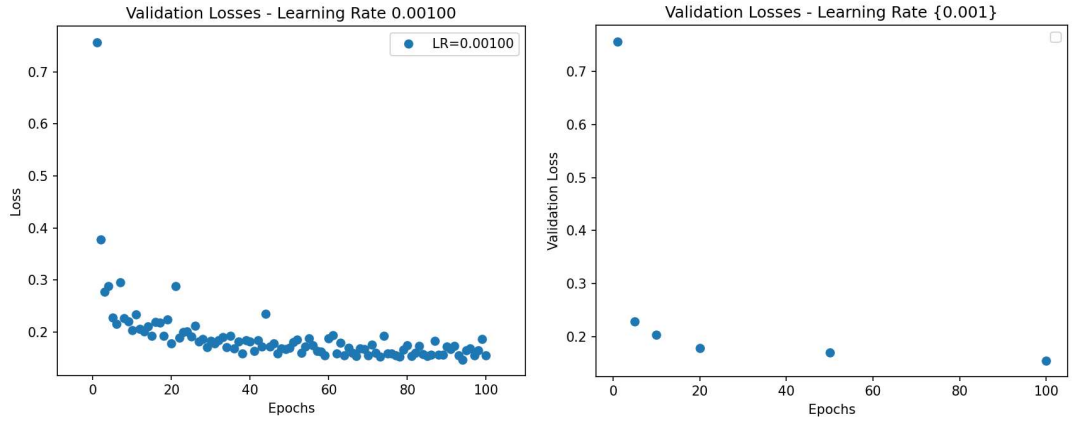
For the model of 0.001, train the network the following epochs: 1, 5, 10, 20, 50, 100. How does the number of epochs affect the training? What happens after too many epochs? How does too little epochs affect? Explain.



In our example, we can see that the more epochs we make our validation losses get smaller. We can see that 50 epochs were not enough because the validation loss kept getting smaller. Generally, increasing epochs leads to improved performance as the model gets more exposure to the training data and has more chances to adjust its weights and learn the underlying patterns. Thus, too few epochs might lead to underfitting since the model doesn't have enough training time to learn the complexities of the data. However, too many epochs might lead to overfitting since the model starts to memorize specific training data points instead of capturing generalizable patterns. In addition, in complex networks too many epochs might lead to very small or very large gradients, ruining the optimization process. As we can see in the left graph, after the 80th epoch there is fluctuation in the validation loss where it even increases by a bit. Still, it's not obvious if there are too many epochs since the validation loss stays very close to the minimal one in the graph.

Question 3:

To the previous model, add a batch norm after each hidden layer. Compare the results of the regular and modified model as before. How does the batch normalization layer contribute the training? Explain.



Epoch 1
Train Acc: Train Acc: 0.612, Val Acc: 0.795, Test Acc: 0.793
Train Loss: 1.660, Val Loss: 0.757, Test Loss: 0.763
Epoch 10
Train Acc: 0.848, Val Acc: 0.940, Test Acc: 0.938
Train Loss: 0.377, Val Loss: 0.204, Test Loss: 0.209
Epoch 20
Train Acc: 0.865, Val Acc: 0.949, Test Acc: 0.946
Train Loss: 0.336, Val Loss: 0.178, Test Loss: 0.186
Epoch 50
Train Acc: 0.881, Val Acc: 0.942, Test Acc: 0.941
Train Loss: 0.292, Val Loss: 0.170, Test Loss: 0.175
Epoch 100

```
Train Acc: 0.893, Val Acc: 0.947, Test Acc: 0.945
Train Loss: 0.260, Val Loss: 0.155, Test Loss: 0.158
```

From the results above, adding a normalization layer to the model has increased the final accuracy by about 2.5% and decreased the loss by about 5%. In addition, it appears that that the convergence to the minimum of the loss function was a lot faster since the loss value was a lot smaller at each epoch in the training process. Even at the first epoch, the validation accuracy was 0.793 instead of 0.375, where similar validation accuracy at the first model (no batch normalization) wad achieved only in 50th epoch. The batch normalization normalizes the activations of each hidden layer across the mini batch during training, which helps in reducing the internal covariate shift by stabilizing the distribution of inputs to each layer. This allows for more stable and faster training, as the optimization algorithm can converge more efficiently. In addition, it seems that the batch normalization also resulted in some fluctuations in the validation loss across epochs, as seen in the left graph. This might be because the batch normalization works by adding noise to the activations of each layer, which helps prevent the model from overfitting by introducing a slight randomness during training. This can sometimes lead to fluctuations in the training process, as the noise introduced by batch normalization can cause the loss to vary from epoch to epoch.

Question 4:

Train the previous network with batch sizes of 1, 16, 128, 1024 with number of epochs 1, 10, 50, 50 accordingly. How did this choice affect the accuracy, convergence speed (number of epochs), and stability? i.e. how “noisy” is the loss over the epochs. Explain Your answers.

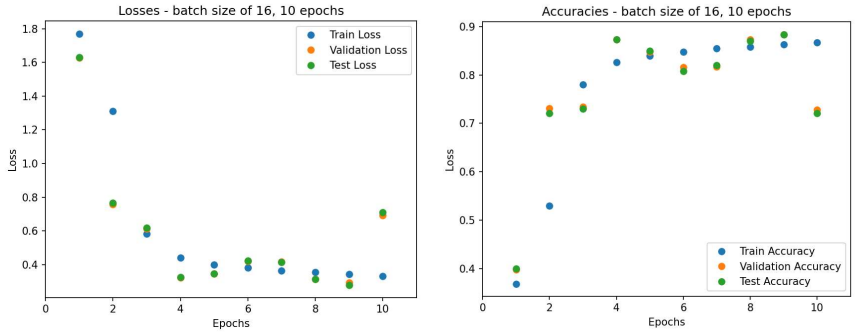
Batch Size: 1, epochs num: 1

```
Train Acc: 0.622, Val Acc: 0.686, Test Acc: 0.678
Train Loss: 1.035, Val Loss: 0.733, Test Loss: 0.734
```

This (batch size=1, epoch num=1) combination has a test accuracy of 0.678. The accuracy is relatively low compared to other combinations, indicating that training with a batch size of 1 and with only one epoch may not have provided enough information for the model to generalize well. In addition, the training time took about 5 minutes, which is relatively long for one epoch, due to the small batch size, as each parameter update requires forward and backward passes for each individual data point in the dataset.

Batch Size: 16, epoch num: 10

```
Train Acc: 0.867, Val Acc: 0.728, Test Acc: 0.720
Train Loss: 0.332, Val Loss: 0.693, Test Loss: 0.710
```

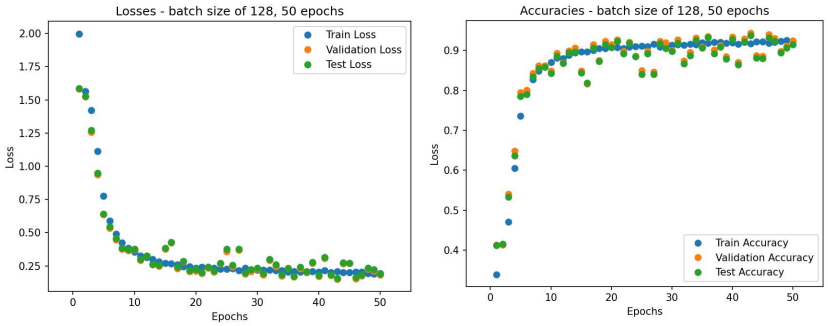


This (batch size=16, epoch num=10) combination has a test accuracy of 0.728. The accuracy improved compared to the previous combination, suggesting that a larger batch size and more epochs allows the model to learn better representations from the data. In addition, it appears that the loss did not converge in those 10 epochs since the losses keep decreasing while the accuracy keeps increasing. Moreover, it is hard to determine if the graph is noisy

since the number of epochs is too small. While it does look noisy, it may be helpful to train on more epochs to get a better overview of the loss over time.

Batch Size: 128, epoch num: 50

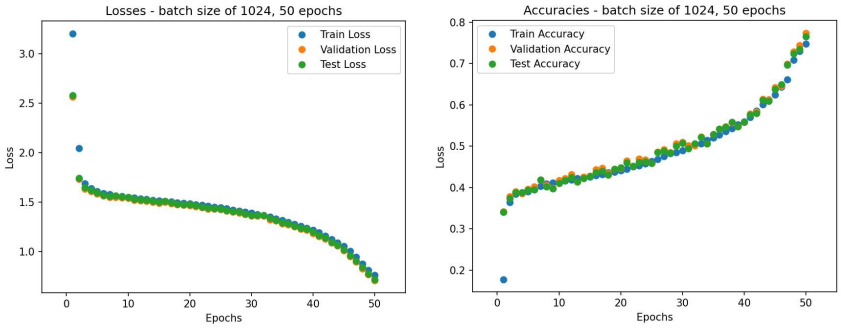
```
Train Acc: 0.923, Val Acc: 0.924, Test Acc: 0.914
Train Loss: 0.195, Val Loss: 0.182, Test Loss: 0.191
```



This (batch size=128, epoch num=50) combination has the highest test accuracy of 0.914, indicating that using a larger batch size and more epochs allowed the model to learn more effectively from the data. In addition, it seems that after 20 epochs the loss and the accuracy graphs start to converge. In addition, there is some noise in the data which can be seen as fluctuating points withing both graphs, but it appears overall that the loss over most of epochs (since about the 20th epoch) stays relatively stable at about 0.3 and that the accuracy stays relatively stable at about 0.9.

Batch Size: 1024, epoch num: 50

```
Train Acc: 0.748, Val Acc: 0.774, Test Acc: 0.766
Train Loss: 0.763, Val Loss: 0.711, Test Loss: 0.718
```



This (batch size=1024, epoch num=50) combination has a test accuracy of 0.774. The accuracy is lower compared to the previous combination, suggesting that using a very large batch size might not be as effective for this task. The training time is the shortest (3 minutes) among all combinations due to the large batch size, but the trade-off is a decrease in accuracy since the loss over epochs shows a steady decrease, but does not seem to converge, indicating that more epochs are needed for further optimization. In addition, the graph looks very steady, which can be attributed to the batch-size, which samples a bigger portion of the train data, making a more stable representation of the data which cause less fluctuations between each representation.

Part 2: Evaluating MLPs Performance

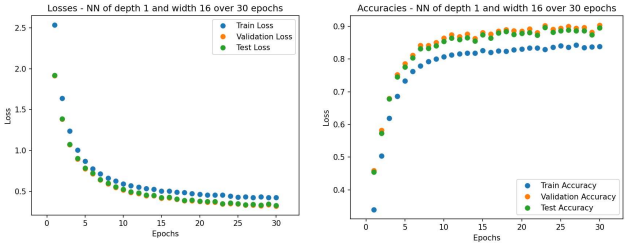
Train 8 classifiers for the following combinations of depth and width.

Depth	1	2	6	10	6	6	6
Width	16	16	16	16	8	32	64

Results: In addition to the previous part of the report, I chose these hyperparameters for this task: learning rate of 0.001, a layer of batch-normalization, a batch-size of 256.

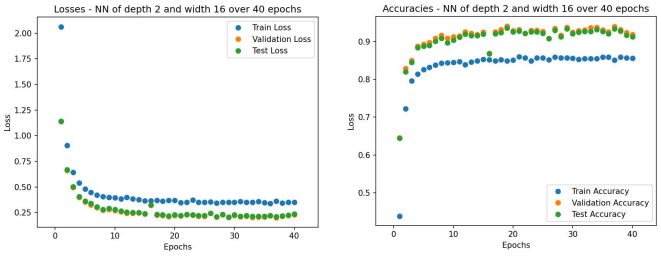
1. Depth: 1, Width: 16, Epoch num chosen: 30

```
Train Acc: 0.847, Val Acc: 0.925, Test Acc: 0.916
Train Loss: 0.390, Val Loss: 0.263, Test Loss: 0.265
```



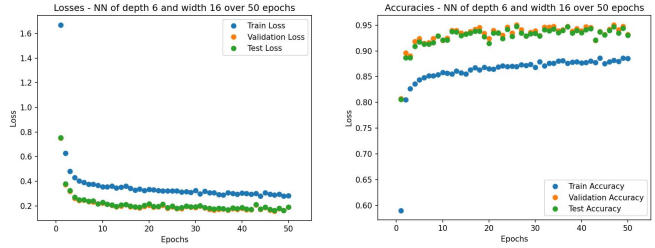
2. Depth: 2, Width: 16, Epoch num chosen: 40

```
Train Acc: 0.854, Val Acc: 0.941, Test Acc: 0.938
Train Loss: 0.357, Val Loss: 0.198, Test Loss: 0.198
```



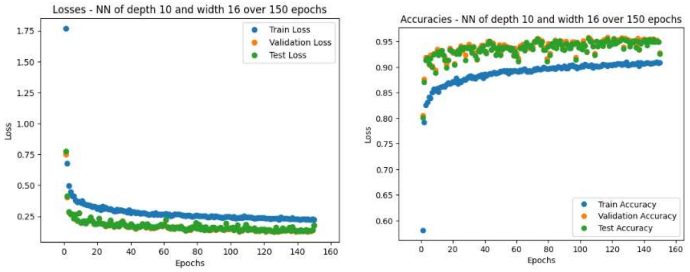
3. Depth: 6, Width: 16, Epoch num chosen: 50

```
Train Acc: 0.885, Val Acc: 0.932, Test Acc: 0.931
Train Loss: 0.282, Val Loss: 0.191, Test Loss: 0.189
```



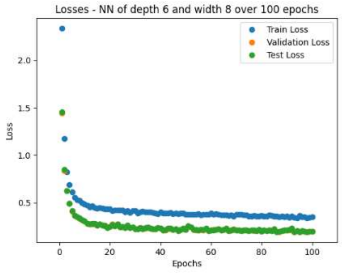
4. Depth: 10, Width: 16, Epoch num chosen: 150

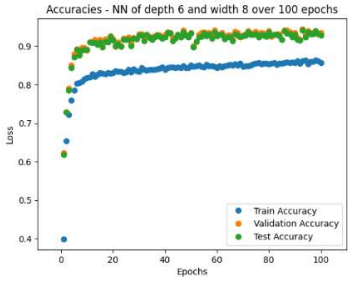
```
Train Acc: 0.910, Val Acc: 0.953, Test Acc: 0.951
Train Loss: 0.224, Val Loss: 0.130, Test Loss: 0.135
```



5. **Depth: 6, Width: 8, Epoch num chosen: 100**

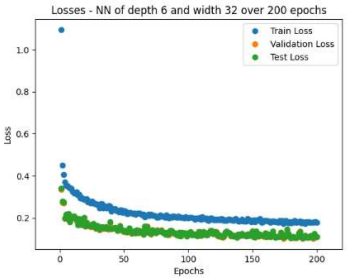
```
Train Acc: 0.857, Val Acc: 0.935, Test Acc: 0.929
Train Loss: 0.353, Val Loss: 0.198, Test Loss: 0.200
```

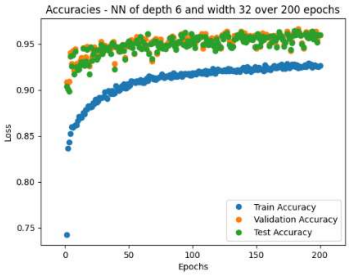




6. **Depth: 6, Width: 32, Epoch num chosen: 200**

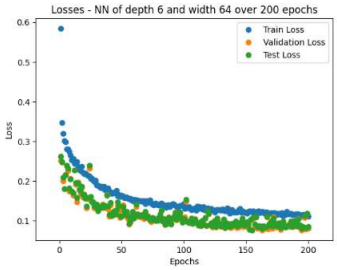
```
Epoch 199, Train Acc: 0.927, Val Acc: 0.960, Test Acc: 0.959
Epoch 199, Train Loss: 0.178, Val Loss: 0.110, Test Loss: 0.111
```

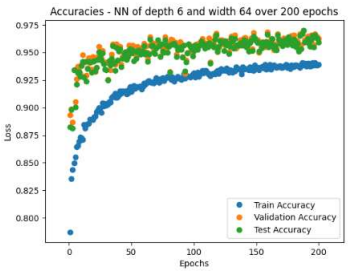




7. **Depth: 6, Width: 64, Epoch num chosen: 200**

```
Train Acc: 0.939, Val Acc: 0.963, Test Acc: 0.959
Epoch 199, Train Loss: 0.145, Val Loss: 0.094, Test Loss: 0.099
```



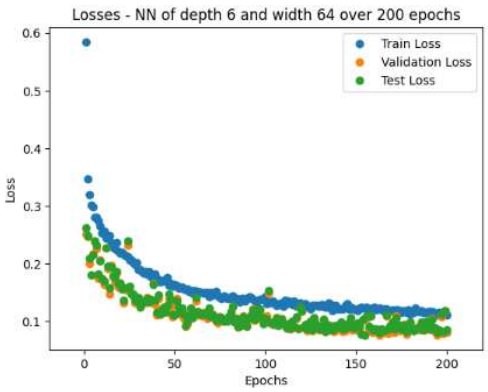


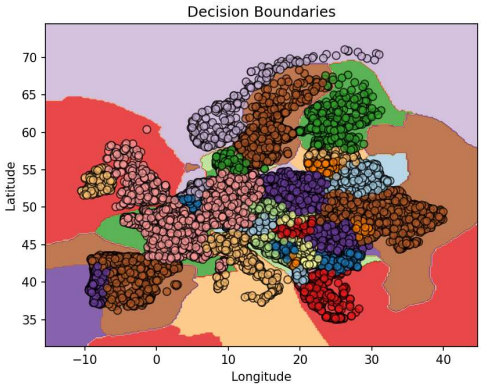
Best model according to validation accuracy is NN of depth 6 and width 64 over 200 epochs. This can be contributed to the large number of epochs, since the other model's accuracies and losses are very similar, using less epochs. This indicates that using a larger number of epochs on the other models may have resulted in having the best accuracy. Moreover, it should be noted that a smaller model would fit this problem better, since there are only two features, and the decision boundary of the data set is mostly linear as we have seen in the previous exercises.

<u>Question 1:</u>

Choose the model with the best validation accuracy. Plot its training, validation, and test losses through the training iterations. Using the visualization helper, plot the test prediction space of the model. Did this model generalize well? Explain.

Best model is the NN of depth 6 and width 64 which got a validation accuracy of 0.963.



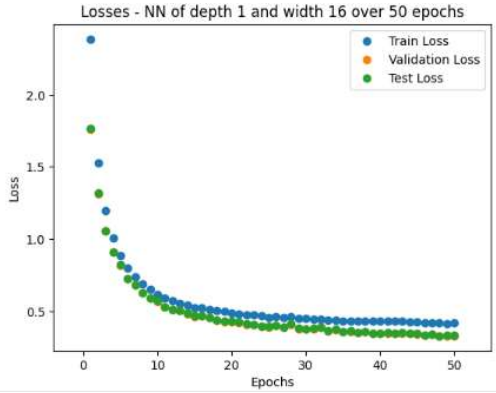


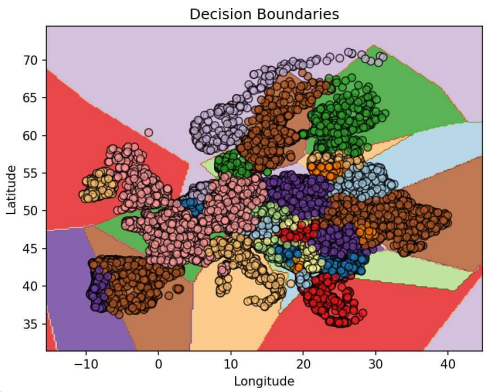
This model did generalize well since as we can see from the graph, the training accuracy is noticeably lower than validation and test accuracies. This gap in accuracies and losses may be attributed to the batch normalization, which helps in reducing overfitting by normalizing the activations of each layer. This can prevent the model from becoming too sensitive to specific weights during training and allows the model to generalize better to unseen data, resulting in better performance on the validation and test sets compared to the training set.

Question 2:

Do the same for the model with the worst validation accuracy. Did this model over-fit or under-fit the dataset? Explain.

Worse model is the NN of depth 1 and width 16 which got a validation accuracy of 0.925.

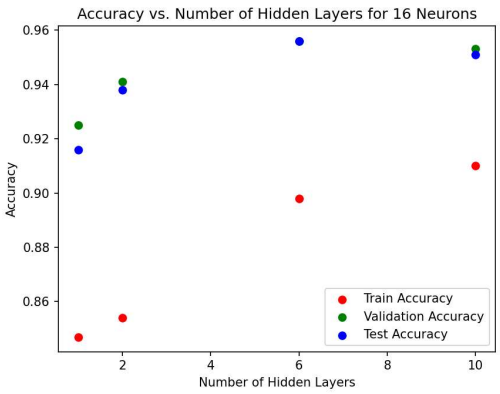




The fact that the losses across the train, validation, and test sets are very close suggests that the model is not overfitting to the training data. In an overfitting scenario we typically expect to see much lower losses on the training set compared to the validation and test sets, indicating that the model is memorizing the training data rather than generalizing well. In addition, the model achieves a relatively high accuracy of 0.925 on the validation set. This indicates that the model can capture some of the underlying patterns in the data, allowing it to generalize reasonably well. However, the accuracy could be higher, suggesting that the model may not be capturing all complexities present in the data and might be underfitting.

Question 3: Depth of Network

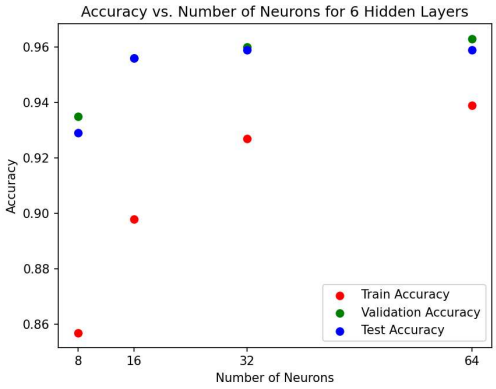
Using only MLPs of width 16, plot the accuracies of the models vs. number of hidden layers. How does the number of layers affect the MLP? What are the advantages and disadvantages of increasing the number of layers in a neural network? Explain.



As we can see from the graph, the more layers result in better train accuracy, while test and validation accuracies were highest in the network of depth 6.[1] On one hand, more layers provide the model with greater ability to represent complex functions. Thus, deeper networks can learn more complex representations of the data. However, Deeper networks require more parameters and computational resources to train and evaluate, leading to higher computational complexity and longer training times. In addition, training deep networks can be more challenging due to issues such as vanishing/exploding gradients, optimization difficulties, and longer convergence times. Also, deep networks are more prone to overfitting, especially when the training data is limited or noisy. Increasing the number of layers without adequate regularization can exacerbate overfitting, leading to poor generalization performance on unseen data.

Question 4:

Using only the MLPs of depth 6, plot the accuracies of the models vs. number of neurons in each hidden layer. How does the number of neurons affect the MLP? Explain.



As we can see from the graph, the more neurons result in better train accuracy.[1] A larger number of neurons provide the model with more degrees of freedom to represent complex functions so the model can potentially capture complicated and nonlinear relationships in the data, leading to better performance in tasks requiring high ability of feature representation. However, increasing the number of neurons also increases the risk of overfitting, especially if the model becomes too complex for the size of the training dataset.

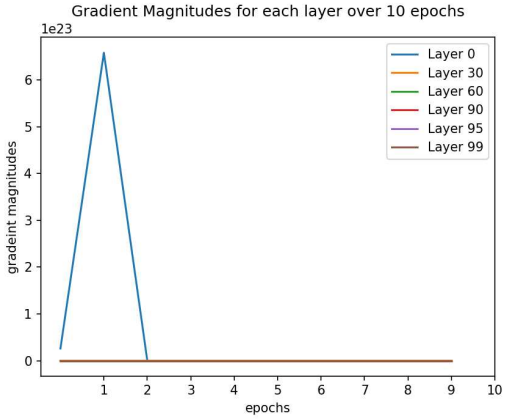
[1] The results can be contributed to the large number of epochs I chose for deeper or wider networks. since the other model's accuracies and losses are very similar, using less epochs. This indicates that using a larger number of epochs on the other models may have resulted in having the best accuracy. Moreover, it should be noted that a smaller model would fit this problem better, since there are only two features, and the decision boundary of the data set is mostly linear as we have seen in the previous exercises.

Therefore, it's essential to balance the model's capacity with the complexity of the dataset and apply different techniques if necessary to prevent overfitting.

Question 5 – Gradient magnitude:

Train another model with 100 hidden layers each with 4 neurons for 10 epochs. This time, keep the magnitude of the gradients for each layer, through the training steps.

I used learning rate of 0.001, a batch size of 256 and batch normalization within layers. Average gradients magnitude through the training epochs for layers 0, 30, 60, 90, 95, 99:



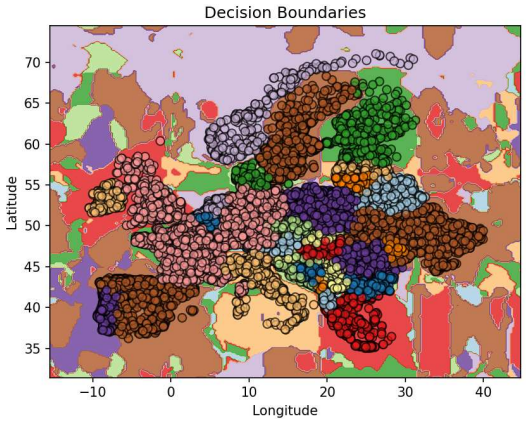
Epoch 1:
Train Acc: 0.067, Val Acc: 0.106, Test Acc: 0.106
Train Loss: 3.443, Val Loss: 3.294, Test Loss: 3.295
Epoch 10:
Train Acc: 0.123, Val Acc: 0.099, Test Acc: 0.099
Train Loss: 3.186, Val Loss: 3.264, Test Loss: 3.269

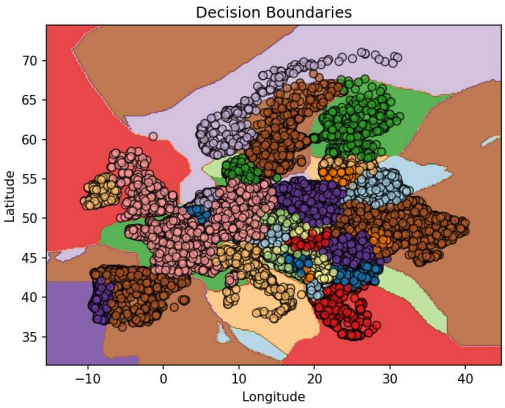
It appears in the graph that all layers except the 0th one has gradient magnitudes very close to zero. In addition, the train, validation and test accuracies and losses did not get better during training and some even got worse. This indicates a problem of vanishing gradients, which occurs when the gradients become extremely small as they propagate backward through the network during training. This can hinder the learning process, especially in deep networks, as the updates to the weights become negligible, leading to stagnation.

Question 7 – Implicit representation (Bonus):

When comparing the performance of a neural network trained on top of the implicit representations to a similar architecture trained on the standard form of the data (a NN of depth 6 and width 16), we observe that the model with the implicit representation preprocessing achieves a higher validation accuracy (0.977) compared to the model trained on the standard form of the data (0.932), after 50 iterations. The implicit representation preprocessing encoded sequential patterns present in the data using sine functions with different frequencies. These encoded patterns served as periodic fluctuations in spatial relationships in the geographical data, which introduce a certain level of smoothness and regularity to the data representation. This helps reduce the complexity of the problem for the neural network, as it no longer needs to discover these patterns from raw, unstructured data. In contrast, the model without this preprocessing step had to rely solely on the raw, unstructured input data to learn these sequential patterns which mase the learning more difficult. Also, the model with implicit representation preprocessing exhibits more structured decision boundaries that align with the underlying patterns encoded by the sine functions.

On the other hand, the model trained on the standard form of the data has decision boundaries that are more complex and less interpretable.





Part 3: Convolutional Neural Networks

Results:

Model	**Test Accuracy**
XGBoost	0.735
Basic CNN - Resnet18 not pre-trained	0.54
Resnet18 Pretrained on Image-Net - optimization over the last layer only	0.6575
Linear probing based on SKlearn (**Bonus**)	0.5075
Resnet18 Pretrained on Image-Net - optimization over the entire NN (fine tuning)	0.7750

Question 1:

What are your two best models and your worst one in terms of test accuracy? What learning rates did you use for the PyTorch CNN baselines? Explain why these are the trends you see.

The two best models in terms of test accuracy are the XGBoost model achieving a test accuracy of 0.735 and the pre-trained fine-tuned model achieving a test accuracy of 0.775, both after one epoch. The two worst models where the basic CNN Resnet18 model which was not pre-trained, with a test accuracy of 0.54 after one epoch, and the logistic regression model that was trained on the features extracted from the ResNet18 pre-trained model, which achieved a test accuracy of 0.5075.

Three different learning rates were experimented with for the PyTorch CNN baselines: 0.01, 0.001, and 0.0001. The initial choice for the learning rate was 0.001, which yielded the best results in terms of accuracy. Lowering the learning rate to 0.0001 slowed down the learning process excessively without providing improvements in accuracy. Increasing the learning rate to 0.01 also did not improve accuracy, indicating that it was too large and led to overshooting during training. Thus, a learning rate of 0.001 was found to be the most effective for the given CNN architectures, indicating that it provided a good balance between training speed and convergence to an optimal solution.

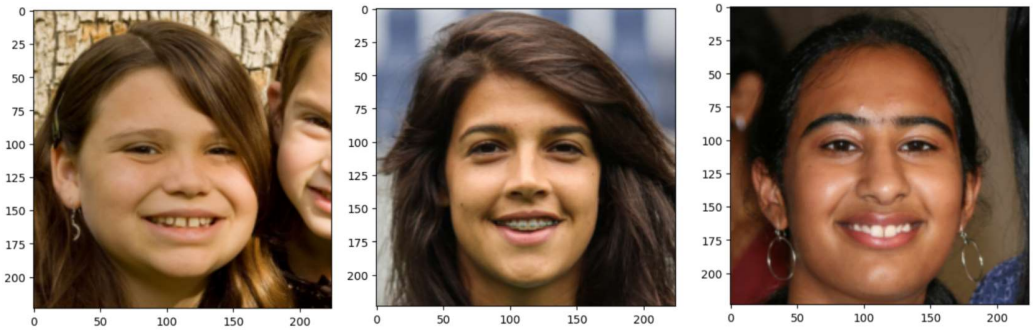
The trends observed can be explained by the reason that models that such as the fine-tuned ResNet18 which was pre-trained on ImageNet dataset, have an advantage in learning meaningful representations of features, which can speed convergence and improve accuracy even after just one epoch. Pre-training allows models to initialize with weights that have already learned general features from a large dataset, enabling them to focus on task-specific features during training. In contrast, models like the basic CNN which was not pre-trained may require many more training examples, longer training or more careful initialization to learn meaningful representations. In addition, Models like XGBoost typically use gradient boosting, which may facilitate faster convergence compared to stochastic gradient descent used in deep learning models like ResNet18.

Question 2:

Visualize 5 samples correctly classified by your best baseline but misclassified by your worst-performing baselines.

5 Samples correctly classified by the best baseline, the pre-trained fine-tuned Resnet18, and misclassified by the worst baseline, the basic Resnet18 not pre-trained:

Real Images:



Fake Images: