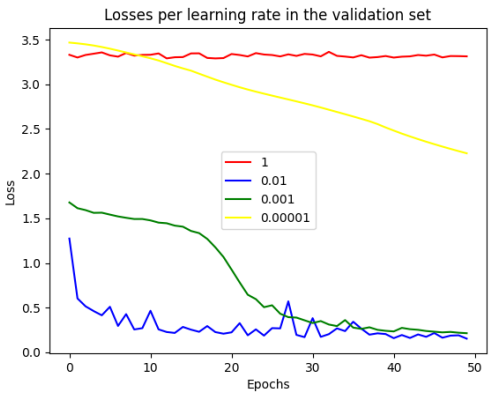
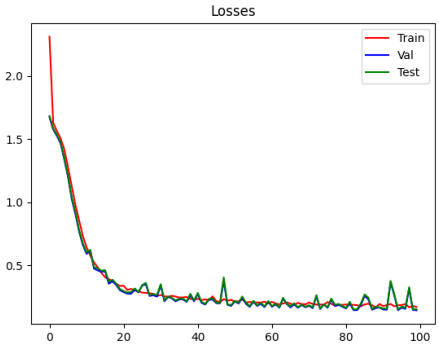
# IML Project 4 Report - Nir Ellor

### 6.1.2.1:




The best learning rate is 0.001, as it steadily reduces the validation loss and achieves the lowest final loss, ensuring effective and stable training. A moderately high learning rate of 0.01 initially reduces the loss quickly but shows instability due to significant fluctuations. The smallest learning rate of 0.00001 converges very slowly, with minimal progress over 50 epochs, indicating inefficient training. In contrast, a high learning rate of 1 fails to reduce the loss at all, as the updates are too large, preventing convergence. This analysis highlights that a too high learning rate can lead to divergence or erratic behavior, while a too low learning rate results in extremely slow convergence, underscoring the importance of proper learning rate selection for efficient and stable training.

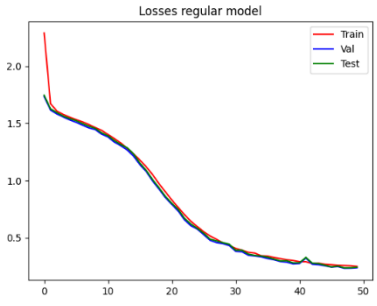
### 6.1.2.2:

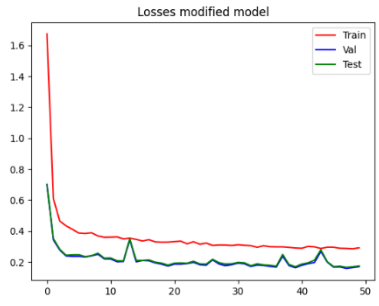



The train loss stabilizes around 0.25 by ~60 epochs. While validation and test losses stabilize between 0.4 to 0.25 with many fluctuations, indicating slight underfitting. With too many epochs, the model risks overfitting to the training dataset, results in low training loss but high validation /test loss. With too few epochs, the model suffers from lack of expressiveness, failing to learn important patterns, leading to high losses on both train, validation and test sets.

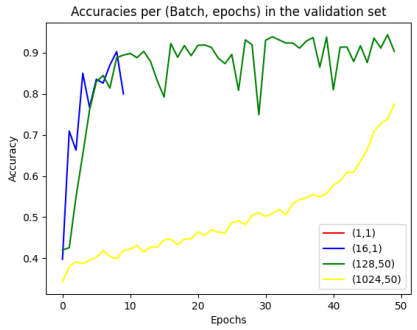
### 6.1.2.3:

The modified model with batch normalization exhibits faster convergence compared to the regular model, as the training loss decreases rapidly in the initial epochs. Additionally, the losses in the modified model stabilize sooner, with some fluctuations in both the validation and testing curves after ~10 epochs. This suggests that batch normalization improves training stability by normalizing inputs to each hidden layer, leading to the modified model achieving lower final losses. Thus, batch normalization accelerates training while reducing overfitting risks.



## 6.1.2.4:




(i): In the plot to the left we can't see the red graph, representing the validation accuracy of 1 batch and 1 epoch since it is only a dot and not a full continuums curve. Therefore, the dot's value is 0.764. With batch size 1 and 1 epoch, the model updates frequently but fails to converge effectively, resulting in low-accuracy outcome. Batch size 16 with 10 epochs shows rapid initial convergence but exhibits high fluctuations due to smaller batches, leading to inconsistent validation accuracy. Batch size 128 with 50 epochs achieves higher accuracy but also suffers from fluctuations as the updates are still not stable. Batch size 1024 and 50 epochs provide smoother convergence, though slower, with fewer fluctuations and consistent improvement in accuracy. It seems that smaller batches lead to faster initial learning but introduce instability, while larger batches ensure more stable and gradual convergence over time, with the cost of slow updates.

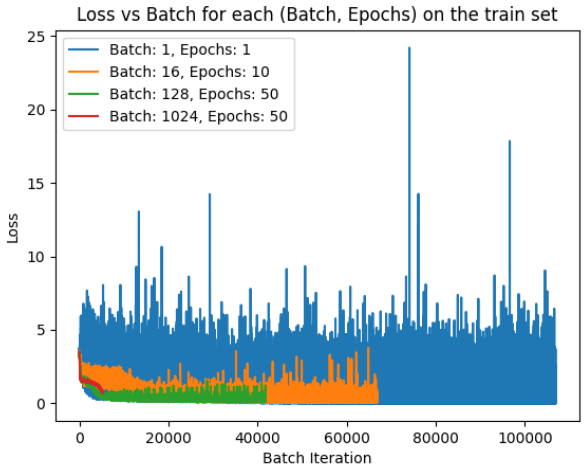
(ii):

**It appears that smaller batches increase the number of iterations, thus decrease speed.**

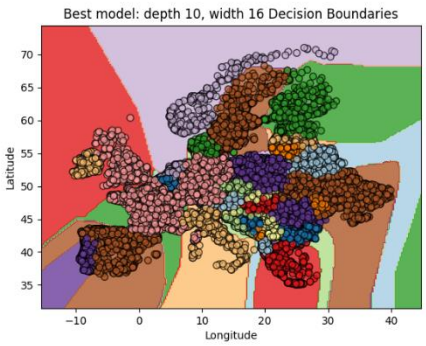
| Batch Size | Epochs | Iterations per Epoch |
|---|---|---|
| 1 | 1 | 106897 |
| 16 | 10 | 6682 |
| 128 | 50 | 836 |
| 1024 | 50 | 105 |

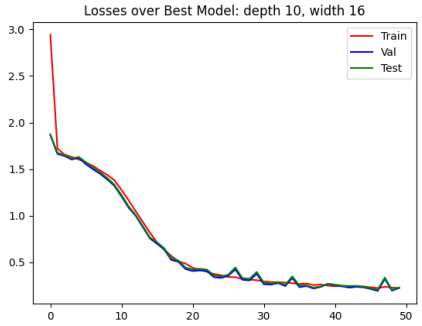
## (iii):

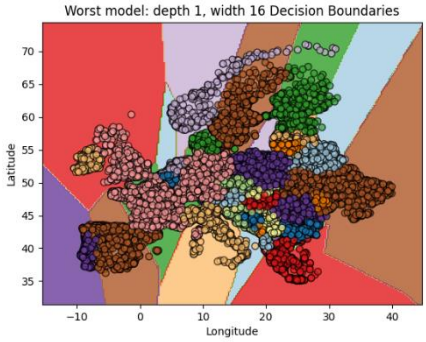
The graph illustrates the impact of batch size and epochson the stability of training loss. With a batch size of 1 and 1 epoch, the loss curve is highly noisy due to frequent updates with minimal data. Increasing the batch size to 16 and 10 epochs reduces noise but still shows noticeable fluctuations. For batch size 128 and 50 epochs, the curve becomes smoother, indicating better stability. The smoothest curve is observed with batch size 1024 and 50 epochs, as larger batches provide more reliable gradient estimates. Overall, larger batch sizes lead to more stable and consistent loss curves, while smaller batches result in noisy training due to higher variance in gradient updates.

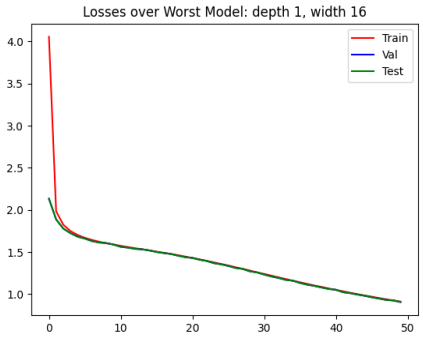


### 6.2.1.1 + 6.1.1.2:



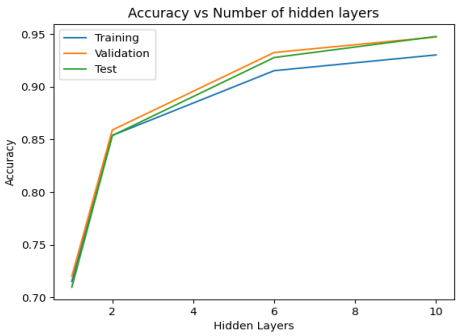

| Model depth | Model weight | Validation Accuract |
|---|---|---|
| 1 | 16 | 0.71997 |
| 2 | 16 | 0.85893 |
| 6 | 16 | 0.93257 |
| 10 | 16 | 0.94731 |
| 6 | 8 | 0.89036 |
| 6 | 32 | 0.93945 |
| 6 | 64 | 0.83446 |

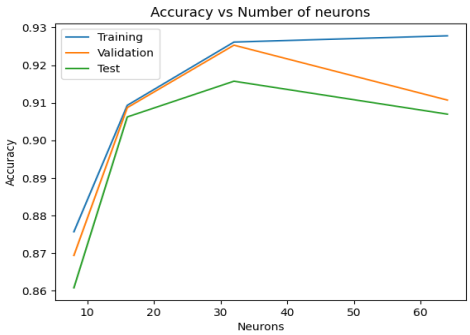
Hence, the best model according to the validation accuracy is [10, 16] with accuracy 0.94731 and the worst model according to the validation accuracy is [1, 16] with accuracy 0.71997.

The best model appears to have generalized well. The train, validation, and test losses closely follow each other throughout the training process, indicating minimal overfitting. Additionally, the decision boundaries in the plot reflect a well-formed and reasonable separation of the classes, showing the model's ability to generalize across different regions.

The worst model underfits the dataset, as indicated by the relatively high losses for all phases (train, validation, and test) that do not approach zero even after many epochs. The decision boundary plot also shows overly simplistic linear regions that fail to capture the complex structure of the data. The lack of fitting capacity is likely due to the shallow depth (1 hidden layer) of the network, which restricts its ability to learn complex patterns.

### 6.2 .1.3 + 6.2.1.4:



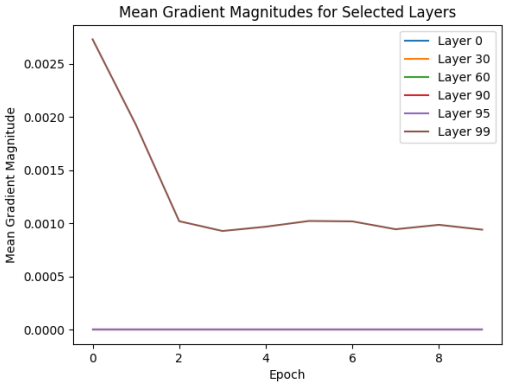
As seen in the left graph, Increasing the number of hidden layers improves model accuracy up to a certain point, as shown by the upward trend in accuracy across training, validation, and test sets. This might happen because more layers allow the model to learn complex patterns better. However, after a certain number of layers, the accuracy doesn't significantly improve.

Increasing the number of layers improves the model's ability to capture complex patterns, leading to higher accuracy for challenging tasks. However, deeper networks may suffer from vanishing gradients, overfitting, and require more data and computational resources.

As seen in the right graph, Increasing the number of neurons initially improves the model's ability to capture patterns, leading to higher accuracy. Beyond a certain point, adding more neurons decreases the accuracy of the model and causes overfitting, as seen by the gap between training and validation/test accuracies.

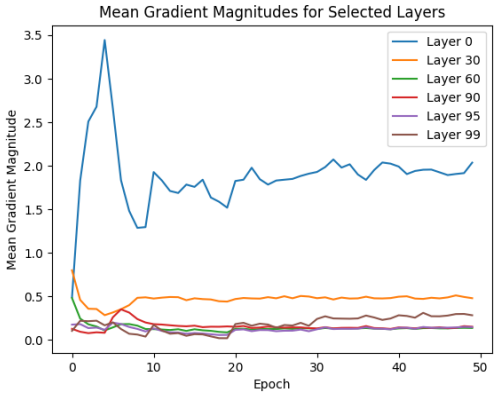
### 6.2.1.5 + 6.2.1.6:

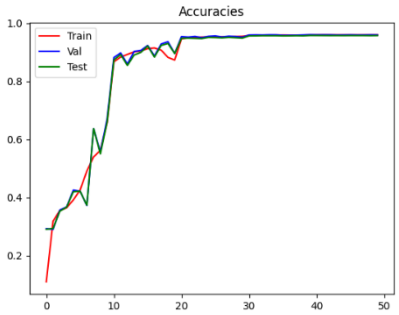
We can see that the gradient magnitude in layer 99 explodes at the first epochs, and in all the other layers the gradient magnitude vanish at every epoch.

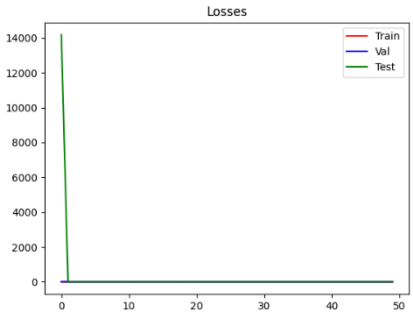



After many iterations, I modified the model in the following way:

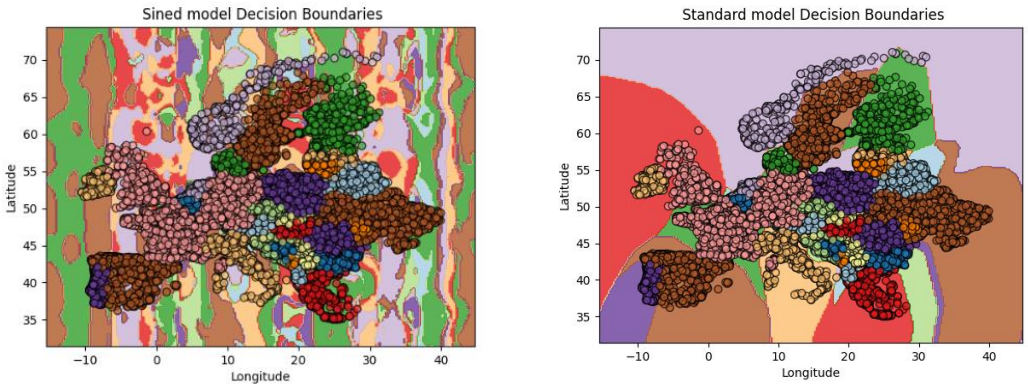
**40 neurons, residual connections, clipping at max norm of 8, learning rate of 0.0005, batch size of 512, 50 epochs, learning rate scheduler and L2 regularization of 1e-4**. The gradient magnitudes neither vanish nor explode, as can be seen in the graph to the left, where at layer 0 the magnitude is high but not exploding and in the other layers the magnitude is between 0.1-0.5. Besides, this model achieves excellent accuracy and loss as can see in the graphs from below**:**



### 6.2.1.7:




The sine model learned complex patterns better by using sine functions to create detailed features. This helped the model capture small differences in the data, unlike the standard model, which created smoother, less accurate boundaries. By using sine transformations, the model focused on more complex relationships directly.

### 7.6.1:

1. XGBoost test accuracy: 0.7350.

2. Training from Scratch:

| Learning Rate | Loss | Validation Accuracy | Test Accuracy |
|---|---|---|---|
| **0.1** | 1.2251 | 0.5500 | 0.5200 |
| **0.01** | 0.8302 | 0.5000 | 0.5700 |
| **0.001** | 0.8006 | 0.5650 | 0.5725 |
| **0.0001** | 0.6828 | 0.5450 | 0.5700 |
| **1e-05** | 0.6859 | 0.5000 | 0.5350 |

3. Linear Probing:

| Learning Rate | Loss | Validation Accuracy | Test Accuracy |
|---|---|---|---|
| **0.1** | 3.4454 | 0.6200 | 0.6100 |
| **0.01** | 0.6329 | 0.7200 | 0.7175 |
| **0.001** | 0.6735 | 0.6550 | 0.6500 |
| **0.0001** | 0.8036 | 0.4500 | 0.5225 |
| **1e-05** | 0.6770 | 0.5050 | 0.5700 |

4. Linear Probing based on sklearn test accuracy: 0.7175

Fine Tuning:

| Learning Rate | Loss | Validation Accuracy | Test Accuracy |
|---|---|---|---|
| **0.1** | 2.1280 | 0.4750 | 0.4975 |
| **0.01** | 0.9058 | 0.5000 | 0.5025 |
| **0.001** | 0.6324 | 0.5700 | 0.6150 |
| **0.0001** | 0.4737 | 0.7750 | 0.7550 |
| **1e-05** | 0.6394 | 0.6750 | 0.7125 |

Hence, the best model according to the test accuracy is Fine-Tuning with learning rate of 0.0001, achieving 75.5% test accuracy, as fine-tuning all layers with a low learning rate allowed better adaptation of pre-trained features. The worst model is Fine-Tuning with learning rate of 0.1, with 49.75% test accuracy, likely due to divergence caused by a high learning rate. The trend shows that lower learning rates perform better in fine-tuning, while higher learning rates tend to cause instability. Overall, the fine-tuning approach receives better test accuracy on average (0.6165) than linear probing (0.614) and training from scratch (0.5535) approaches. Moreover, the best ResNet16 model above achieves higher accuracy than both XGBoost and Linear Probing based on sklearn, as the results appear on previous page.

### 7.6.2:

Here are 5 samples correctly classified by the best baseline, Fine-Tuning with learning rate of 0.0001, but misclassified by the worst-performing baseline, Fine-Tuning with learning rate of 0.1: