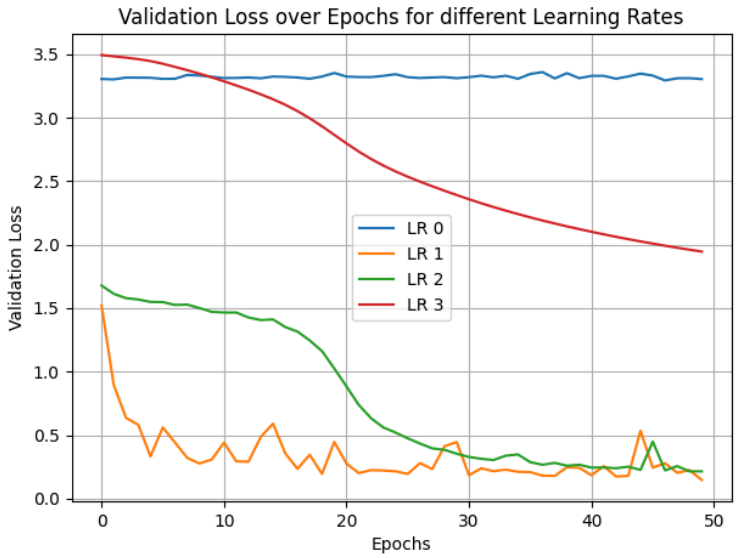
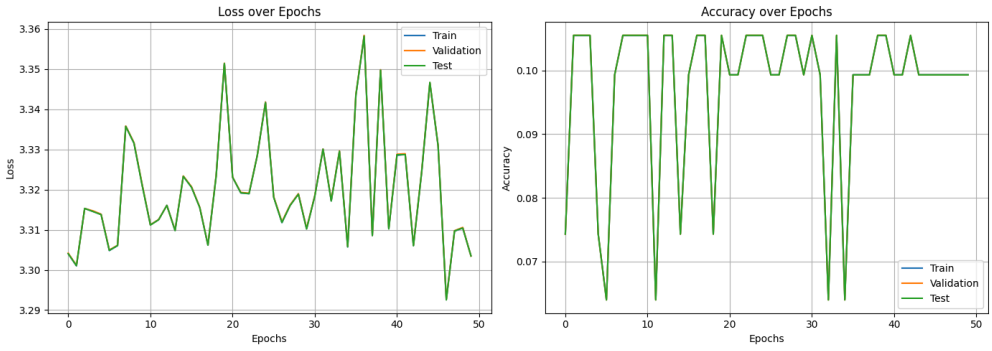
IML Execise 4 - results

6.1.2

1. Without looking at the results a theoretical understanding of the way GD works, is that for an infinitely small eta - it will cancel out the gradients leading to barely any movement in our input values. An infinitely large eta will cause our input values to go toward -inf, regardless of the gradient. Our results show this exactly - for a large eta the results are noise, and do not converge, for the smallest eta the results do not converge fast enough:

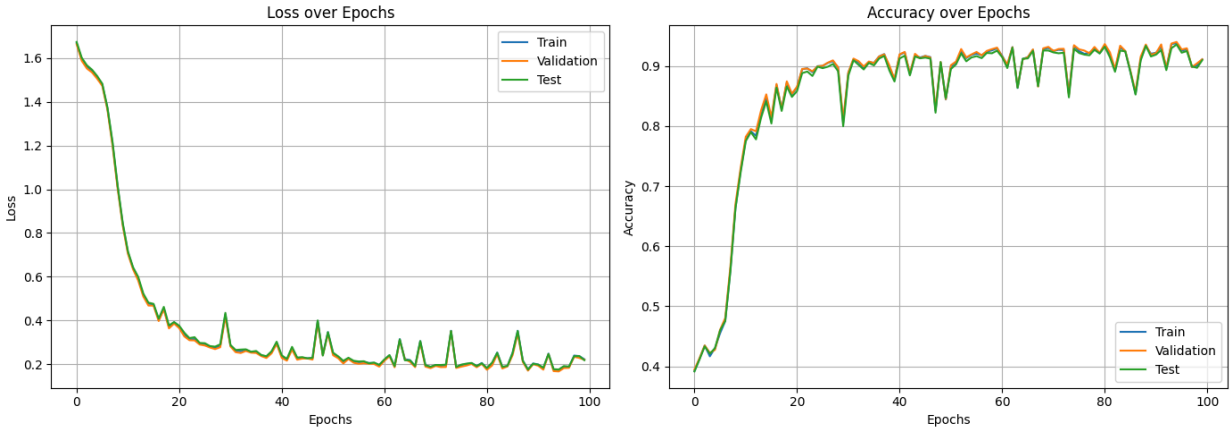


Loss and accuracy for Learning rate = 1.0



2. The epochs play an important role in GD convergence. As for regular GD the number of iterations is what promising convergence up to epsilon of the minimal value of the lost function, with SGD - the convergence is of the expected value, so the more iterations the

better chances of convergence. However, There is the risk of overfitting the training data, this is why mid-traing validation is crucial and not taking the results of the last epoch.

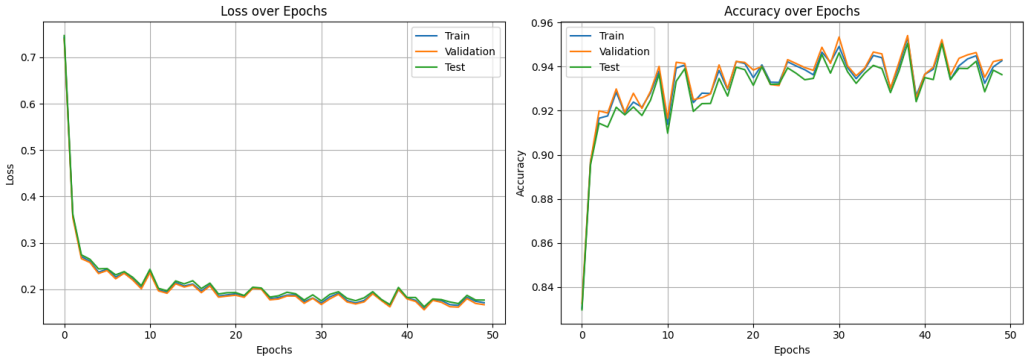


In the results, we mostly see a vanquishing return to the runtime, but a close look will show after the 50th epoch a start of training data improvement, with a very small degradation in validation and test results.

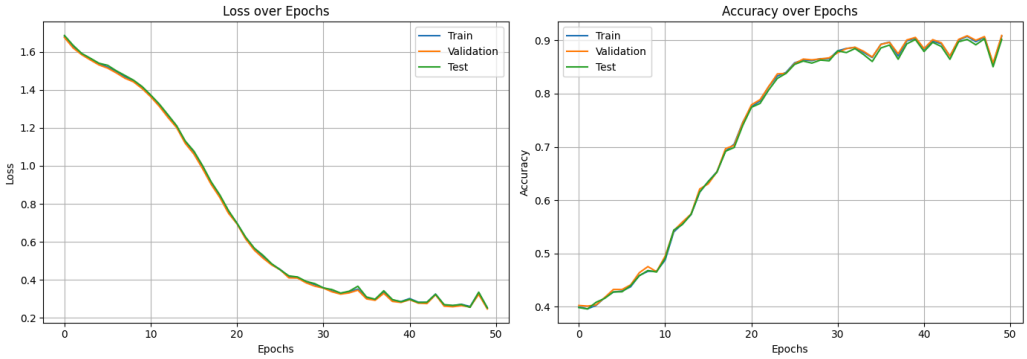
3. BatchNorm normalizes activations within each layer to have mean=0 and variance=1 across the batch. This prevents activations from becoming too large or too small, which would cause vanishing or exploding gradients. By maintaining stable activation magnitudes throughout the network, gradients remain in a reasonable range, allowing for:
    - Higher learning rates without divergence
    - Faster, more stable convergence
    - Reduced sensitivity to weight initialization

    However, this process can introduce some issues as well, due to the fact that it utilizes batch statistics that could be noisy. Our results showed better results without normalization because our network is not as deep and our dataset are relatively small.

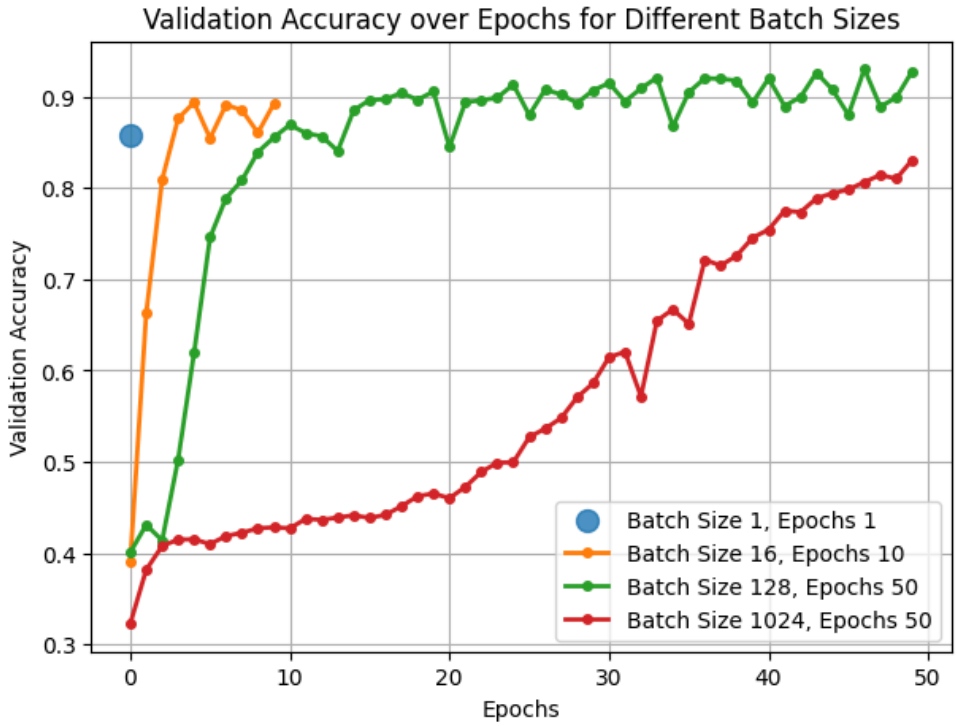
With normalization:



Without Normalizations

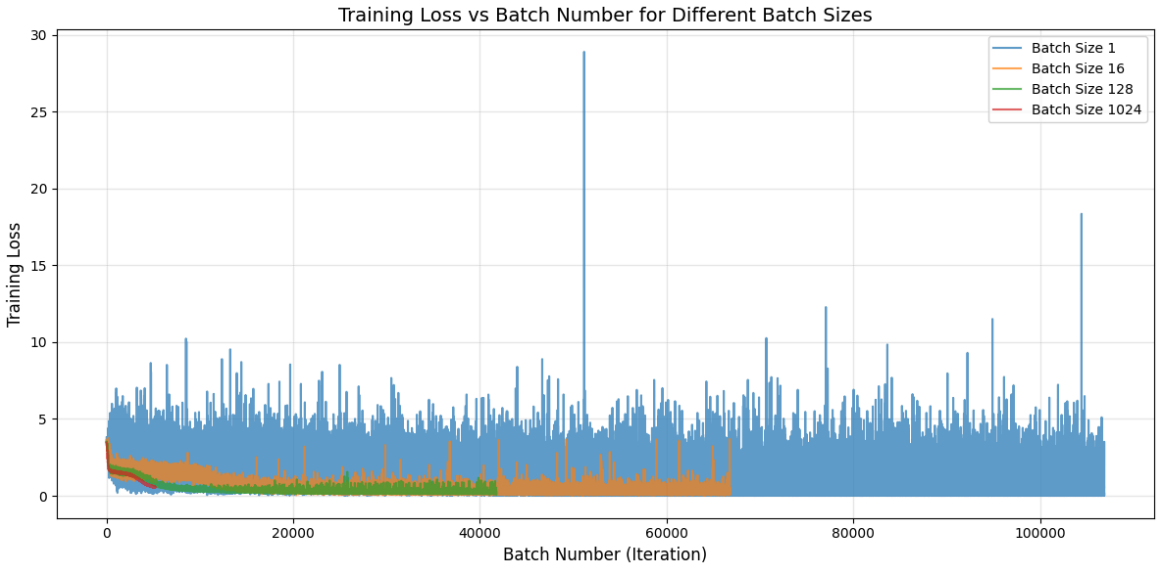


4. Generally the batch size determines how close is each iteration gradient to the real gradient of the loss function. There is a balance between runtime, and SDG accuracy. :
    a. Four different accuracies in relation to validation set.



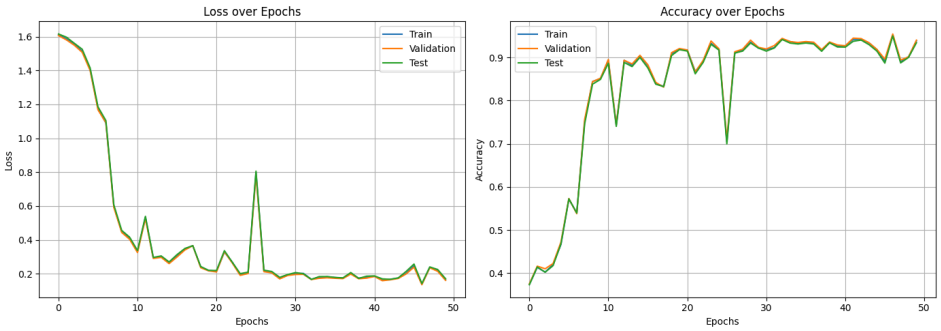
The results are actually very enlightening, as mentioned above I would expect the sharpes convergence with the larger batch size as it is the closest to the true gradient each step. However this is not true for several reasons. The first - it needs more iterations, a very important parameter of any type of GD. The second reason less significant in this case is that the noise made by the inaccuracy of the stochastic gradient is beneficial to escape local minima,

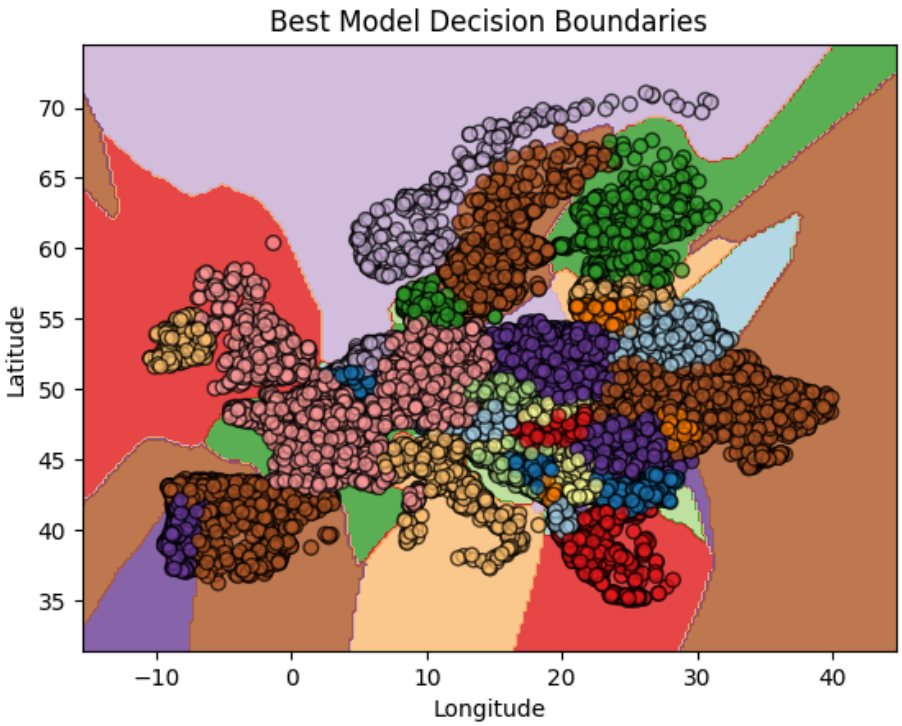
    b. Each iteration amount of batches per epoch :106897, 6681, 835. 104.
    c. This show exactly how noisy the loss is as a factor of the batch size, with a single input batch being extremely noisy due to inaccurate gradients.



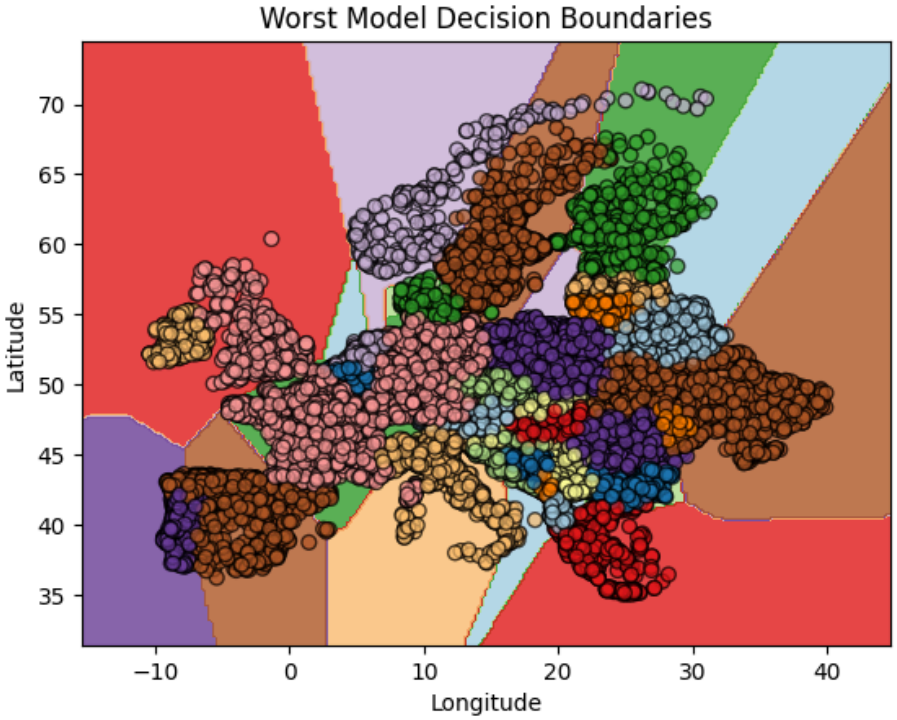
6.1.2

5. The best model generalized very well to the test set. The graph shows clearly that the model generalizes very well from the train set, and keeps a small gap between the train accuracy and test accuracy. In the end, the most expressive model (with the most variables) showed the best results width 64, depth 6.

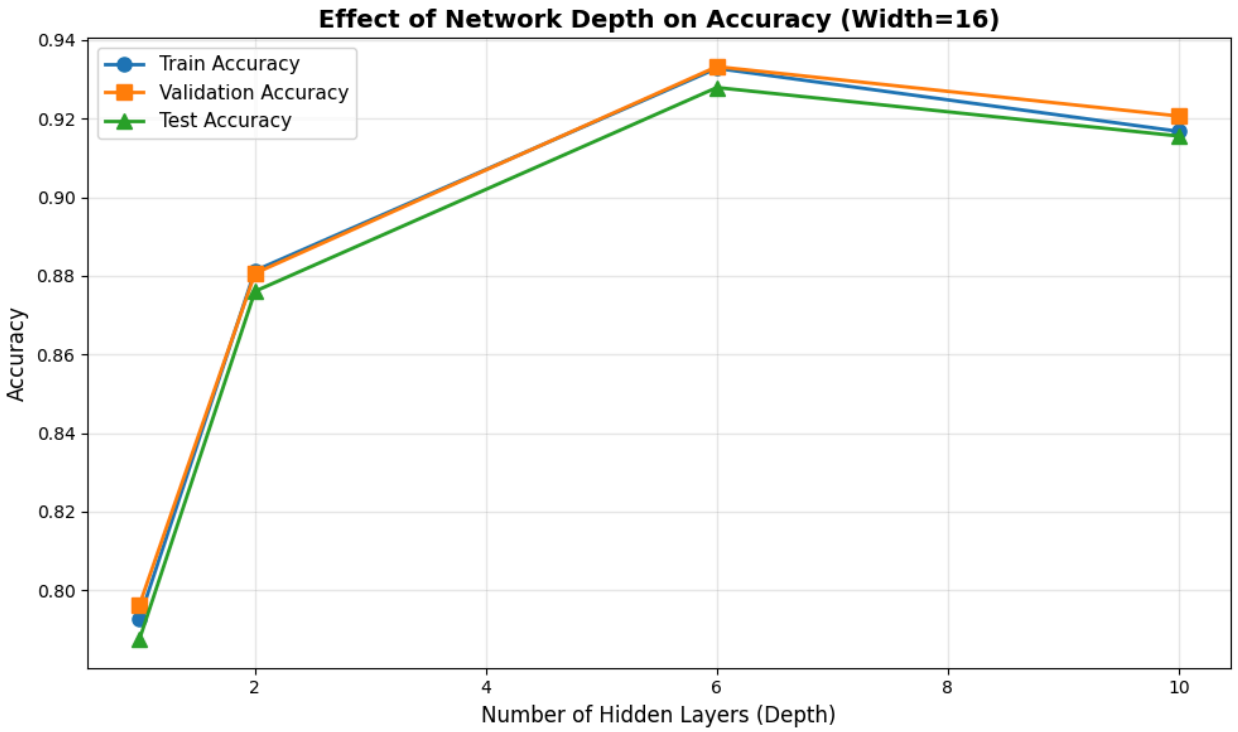




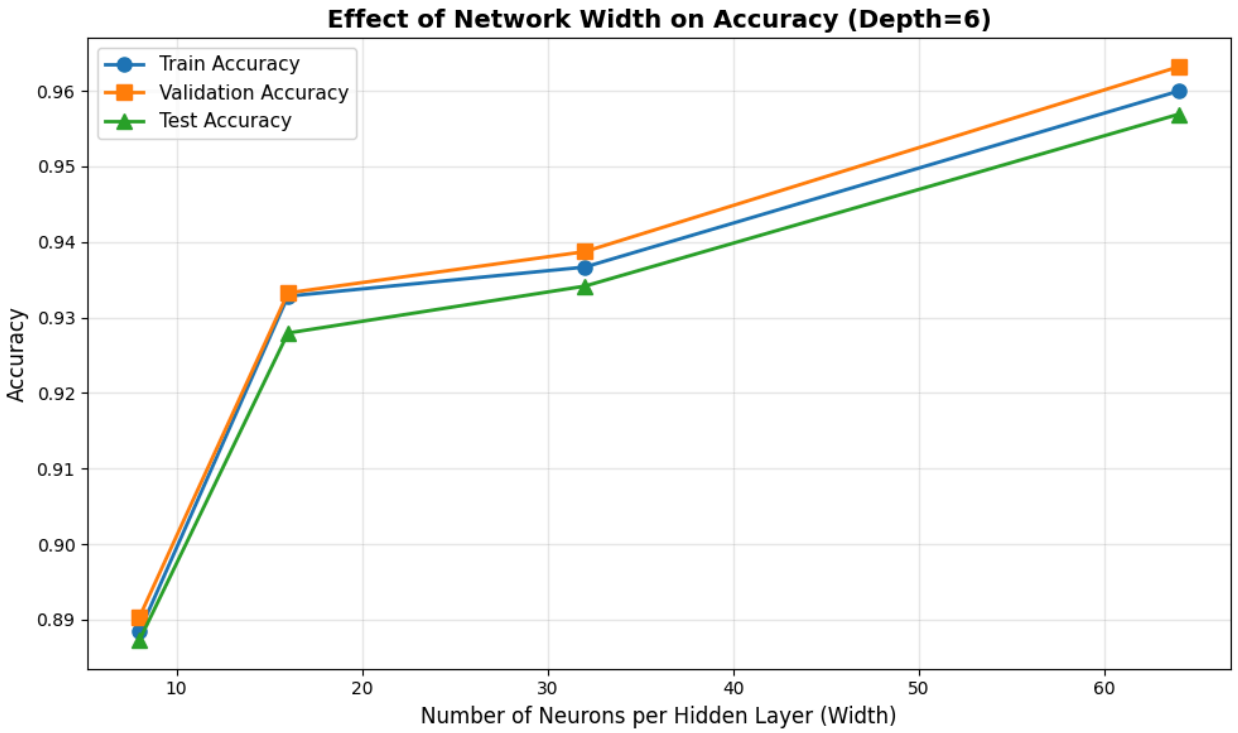
6. The 1,16 was not descriptive enough, 16 variables are not enough to differ 10 countries.



7. The experiment shows a decrease in performance in the 10 layers from the 6 layers. Likely due to exploding/ vanishing gradients.

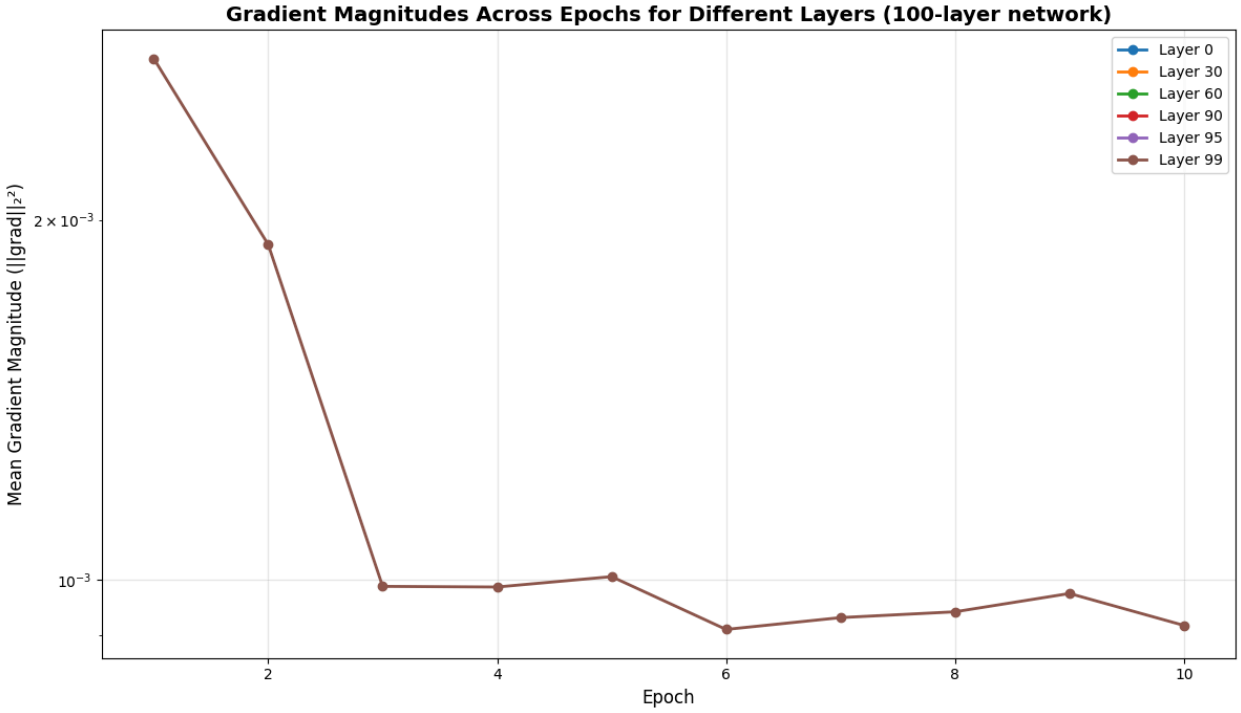


8. Results continue to improve as the width gets wider without opening a generalization gap.



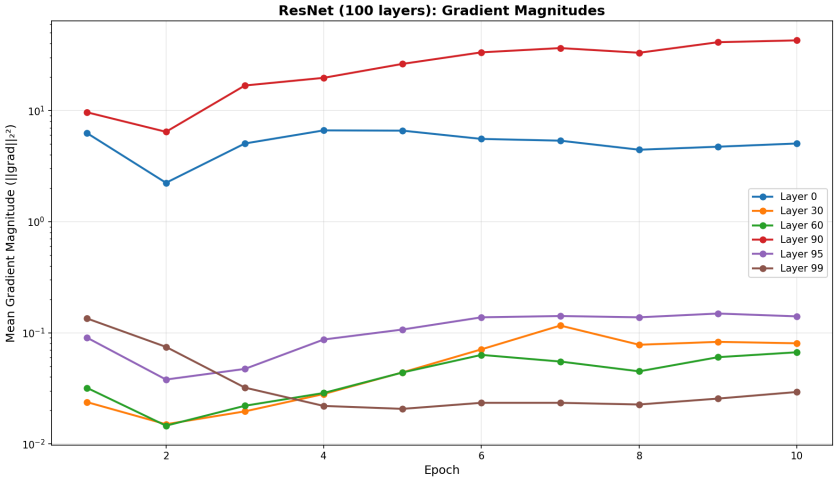
9. What we see in the 100 depth experiment is diminishing gradient on every hidden layer until the 99th. As expected since the gradient of the first layer is a multiple of the other

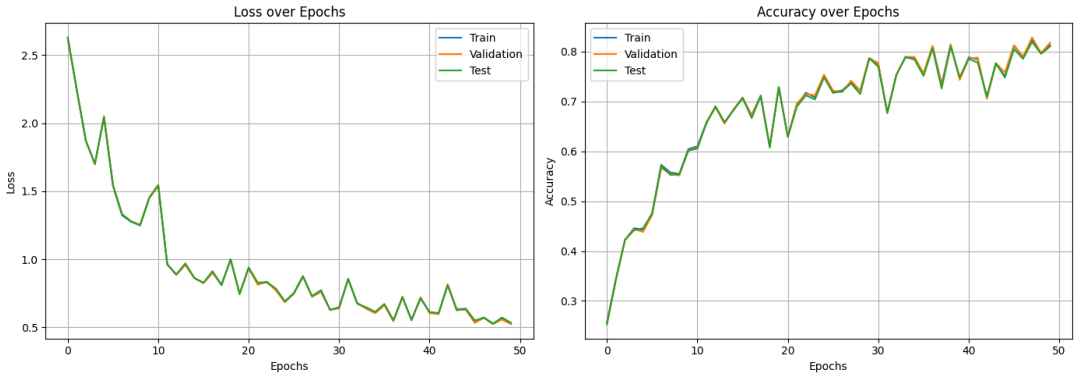
99 layers it leads to very small gradients. The solution, batch normalization and residual layering, and possibly change in activation function to keep some negative values.



Other gradients are 0.

10. I implemented a resnet with batch normalization. This solved the problem while not reaching good enough results. Gradients were kept but probably the insufficient width did not allow the model to be expressive enough to model 10 different countries.





7.6

1. Best and Worst Models:
    a. The Best (Winner): Fine-Tuning. The model starts with a rich understanding of the world (edges, textures, shapes from ImageNet). Fine-tuning allows it to adjust its "mental model" specifically to detect the subtle artifacts of GAN-generated faces vs real camera noise, without having to learn what an "eye" is from scratch. The Runner-Up: Linear Probing. The features extracted by a pre-trained ResNet are incredibly robust. Even though the backbone is frozen, the high-level semantic separation is strong enough that a simple Logistic Regression (the final layer) can find a decent boundary.
    b. The Worst: Training from Scratch. The reason being we simply don't have enough data and the models rather memorize the training set then understanding structure. They have millions of parameters. A possible solution could have been data augmentation or pre-training the model on a similar data rich problem similar to Fine-Tuning. Even training the model for 50 epochs showed little improvement over the 0.75 accuracy bar.
    c. Note regarding XGBoost: While surprisingly decent, XGBoost on raw pixels usually falls behind CNNs because it lacks locational invariance (it doesn't understand that an eye in the top-left is the same object as an eye in the center).
    d. 2. Learning Rates Logic:
        i. Fine-Tuning and linear-probing - I used a lower learning rate: exp(-5) and exp(-4). We rely on the pre-trained weights being 99% correct; and only want to nudge them slightly. A high LR showed seemed to shock the values
        ii. For the other model I used exp(-3) for the first part and exp(-4) for later parts

L: stands for Label. S - scratch_model. F: stands for fine tune.