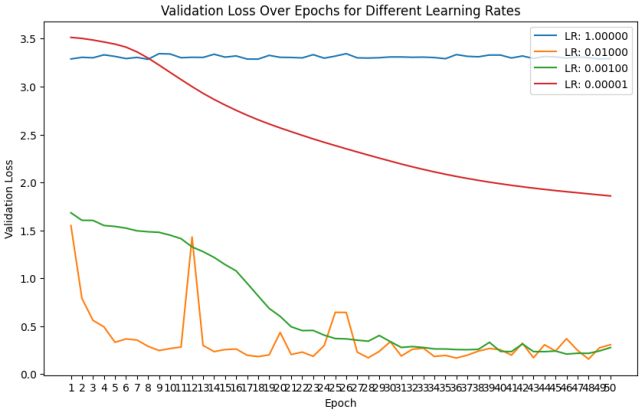
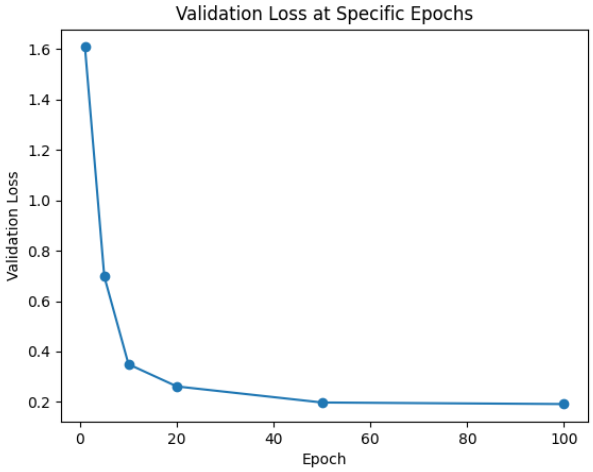
6.1.2

1. We observe that the loss for lr=1 does not decrease, indicating that the learning rate is too high, causing the optimization process to skip the local minimum in the convex space what cause it to spike. Similarly, for lr=0.0001, the loss decreases, but not rapidly enough. This is attributed to the small steps taken during optimization. When lr=0.001, the model achieves the best result with a minimal loss. This is because 0.001 strikes a balance between being large enough to make progress quickly and small enough to avoid overshooting the minimum in the convex space significantly. For lr=0.01, the model starts with a low loss due to the large step size but experiences spikes in the loss because it skips over the minimum and struggles to find it again but it seems that in the end he is getting closer and closer to it.



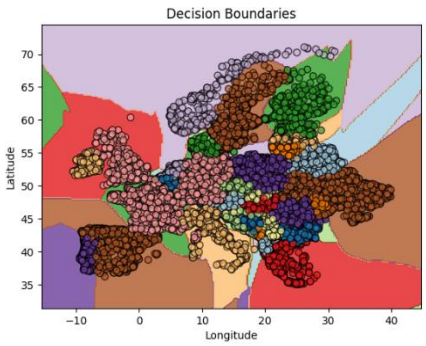
2.

We observe substantial learning in our model between epochs 0-5, leading to a significant decrease in loss. The improvement continues between epochs 5-10. However, beyond this point, our model shows diminishing returns in terms of improvement. This behavior can be attributed to the fact that gradient descent takes larger steps to locate the minimum during the initial epochs. By epoch 20, our model is in close proximity to the minimum, resulting in only marginal improvements thereafter.

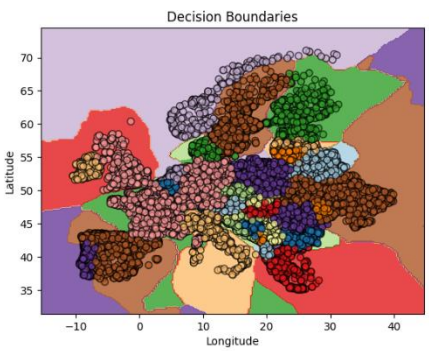


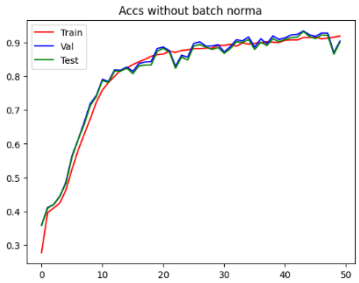
3.

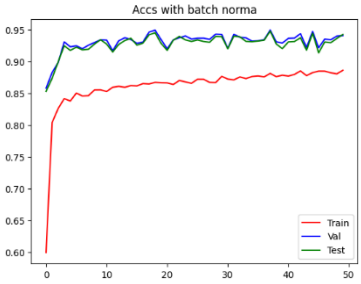
Without batch norma



With batch norma







First, we observe that our model with batch normalization achieved a higher score on the validation set. This improvement may be attributed to the fact that normalization helps mitigate the impact of weights that contribute a significant amount of noise. By normalizing, it aids in training the model more effectively, leading to better generalization and improved performance on the validation data (but still its very odd) and the spikes accour because our model skips the min value .

4.

Batch Size	Epochs	Model Name	Train Accuracy	Test Accuracy	Val Accuracy
1	1	Model_Batch_1_Epochs_1	0.621757392630289	0.6780663024769887	0.6860734864925541
16	10	Model_Batch_16_Epochs_10	0.8671244281878818	0.7204968944099379	0.727680909975305
128	50	Model_Batch_128_Epochs_50	0.9228509686894861	0.9143156476839033	0.9239691685998653
1024	50	Model_Batch_1024_Epochs_50	0.747644929230942	0.7656963256753723	0.7744518446456634

Test accuracy

(1, 1) Test Accuracy is 0.678: This is due to the model's adaptability to individual data points with a small batch size of 1,that mean we are doing gradient decent for each example, capturing fine-grained patterns but potentially being sensitive to noise, this sensitive is what led to small val acc and being worse model .

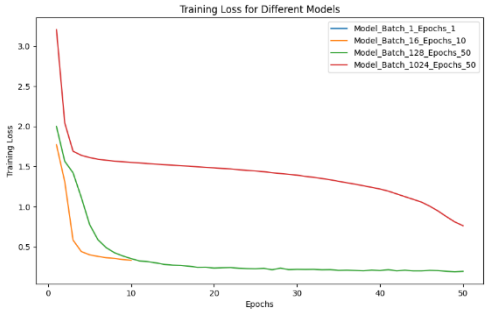
(16, 10) Test Accuracy is 0.72: Achieved through a balanced configuration with a batch size of 16 and 10 epochs, allowing the model to generalize well with improved accuracy compared to (1, 1).

(128, 50) Test Accuracy is 0.914: The larger batch size (128) and extended training (50 epochs) contribute to increased efficiency and better pattern capture, resulting in a higher test accuracy, due to the fact that he isn't take into account every example of, what make him not sensitive to noise, but in the other hand the batch size small enough to be able to find the global min

(1024, 50) Test Accuracy is 0.765: Due to the large batch size, this model struggles to converge to the global minimum. The substantial batch size results in an average loss of 1024. This implies that if our model is situated within a convex region of local minima, it will face challenges in navigating towards the global minimum due to the excessively large average.

Stability:

(1, 1) - Singular Stability: The model with a batch size of 1 and 1 epoch exhibits a single dot on the graph, symbolizing exceptional stability.

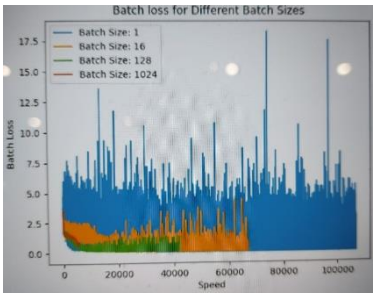


(16, 10) - Moderate Stability with Oscillations: The model with a batch size of 16 and 10 epochs demonstrates moderate stability. Initially, the loss decreases steadily, but after approximately half of the epochs, it introduces oscillations, suggesting intermittent challenges in learning specific patterns. Because of the small batch size ,when batches are with a lot of noise ,small spikes may accrue

(128, 50) - Varied Stability with Small Spikes: The model with a batch size of 128 and 50 epochs displays varied stability. The loss maintains a consistent range for the majority of epochs, with a steady rate of decrease up to epoch 10. However, the presence of small spikes from epoch 10 to 50 indicates occasional challenges or fluctuations in the learning process, due that our model trying to get to the global min, and stepping above it.

(1024, 50) - Initial Instability Followed by Recovery: The model with a batch size of 1024 and 50 epochs exhibits initial instability, noticeable around epoch 4. Despite this, the model manages to recover and achieve stability in later epochs. This appens due the fact the our big batch size give about the same average for each batch,what make it hard from our model to significant learn from the data, so our model still learn but in the same small rate, coming closer to the local min in the convex in about the same size steps.

Speed:

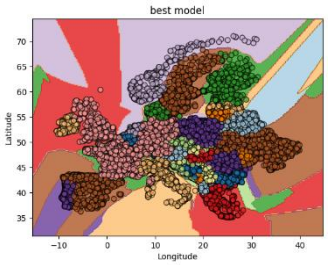


(1, 1): This setting exhibits non-convergence, likely attributed to the fact that the gradient descent is calculated after every example, making it highly sensitive to noise. The frequent updates may result in erratic behavior, hindering the model from finding a stable solution.

- (16, 10): Similar to (1, 1), this configuration also fails to converge and displays numerous spikes. The erratic behavior makes it challenging to assess its speed accurately, as the model's performance seems volatile. - (128, 50): Convergence is observed after approximately 2000 iterations. The loss stabilizes with occasional minor spikes. This indicates a more stable learning process compared to the previous configurations, providing a clearer view of the model's behavior.

- (1024, 50): This setting also fails to converge, or at least it's challenging to ascertain convergence due to the high batch number. The limited number of iterations may prevent us from observing whether the loss stabilizes at a low value.

6.2



1. **Model "(1,16)":** Batch size = 256, Epochs = 29, Learning rate = 0.01, Validation accuracy = 0.897.

Model "(2,16)": Batch size = 16, Epochs = 46, Learning rate = 0.001, Validation accuracy = 0.917.

Model "(6,16)": Batch size = 256, Epochs = 86, Learning rate = 0.01, Validation accuracy = 0.945.

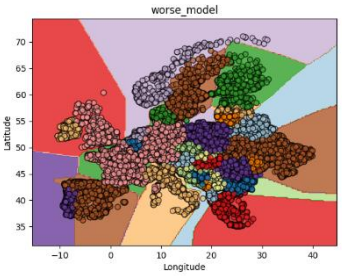
Model "(10,16)": Batch size = 1024, Epochs = 73, Learning rate = 0.01, Validation accuracy = 0.948.

Model "(6,8)": Batch size = 1024, Epochs = 83, Learning rate = 0.01, Validation accuracy = 0.900.

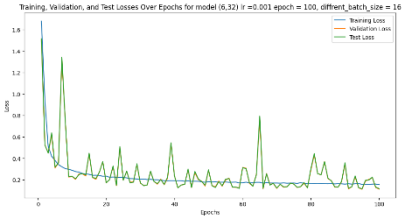
Model "(6,32): Batch size = 16, Epochs = 100, Learning rate = 0.001, Validation accuracy = 0.955.

Model "(6,64)": Batch size = 1024, Epochs = 100, Learning rate = 0.001, Validation accuracy = 0.915.

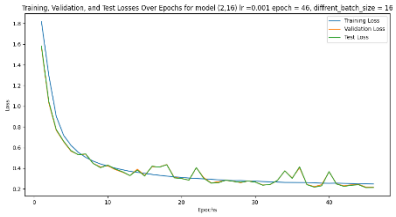
It appears that Model "(6,32)" achieved the highest validation accuracy of 0.955, making it the best-performing model among those you tested. Model configurations such as batch size, number of epochs, and learning rate significantly impact model performance, and these results provide insights into the effectiveness of different setups.

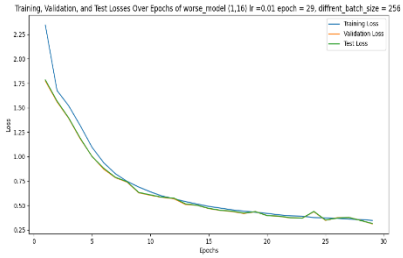


the best model is with acc of 0.954 is model 6, with depth 6 and width 32 and lr 0.001 and batch size of 16, we can see that those para allowed this model the achieve high acc quickly and keep improving all the time without doing a sever overfitting but we can see that the validation and test loss have a lot of spikes, this nosie comes from the fact the our batch size is quite small, which led to high sensitivity to example with high nosie, which led to spikes.



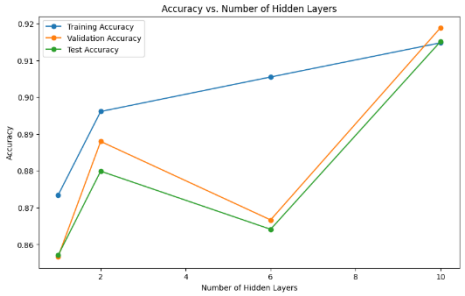
2.the worst is model 1 ,with 0.897 acc validation.



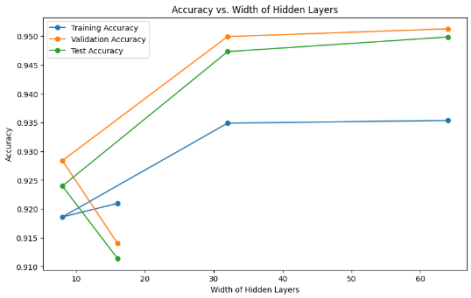


We can see here that our model generalizes well and the validation acc is close to the train acc, like the best model we have spikes from the same reasons, but smaller spikes indicate less overfitting.

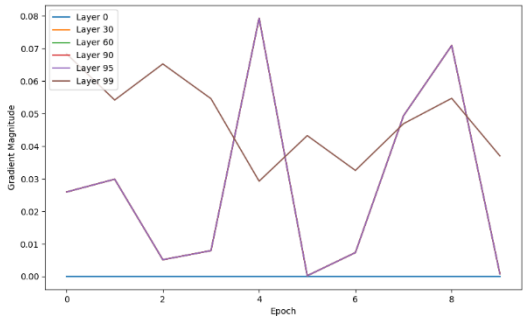
3. we compered between the models "(1,16)","(2,16)","(6,16)","(10,16)", but we got here that (6,16) did worse then he did on 6.2, what indicated that we got a validation acc on validation data that was similar to the train data, what led us the miss the overfit , or that we got in our case a result with a lot of noise, what led our model to not generalize well(our just because of the different starting weights cause of the seed), in each case we can conclude that adding more layers to a model can make it better at understanding complex patterns in data. It helps the model to learn different levels of details, like recognizing simple shapes and then combining them into more complex features. But having too many layers might cause problems during training, like making it slow to learn or memorize things that don't really matter. It can also make the model too complicated, especially if there's not a lot of data to learn from. This complexity can lead to overfitting, where the model gets too good at the training data but struggles to generalize to new, unseen data. So, while having more layers can be good for learning, we need to be careful not to make the model too deep, or it might have a hard time learning effectively and end up fitting too closely to the training data. The plot confirms that with ten layers, the model generalizes well to validation and test data, achieving accuracy better than the training set(due the fact the the after multiplying all those matrix each data receive a better "הקשר" which might not have the same effect on the training and the validation set, which led them to be higher. However, with six layers, while the training



accuracy increases, the validation and test accuracies drop, suggesting overfitting (or the other option like we offered). This underscores the need to balance model complexity, as too many layers can lead to overfitting, compromising performance on new, unseen data.

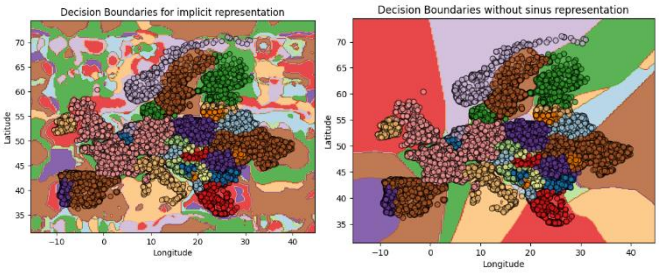


4. models names (6,16),"(6,8)","(6,32)","(6,64)" we can see that Enhancing the number of neurons generally boosts the model's ability to capture intricate patterns and relationships in the data. However, this improvement comes with trade-offs. Wider networks carry a higher risk of overfitting, especially with limited datasets, as they may memorize training data without effectively generalizing. Additionally, wider networks are computationally more expensive. In our case, increasing the number of neurons has enhanced the model's pattern-capturing capacity without causing overfitting. Notably, the test and validation accuracies increased with the increment in the number of neurons. Surprisingly, the training accuracy did not show a significant increase when transitioning from a width of 32 to 64. This could be attributed to the fact that 32 neurons were sufficient to capture the dataset's patterns or that 32 was close to the point of overfitting. Consequently, the introduction of 64 neurons did not considerably alter the training accuracy but positively influenced the validation and test accuracies.



5. We observed that all the layers, except for 95 and 99, exhibit a gradient magnitude of 0, indicating a vanishing gradient problem. To address this issue, potential solutions include leveraging Batch Normalization, Skip Connections/Residual Networks (ResNets), and Gradient Clipping. Batch Normalization proves valuable for normalizing layer inputs during training, thereby mitigating the vanishing gradient problem and maintaining stable activations. Skip Connections or Residual Networks enhance gradient flow through the incorporation of shortcut connections, facilitating smoother backpropagation and more effective learning across layers. Additionally, Gradient Clipping is a beneficial technique that imposes limits on gradient values, preventing excessively small gradients that can impede the optimization process. Implementing these approaches can help overcome vanishing gradient challenges and enhance the training stability of deep neural networks.

7.our model with the sinus rep achieve Train Accuracy: 0.963 Validation Accuracy: 0.962 Test Accuracy: 0.959 in contrast to our model without the sinus data that achieved only Train Acc: 0.889, Val Acc: 0.895, Test Acc: 0.891



The observed improvement in model accuracy when incorporating sinusoidal representations can be attributed to several factors. Firstly, sinusoidal functions introduce non-linear patterns to the data, enabling neural networks to capture complex relationships more effectively. The augmented feature space contributes to a richer set of information for the model to learn from, especially if the original feature space lacked expressiveness (only 2 features). The increased complexity introduced by the additional features enables the model to represent more intricate relationships in the data. Moreover, sinusoidal functions are commonly employed to model time-series data, making them effective for capturing temporal dependencies if present. Lastly, the generation of new features through sinusoidal functions acts as a form of data augmentation, potentially preventing overfitting and enhancing generalization. The effectiveness of sinusoidal features is context-dependent and may vary based on the nature of the data and the specific problem at hand, often requiring experimentation for optimal utilization.

7

1. XGBoost Models:

- The best XGBoost model achieved a test score of 0.7375 with a learning rate of 0.1, showcasing the positive impact of a larger learning rate. The second-best model scored 0.71 with a learning rate of 0.01, while the worst model scored 0.63 with a learning rate of 1e-05. The trends observed suggest that a larger learning rate facilitates quicker convergence, allowing the model to explore the parameter space more efficiently. However, an excessively large learning rate might lead to overshooting and poorer performance.

2. From Scratch Model:

- In the model trained from scratch, the best performance was observed with an accuracy of 0.54 and a learning rate of 0.01. The second-best model achieved an accuracy of 0.52 with a learning rate of 0.1, and the worst model scored 0.5 with a learning rate of 1e-05. The lower accuracy (0.54) may be attributed to the inherent challenge of training all weights from scratch. The model requires more epochs to find the global minimum, and a balance between a reasonable learning rate and training duration is crucial.

3. Linear Probing:

Linear probing demonstrated its effectiveness, with the best model achieving an accuracy of 0.7175 and a learning rate of 0.1. The second-best model scored 0.665 with a learning rate of 0.001, and the least successful model achieved an accuracy of 0.5075 with a learning rate of 1e-05. Linear probing leverages a pre-trained model, enriching input data with learned features. Interestingly, fine-tuning outperformed linear probing, indicating that further optimization of weights is more effective than linear model processing.

4.Sklearn Linear Regression Model:

The Sklearn linear regression model scored a train accuracy of 0.553, test accuracy of 0.5275, and validation accuracy of 0.5. Its performance closely resembled the "from scratch" model, indicating that training in isolation without leveraging pre-existing knowledge resulted in comparable accuracy. This trend highlights the importance of feature engineering and leveraging pre-trained models for better performance.

5. Fine-Tuning Model:

-Fine-tuning showcased its prowess, with the best model achieving an accuracy of 0.8125 and a learning rate of 0.0001. The second-best model scored 0.7875 with a learning rate of 0.001, while the least successful model achieved an accuracy of 0.495 with a learning rate of 0.1. The success of a small learning rate (0.0001) suggests that the fine-tuning model was already in close proximity to the minimum during training. However, the surprising underperformance of the model with a learning rate of 0.1 may be attributed to overshooting the minimum due to the model's proximity or a fortuitous starting point in the "from scratch" model. These trends emphasize the delicate balance required in choosing an optimal learning rate for effective fine-tuning..

7.2:

They are the images that my best model, fine tuning with a learning rate of 0.001 classify as fake images and was fake, and my worst model in another baseline from scratch with lr of 0.00001, classify as real images