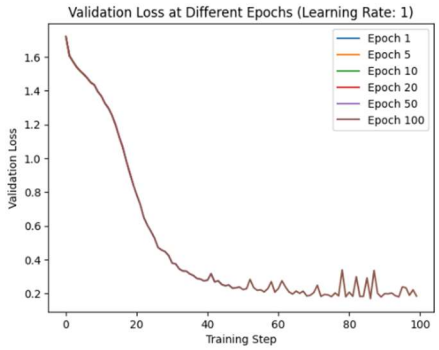
Daniel Nisnevich 207447970



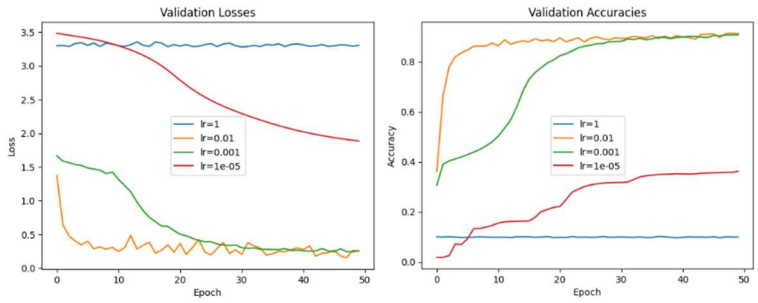
Question 6.1.2.1:

For the blue line(high learning rate = 1) we can see that the model seems unstable because of the oscillations and no improvement throughout the learning.

For the medium yellow line (lr = 0.01) we do see oscillations however it does seem to minimize the loss pretty quickly and stay near that minima.

For the Red Line with very low l.r we can see that it does slowly and steadily converge however it will take a lot of epochs to achieve that goal.

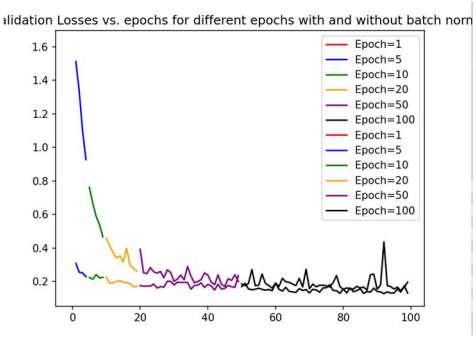
Looks like the best's were 0.01 , 0.001 (0.001 was more steady but slower).



In order to save space in the pdf I merged the question of epochs and batch normal

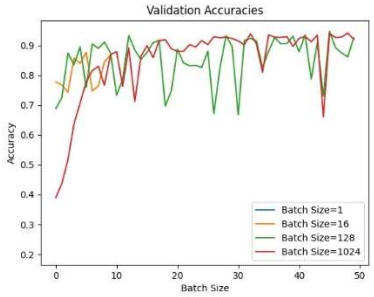
Overall while exploring the effects of different amount of epochs on the minimization of loss and optimization of accuracy I saw that for some training there was a fluctuation after 80 epochs but after training again sometimes the model did achieve better results with 100 epochs. I do conclude that too many epochs can might lead to overfitting and might damage the accuracy of the model. Although, when training again sometime I did see that the model was better with many epochs.

On the first epochs the model is trying to learn the complexity of the data and find patterns after that it usually around the 10^{th}-20^{th} epoch does achieve some minima and only moves a bit around it for the rest of the epochs.



The batch normal is the lower curve as we learned in class I can see during the training with batch normal that batch normal affect the learning curve in a good way, the minima is reached a lot faster than without the normal. Overall the accuracy was better with batch normal and achieved faster somewhat good accuracy (In compression to this exercise's accuracy).

Batches:

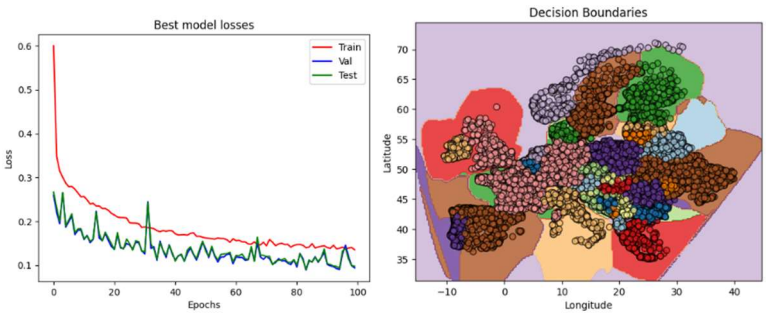


I understand I might have a problem with my graph as there is too much fluctuations however due to the collab free gpu ending I wont run it again unfortunately

In this experiment I did achieve better accuarcies with 1024 batches and 128 but as you can see the deviations are a bit suspicious to problem during training. I do think that the epochs amount had impact in this experiment too.1

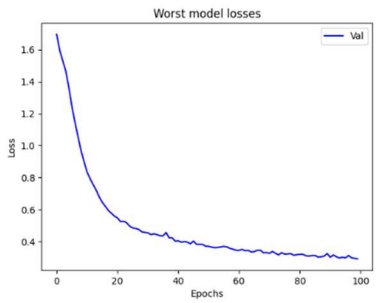
Question 6.2.1 , Parmaters: LR -0.001 , width of 64 depth of 6, with batch normal

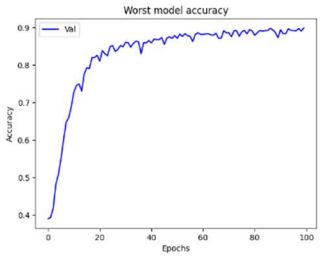
And batch size of 256

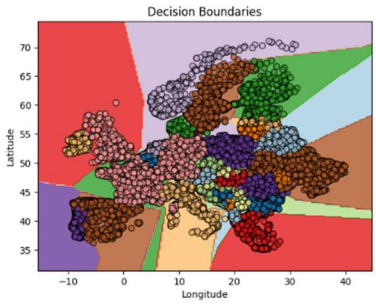


Achieved accuracy 0.**962** on the Validation set

However the worst model achieved with depth of 1 and width of 16 an accuracy of 0.900





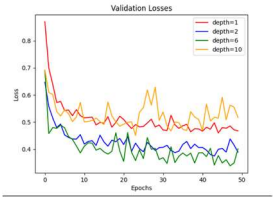


Best model Did generalize well as we can see from the good results and graphs during the training.

I do think that the batch normalization helped avoiding overfitting because I did do 100 epochs but it didn't fluctuate and overfit to the training data.

The worst model also did comparatively well it did achieve a lower accuracy but it was able to minimize the loss with every epoch and not overfit on the training data even though it had

100 epochs as well.



3) affects of depth on the training of the model;

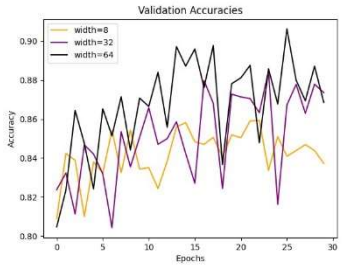
The effect of the amount of layers on the model is that with many layers make the model complex and able to define patterns that are pointing to a way to generalize the ability to achieve the best accuracy.

There is a trade off in making the network deeper, the runtime is longer as the backward-pass and forward-pass are longer and if we want to train with many epochs without GPU it can be uncomfortable and unnecessary, as our specific data did achieve very high accuracy using trees in the previous exercise.

I do know that deep neural networks are needed to achieve hard goals as an AI Chat bot but for our data it is not as good and the tradeoff isn't worth it.

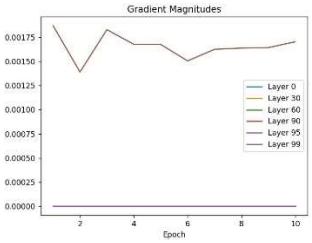
4)

Width of the model:



By enlarging the amount of neurons in each layer we can increase the ability to capture complexity same as with depth it not only captures more patterns but captures overall details in a better way. It does increase the risk of overfitting and we do see that on our dataset it did fluctuate more . I do think that increasing the width in this dataset is not good and it affected the accuracy of the model badly.

Gradient Magnitude:



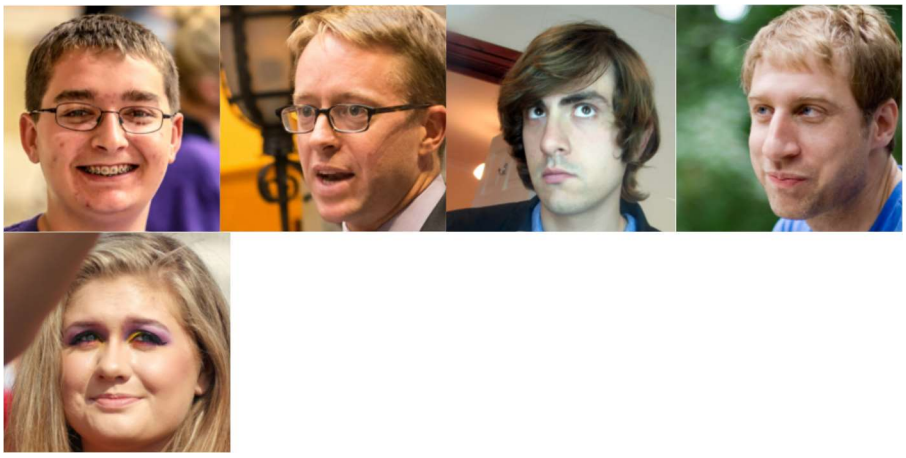
The behavior that we can see in this graph is that initially it we can see that the last layer has somewhat gradient but afterwards the graph flatns, even the highest value is 0.00175 which is also very low. As gradient magnitude reflects " how much" the model's parameter in the layer need to be adjusted , as we expect the changes flattens out.

CNN

Xgboost :

```
Train: 100%| | 44/44 [00:35<00:00,  1.23it/s]
Test: 100%| | 13/13 [00:10<00:00,  1.19it/s]
Train accuracy: 1.00
Test accuracy: 0.73
```

LR	0.1	0.01	0.001	0.0001
Scratch	0.475	0.535	0.45	0.556
Probing	0.72	0.695	0.635	0.495
Tuning	0.5	0.563	0.725	0.73



I added 5 images in which the best model succeed at guessing the label and the worst

model didn’t.

Best model was with tuning and LR of 0.001 achieved a 72.5% success rate

Worst one was from scratch with the same LR of 0.001.

From the scratch row we can deduce that the lower the learning rate the better is the generalization on the training data, I do take this sentence not with caution as we ran it for 1 epoch.

From the probing row it does look like the opposite is happening the higher the learning rate the better is the accuracy but still I say this with caution because maybe it just achieved a minima faster with 1 epoch and when looking at more epochs it would easily be the opposite.

Tuning - again we can see from this row that the lower the learning rate the better is the accuracy of the model which does seem appropriate.