CS 203 Software Tools and Techniques for AI

Group: 12
Name: Harinarayan J (23110128) , Vyomika Vasireddy (23110363)

Github Repository: https://github.com/HarinarayanJ/CS203-Lab6

Section 1: MLP Model Implementation & Experiment Tracking

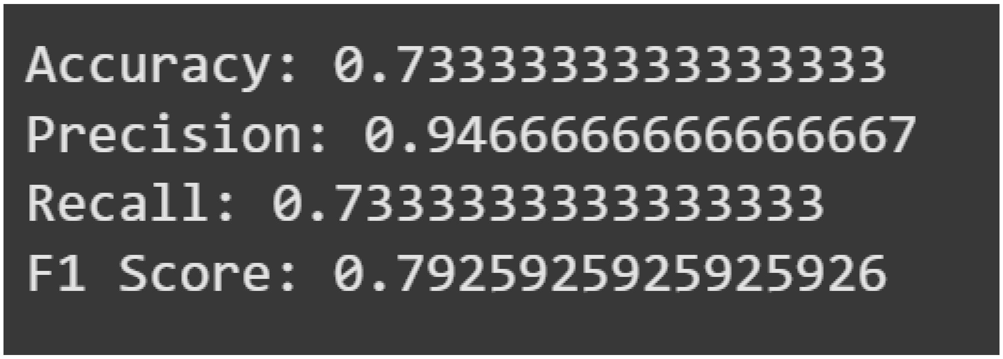
We used Python 3.11.11 for the first part.

We loaded the data, normalized it, and split it for the data cleaning process.

We used a MLP model with 4 neurons (input), 16 neurons (hidden), 3 neurons (output). We used the Cross Entropy Loss and Adam Optimizer.

We ran the model for 50 epochs.

These are the accuracy, precision, recall, F1 score for the first 50 epochs.



```
Accuracy: 0.7333333333333333
Precision: 0.9466666666666667
Recall: 0.7333333333333333
F1 Score: 0.7925925925925926
```

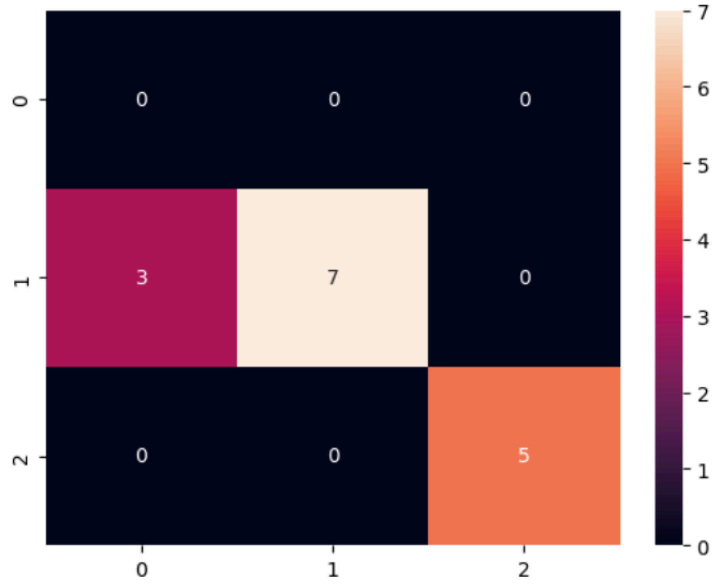


Fig. Confusion Matrix

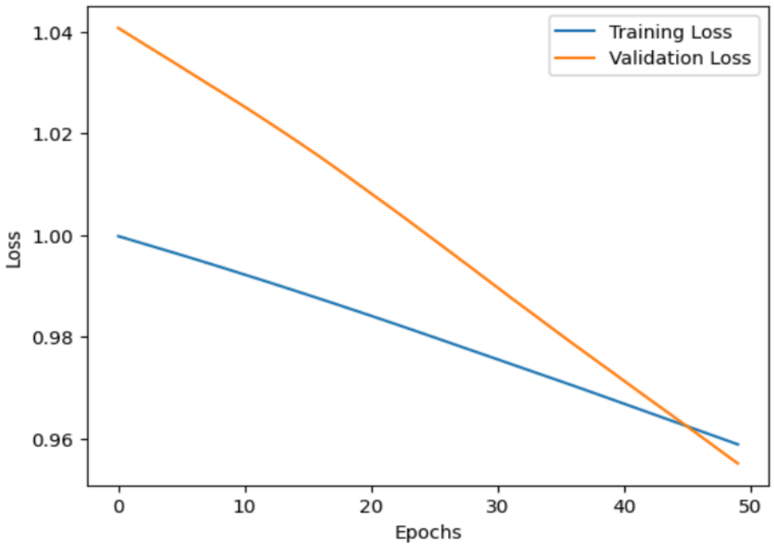
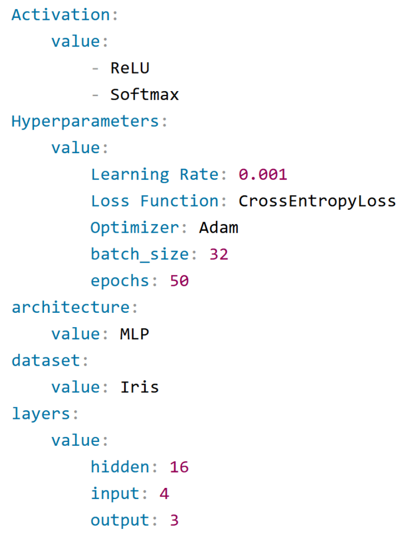


Fig. Loss vs epoch

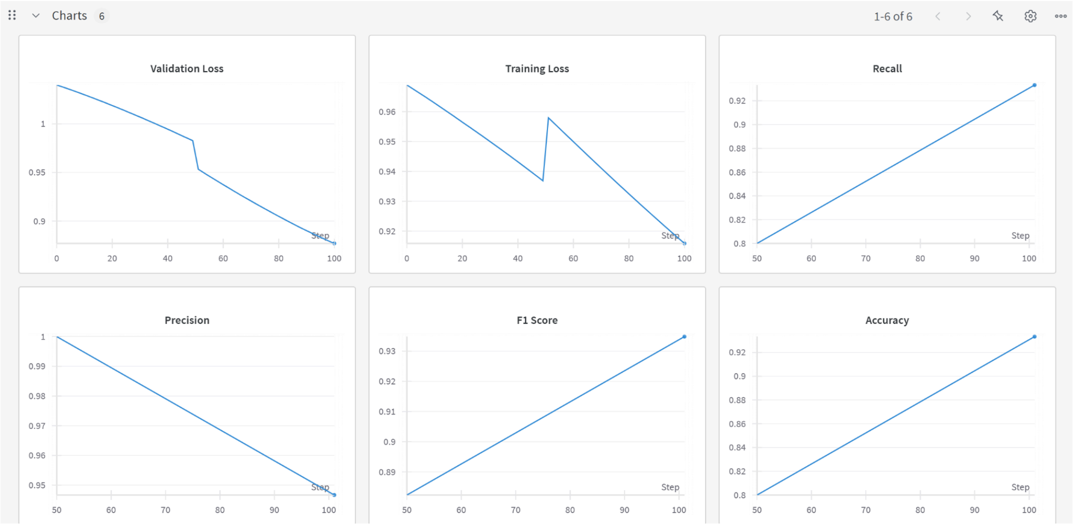
This is the comparison between the training and validation loss over the epochs.

Now, we used the Weights and Biases module to check for the hyperparameters.

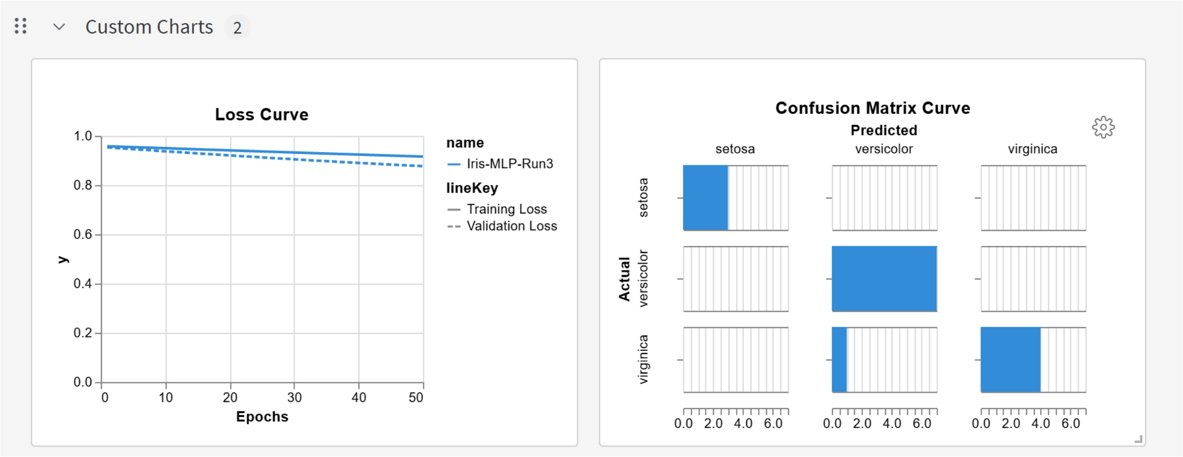


```
Activation:
    value:
        - ReLU
        - Softmax
Hyperparameters:
    value:
        Learning Rate: 0.001
        Loss Function: CrossEntropyLoss
        Optimizer: Adam
        batch_size: 32
        epochs: 50
architecture:
    value: MLP
dataset:
    value: Iris
layers:
    value:
        hidden: 16
        input: 4
        output: 3
```

Parameters from the wandb site.



Charts of the losses and metrics.

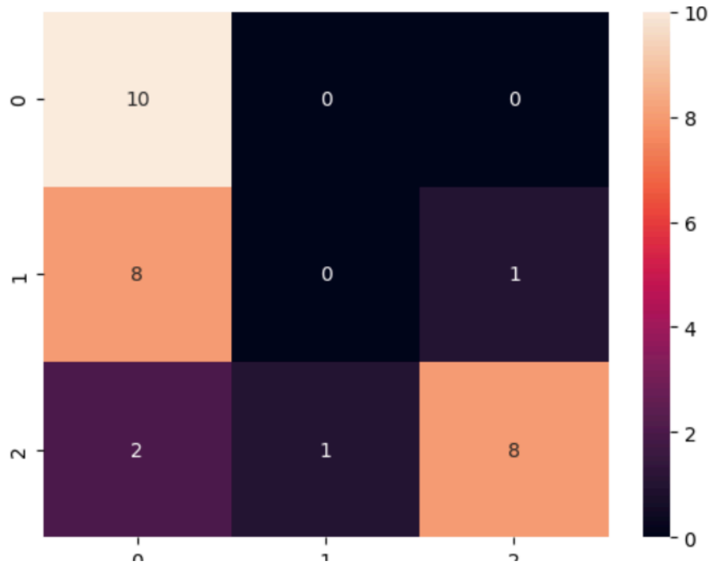


Two custom charts with the loss curves and the confusion matrices.

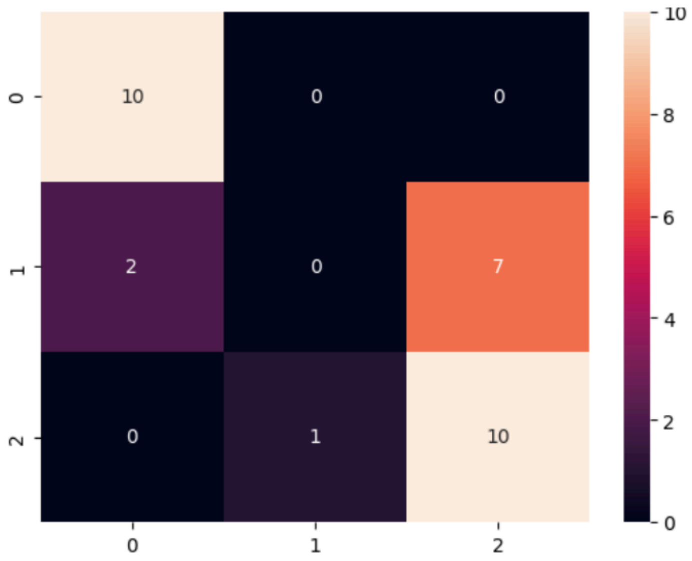
Section 2: Hyperparameters

First, we use the manual tuning, where we just iterate through all the possibilities, the following are the confusion matrices from each of the possibilities of the hyperparameters.

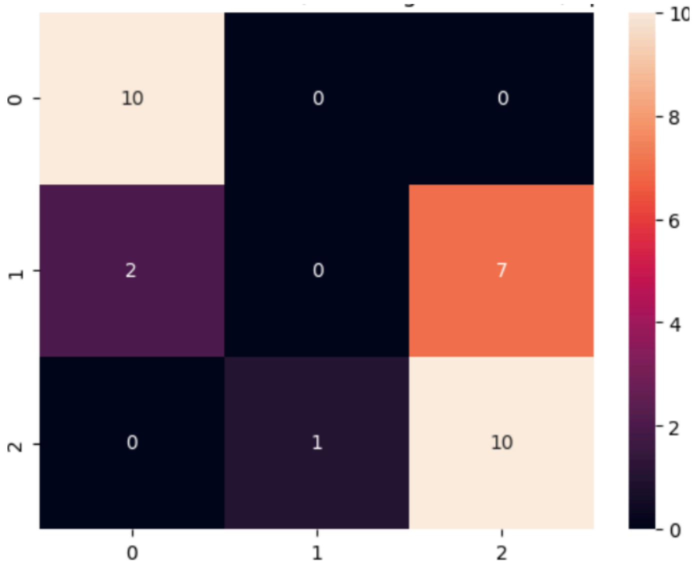
Confusion matrix



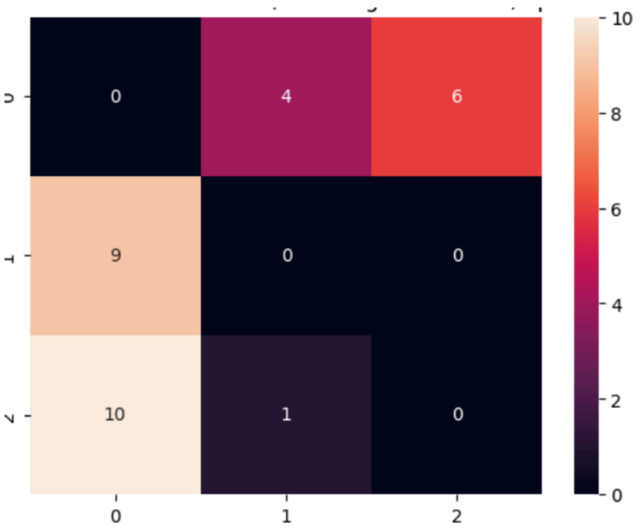
Confusion Matrix Batch Size 2, Learning Rate: 0.001, Epochs: 1



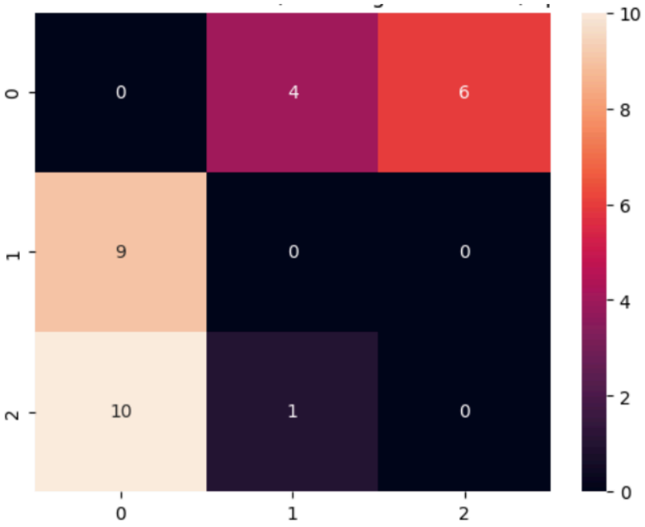
Confusion Matrix Batch Size 2, Learning Rate: 0.001, Epochs: 3



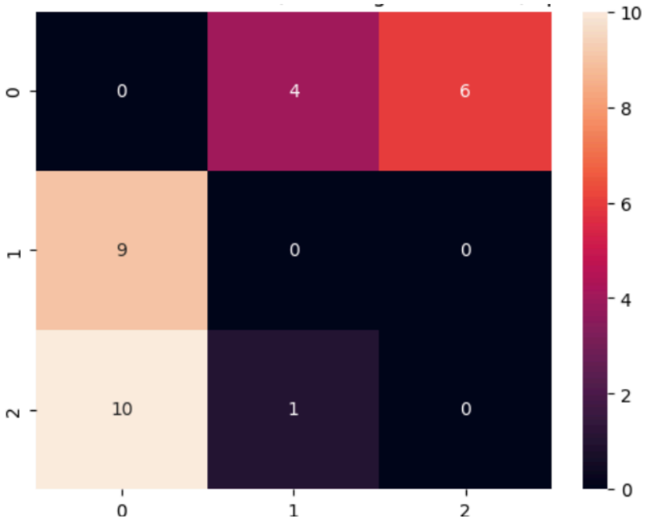
Confusion Matrix Batch Size 2, Learning Rate: 0.001, Epochs: 5



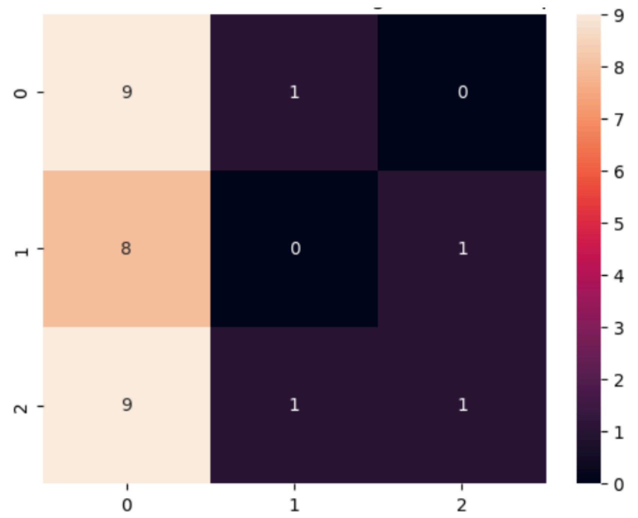
Confusion Matrix Batch Size 2, Learning Rate: 0.00001, Epochs: 1



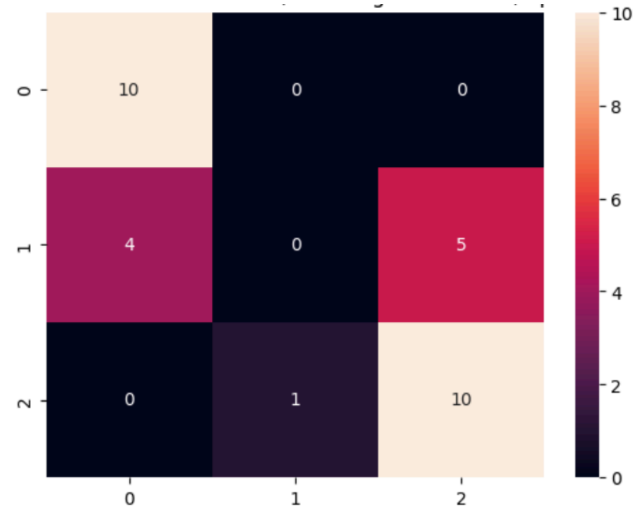
Confusion Matrix Batch Size 2, Learning Rate: 0.00001, Epochs: 3



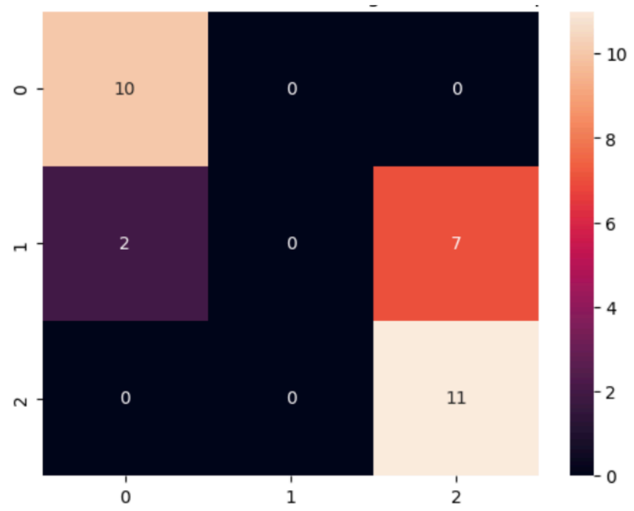
Confusion Matrix Batch Size 2, Learning Rate: 0.00001, Epochs: 5



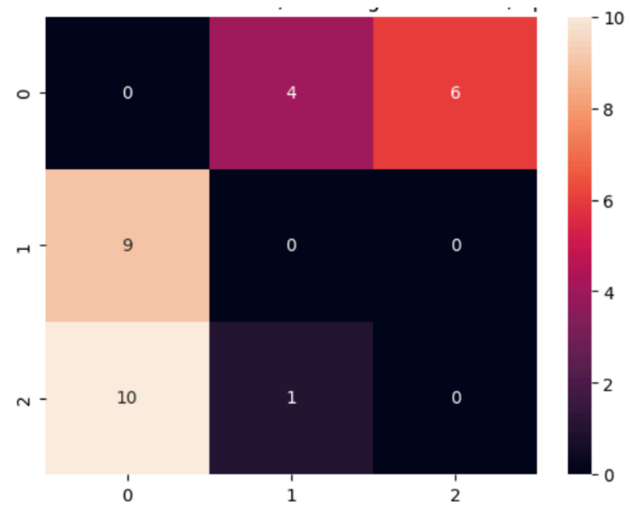
Confusion Matrix Batch Size 4, Learning Rate: 0.001, Epochs: 1



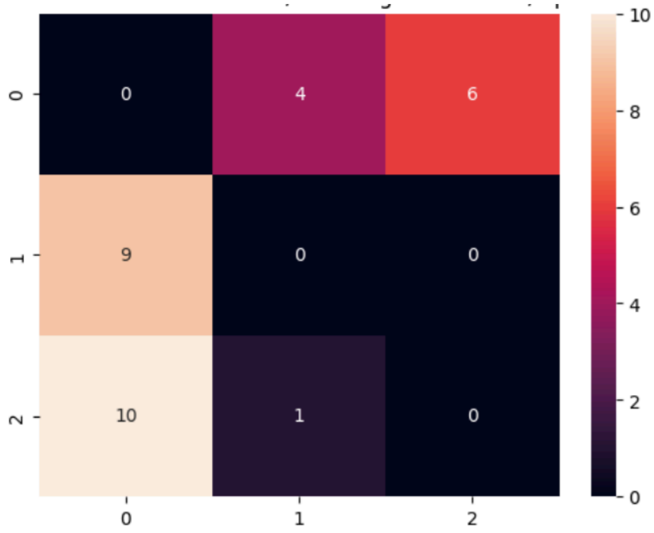
Confusion Matrix Batch Size 4, Learning Rate: 0.001, Epochs: 3



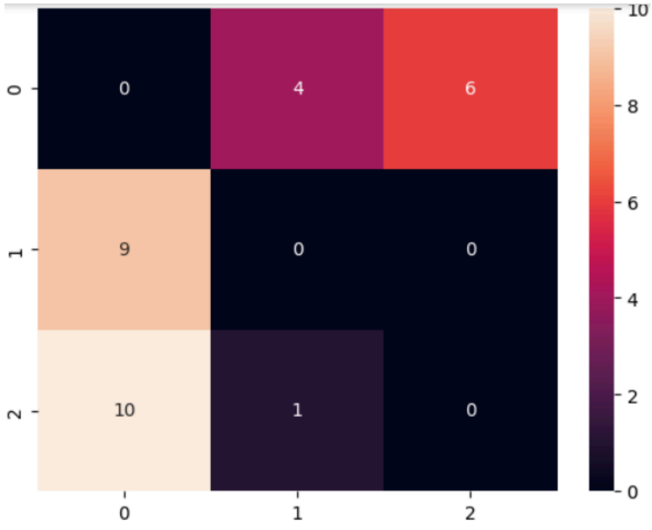
Confusion Matrix Batch Size 4, Learning Rate: 0.001, Epochs: 5



Confusion Matrix Batch Size 4, Learning Rate: 0.00001, Epochs: 1

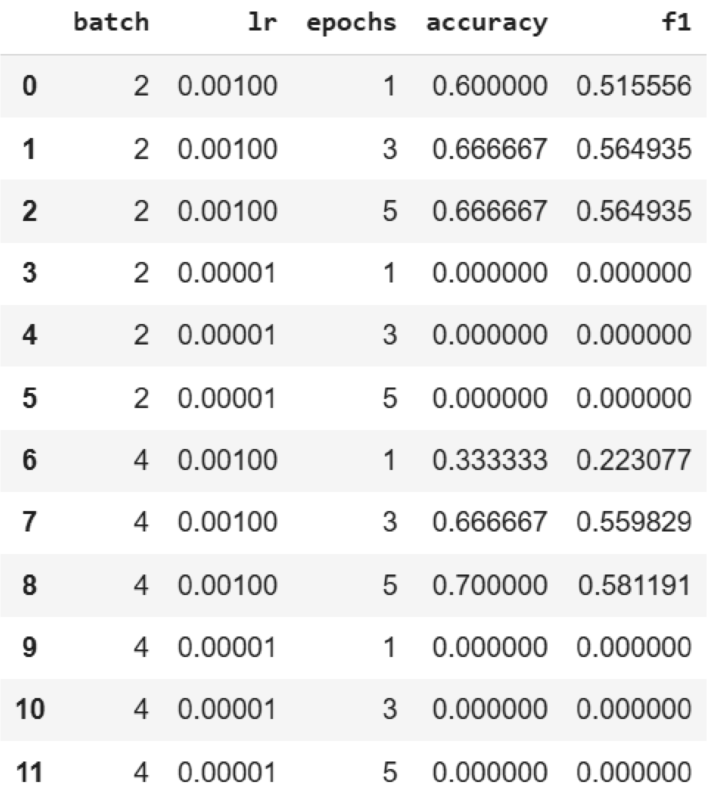


Confusion Matrix Batch Size 4, Learning Rate: 0.00001, Epochs: 3



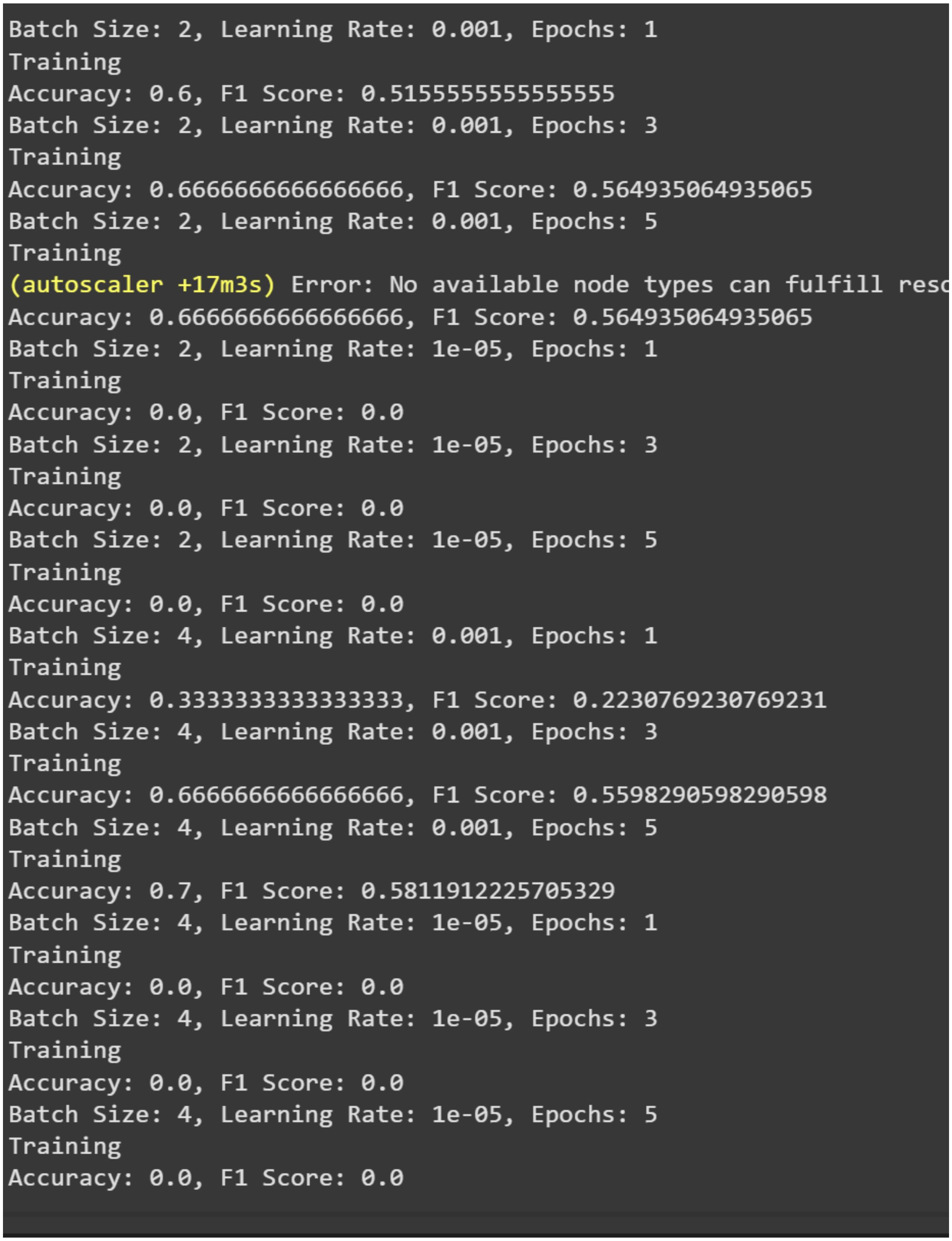
Confusion Matrix Batch Size 4, Learning Rate: 0.00001, Epochs: 5

The following is the metrics dataframe which shows the accuracy and F1 score of all the possibilities.



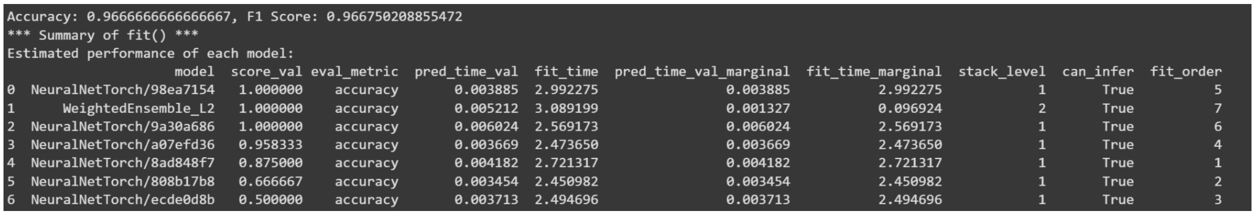
	batch	lr	epochs	accuracy	f1
0	2	0.00100	1	0.600000	0.515556
1	2	0.00100	3	0.666667	0.564935
2	2	0.00100	5	0.666667	0.564935
3	2	0.00001	1	0.000000	0.000000
4	2	0.00001	3	0.000000	0.000000
5	2	0.00001	5	0.000000	0.000000
6	4	0.00100	1	0.333333	0.223077
7	4	0.00100	3	0.666667	0.559829
8	4	0.00100	5	0.700000	0.581191
9	4	0.00001	1	0.000000	0.000000
10	4	0.00001	3	0.000000	0.000000
11	4	0.00001	5	0.000000	0.000000

Metrics(accuracy/learning rate/epoch/f1 score)



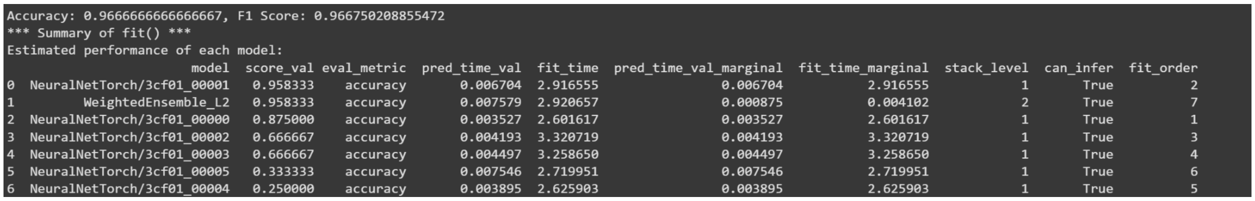
```
Batch Size: 2, Learning Rate: 0.001, Epochs: 1
Training
Accuracy: 0.6, F1 Score: 0.5155555555555555
Batch Size: 2, Learning Rate: 0.001, Epochs: 3
Training
Accuracy: 0.6666666666666666, F1 Score: 0.564935064935065
Batch Size: 2, Learning Rate: 0.001, Epochs: 5
Training
(autoscaler +17m3s) Error: No available node types can fulfill reso
Accuracy: 0.6666666666666666, F1 Score: 0.564935064935065
Batch Size: 2, Learning Rate: 1e-05, Epochs: 1
Training
Accuracy: 0.0, F1 Score: 0.0
Batch Size: 2, Learning Rate: 1e-05, Epochs: 3
Training
Accuracy: 0.0, F1 Score: 0.0
Batch Size: 2, Learning Rate: 1e-05, Epochs: 5
Training
Accuracy: 0.0, F1 Score: 0.0
Batch Size: 4, Learning Rate: 0.001, Epochs: 1
Training
Accuracy: 0.3333333333333333, F1 Score: 0.2230769230769231
Batch Size: 4, Learning Rate: 0.001, Epochs: 3
Training
Accuracy: 0.6666666666666666, F1 Score: 0.5598290598290598
Batch Size: 4, Learning Rate: 0.001, Epochs: 5
Training
Accuracy: 0.7, F1 Score: 0.5811912225705329
Batch Size: 4, Learning Rate: 1e-05, Epochs: 1
Training
Accuracy: 0.0, F1 Score: 0.0
Batch Size: 4, Learning Rate: 1e-05, Epochs: 3
Training
Accuracy: 0.0, F1 Score: 0.0
Batch Size: 4, Learning Rate: 1e-05, Epochs: 5
Training
Accuracy: 0.0, F1 Score: 0.0
```

Bayes Search using AutoGluon 1.2

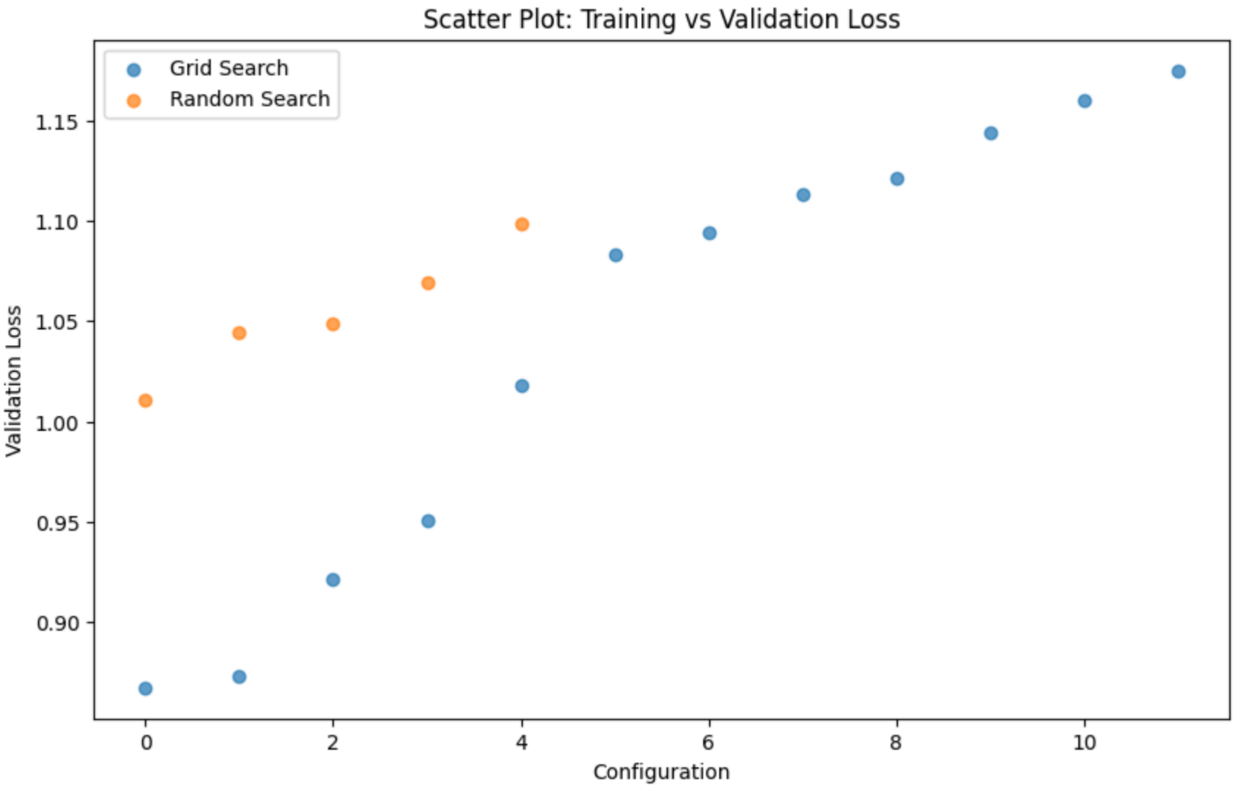


```
Accuracy: 0.9666666666666667, F1 Score: 0.966750208855472
*** Summary of fit() ***
Estimated performance of each model:
                     model  score_val eval_metric  pred_time_val  fit_time  pred_time_val_marginal  fit_time_marginal  stack_level  can_infer  fit_order
0  NeuralNetTorch/98ea7154   1.000000    accuracy       0.003885  2.992275                0.003885           2.992275            1       True          5
1      WeightedEnsemble_L2   1.000000    accuracy       0.005212  3.089199                0.001327           0.096924            2       True          7
2  NeuralNetTorch/9a30a686   1.000000    accuracy       0.006024  2.569173                0.006024           2.569173            1       True          6
3  NeuralNetTorch/a07efd36   0.958333    accuracy       0.003669  2.473650                0.003669           2.473650            1       True          4
4  NeuralNetTorch/8ad848f7   0.875000    accuracy       0.004182  2.721317                0.004182           2.721317            1       True          1
5  NeuralNetTorch/808b17b8   0.666667    accuracy       0.003454  2.450982                0.003454           2.450982            1       True          2
6  NeuralNetTorch/ecde0d8b   0.500000    accuracy       0.003713  2.494696                0.003713           2.494696            1       True          3
```

Random Search using AutoGluon 1.2



```
Accuracy: 0.9666666666666667, F1 Score: 0.966750208855472
*** Summary of fit() ***
Estimated performance of each model:
                        model  score_val eval_metric  pred_time_val  fit_time  pred_time_val_marginal  fit_time_marginal  stack_level  can_infer  fit_order
0  NeuralNetTorch/3cf01_00001   0.958333    accuracy       0.006704  2.916555                0.006704           2.916555            1       True          2
1         WeightedEnsemble_L2   0.958333    accuracy       0.007579  2.920657                0.000875           0.004102            2       True          7
2  NeuralNetTorch/3cf01_00000   0.875000    accuracy       0.003527  2.601617                0.003527           2.601617            1       True          1
3  NeuralNetTorch/3cf01_00002   0.666667    accuracy       0.004193  3.320719                0.004193           3.320719            1       True          3
4  NeuralNetTorch/3cf01_00003   0.666667    accuracy       0.004497  3.258650                0.004497           3.258650            1       True          4
5  NeuralNetTorch/3cf01_00005   0.333333    accuracy       0.007546  2.719951                0.007546           2.719951            1       True          6
6  NeuralNetTorch/3cf01_00004   0.250000    accuracy       0.003895  2.625903                0.003895           2.625903            1       True          5
```



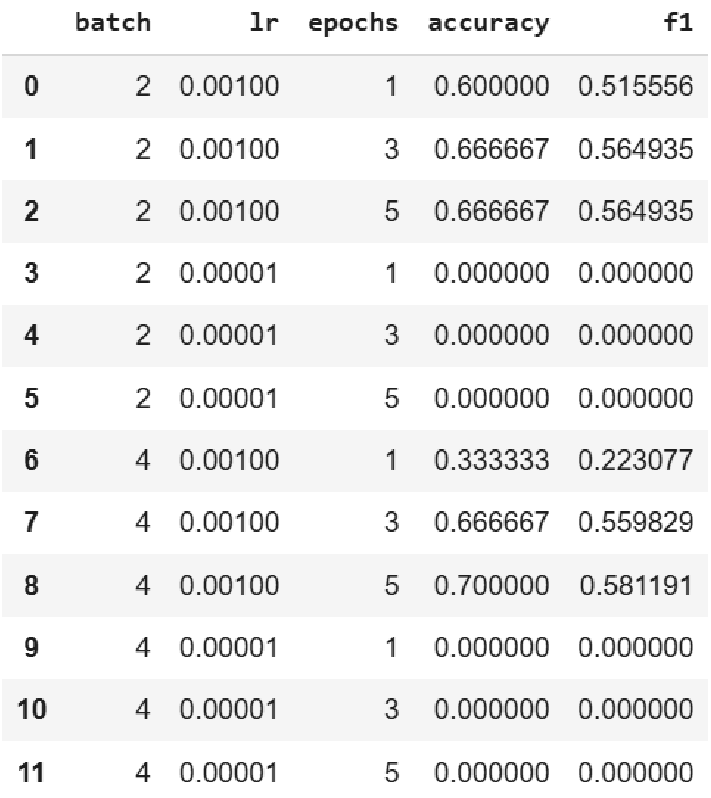
For automated search, we were able to do Random Search and Bayes Search on the current version of Python 3.11.11 and for the rest, we revert the python version to 3.9 as the AutoGluon library was updated from version 0.1.0 to 1.2.0 where we do not have the gridsearch and hyperband search functionality. So we use the old and deprecated version of autogluon 0.1.0. The documentation for that is in the second file in the github repository.

First, as mentioned in the classroom comments, I tried with python 3.9, however there was a package conflict as numpy won't work for the needed AutoGluon module at python 3.9. I tried various methods to fix this.

Then, I tried with python 3.8, after a lot of attempts I was able to get the AutoGluon version as required but the AutoGluon.tabulate was not able to be imported properly. Then, the PyTorch module also failed as it is not supported.

I have uploaded all these failed tries in the github repository. I ended up exhausting the credit allocation on two different Google Colab accounts. I request the TAs to check for deprecations before handing out assignments or at least inform us about which version to use, well in advance.

Performance for each hyperparameter combination over accuracy and F1.



	batch	lr	epochs	accuracy	f1
0	2	0.00100	1	0.600000	0.515556
1	2	0.00100	3	0.666667	0.564935
2	2	0.00100	5	0.666667	0.564935
3	2	0.00001	1	0.000000	0.000000
4	2	0.00001	3	0.000000	0.000000
5	2	0.00001	5	0.000000	0.000000
6	4	0.00100	1	0.333333	0.223077
7	4	0.00100	3	0.666667	0.559829
8	4	0.00100	5	0.700000	0.581191
9	4	0.00001	1	0.000000	0.000000
10	4	0.00001	3	0.000000	0.000000
11	4	0.00001	5	0.000000	0.000000

Here, in manual tuning we can see that the best combination is 4 batch, with 1e-3 lr, and for 5 epochs.

As expected, more epochs tend to improve performance especially since we have very less number of epochs. Learning rate here is higher but still is better (1e-3) since 1e-3 itself is a pretty low learning rate which seems to be ideal for this dataset whereas we do not learn anything from the 1e-5 as observed from 0 accuracies.

Compare manual tuning vs. automated search

In this case, manual tuning gave us better results as it was a very limited set so we could identify it easily. In a bigger hyperspace, automated search is better as even here it gave us reasonable results without much performance loss.