EPOCH

Focusing on optimal convergence

Convergence

An iterative process that produces a sequence of candidate solutions until ultimately arriving at the end of the process-(final solution)

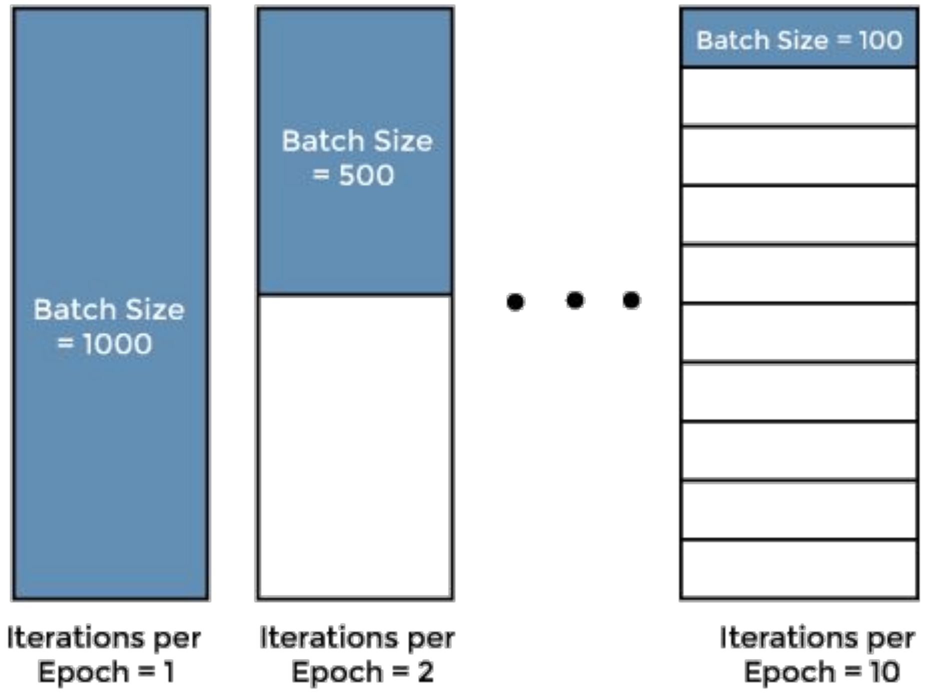
一個迭代過程，產生一系列候選解決方案，直到最終在過程結束時得出最終解決方案。

What is an epoch?

One entire passing of training data through the algorithm.

通過算法完整傳遞訓練數據

How the epoch work



Batch size refer to the number of training examples used in one iteration.

Batch size 是指在一次迭代中使用的訓練示例的數量

The larger the batch size, the quicker the our model training will complete each epoch while training.And vice versa.

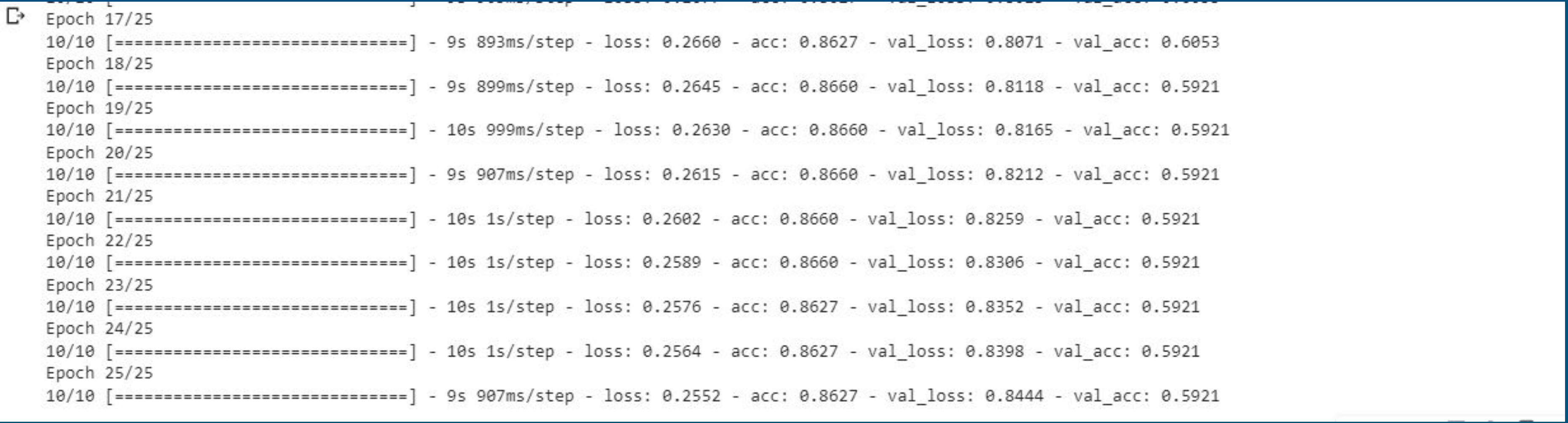
Batch size 越大，我們的模型訓練完成每個 Epoch 的速度就越快，而訓練將在訓練時完成每個 epoch

How the epoch work

Example:

Number of training examples = 300
Each batch size = 50
Number of iterations = 300/50 = 6
Therefore 1 epoch = 6 iterations.

Continued



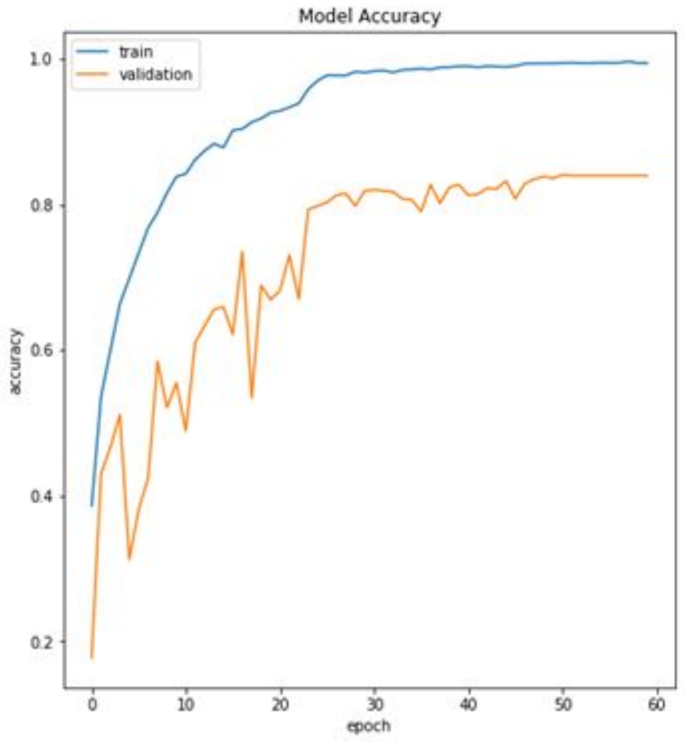
```
Epoch 17/25
10/10 [==============================] - 9s 893ms/step - loss: 0.2660 - acc: 0.8627 - val_loss: 0.8071 - val_acc: 0.6053
Epoch 18/25
10/10 [==============================] - 9s 899ms/step - loss: 0.2645 - acc: 0.8660 - val_loss: 0.8118 - val_acc: 0.5921
Epoch 19/25
10/10 [==============================] - 10s 999ms/step - loss: 0.2630 - acc: 0.8660 - val_loss: 0.8165 - val_acc: 0.5921
Epoch 20/25
10/10 [==============================] - 9s 907ms/step - loss: 0.2615 - acc: 0.8660 - val_loss: 0.8212 - val_acc: 0.5921
Epoch 21/25
10/10 [==============================] - 10s 1s/step - loss: 0.2602 - acc: 0.8660 - val_loss: 0.8259 - val_acc: 0.5921
Epoch 22/25
10/10 [==============================] - 10s 1s/step - loss: 0.2589 - acc: 0.8660 - val_loss: 0.8306 - val_acc: 0.5921
Epoch 23/25
10/10 [==============================] - 10s 1s/step - loss: 0.2576 - acc: 0.8627 - val_loss: 0.8352 - val_acc: 0.5921
Epoch 24/25
10/10 [==============================] - 10s 1s/step - loss: 0.2564 - acc: 0.8627 - val_loss: 0.8398 - val_acc: 0.5921
Epoch 25/25
10/10 [==============================] - 9s 907ms/step - loss: 0.2552 - acc: 0.8627 - val_loss: 0.8444 - val_acc: 0.5921
```

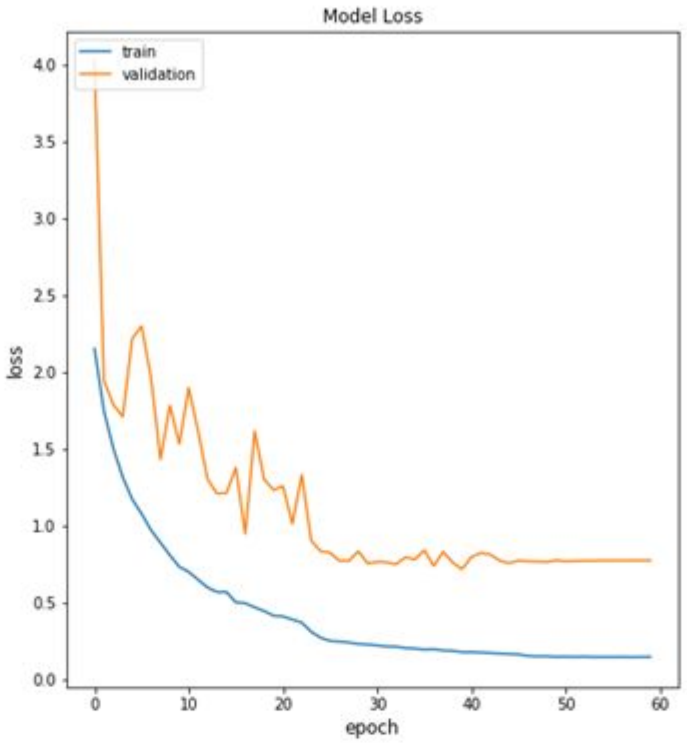
Accuracy and loss

Loss value implies how poorly or well a model behaves after each iteration of optimization.An accuracy metric is used to measure the algorithm's performance in an interpretable way.

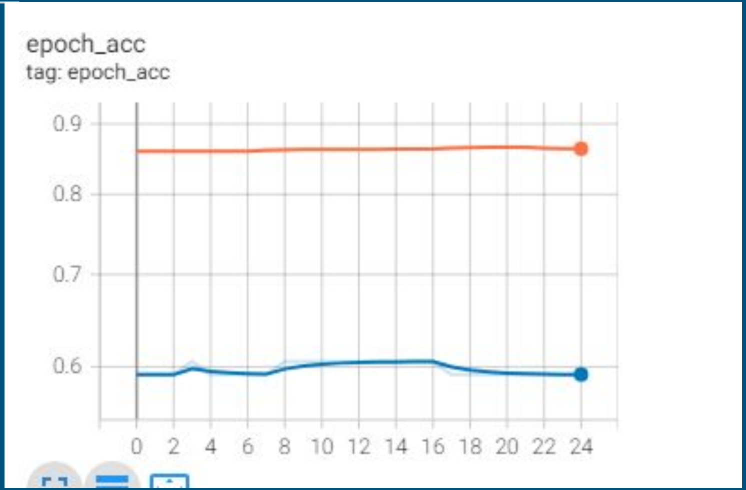
The accuracy of a model is usually determined after the model parameters and is calculated in the form of a percentage.

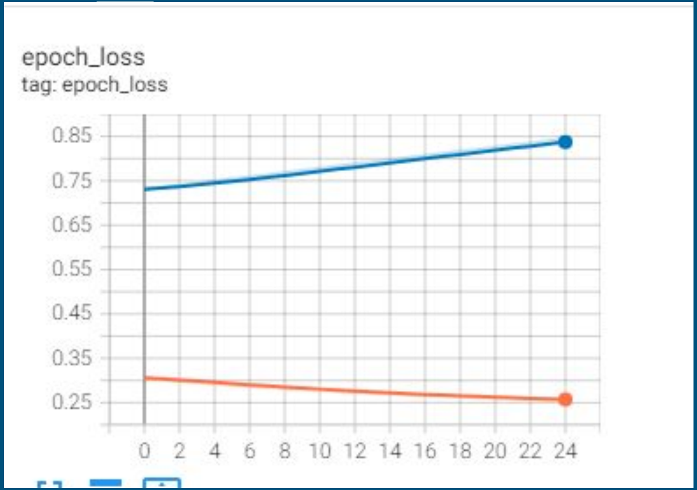
Accuracy and loss(Example graphs)





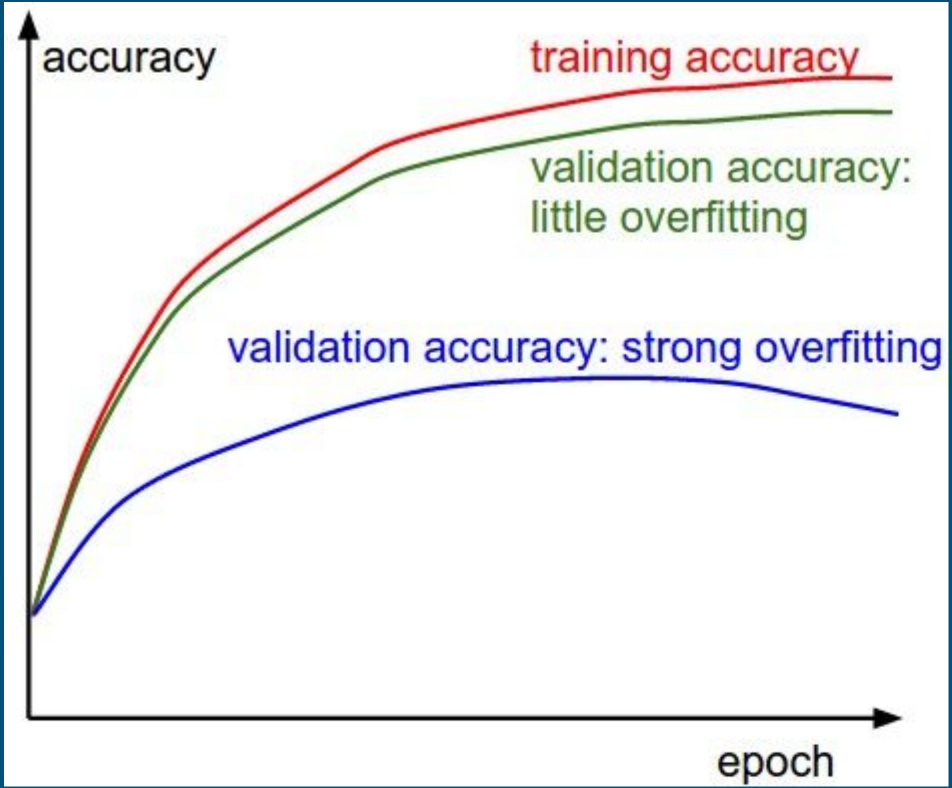
Example from my previous code

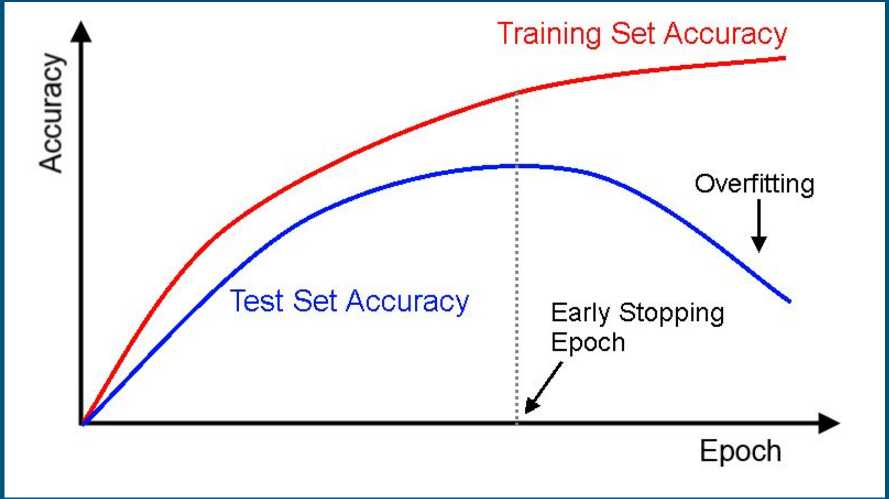




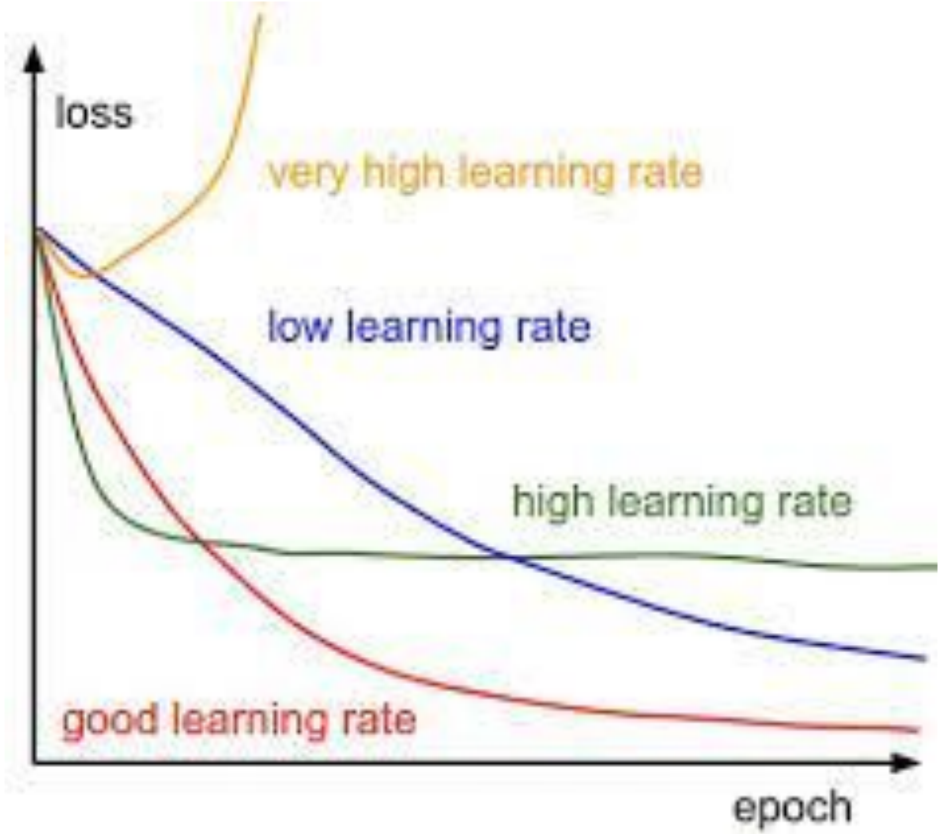
● train
● test

Accuracy vs epoch definitions





Loss vs epoch definitions



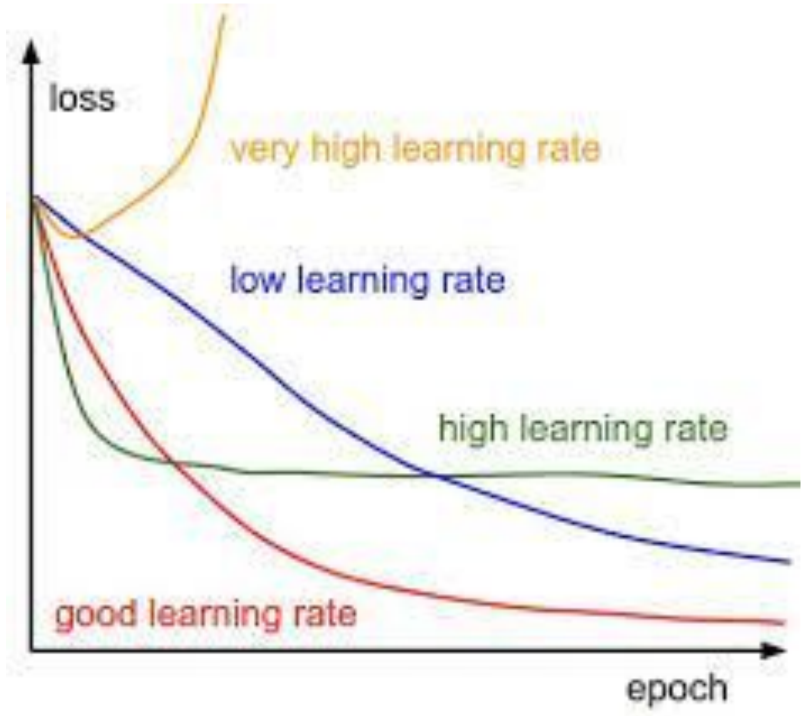
Learning rate

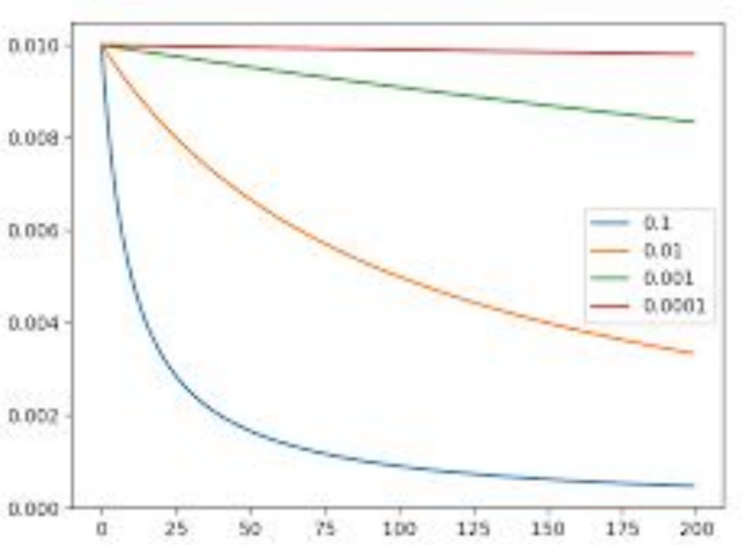
A tuning parameter in an optimization algorithm that determines the step size at each iteration while moving toward a minimum of a loss function.

Smaller learning rates require more training epochs.

Smaller batch sizes are suited to smaller learning rate.

Loss vs epoch and Learning rate relations





Previous code observations

Problem

- Previous code had only 25 epochs

Result

- The model was not able to fully learn from the data(It didn’t reach full or optimal convergence.)

How to solve the problem

1. Increase the number of epochs.But you have to consider certain factors to find the best number.

2.If there is an inflection point when training goes above the validation, you might be able to use early stopping(stop training)

3.If training and validation are both low, you are probably under fitting and you can probably increase the capacity of your network and train more or longer (increase the number of epochs).

So what determines the number of epochs?

If training and validation are both low, you are probably under fitting and you can probably increase the capacity of your network and train more or longer (increase the number of epochs).

If there is an inflection point when training goes above the validation, you might be able to use early stopping(stop training).

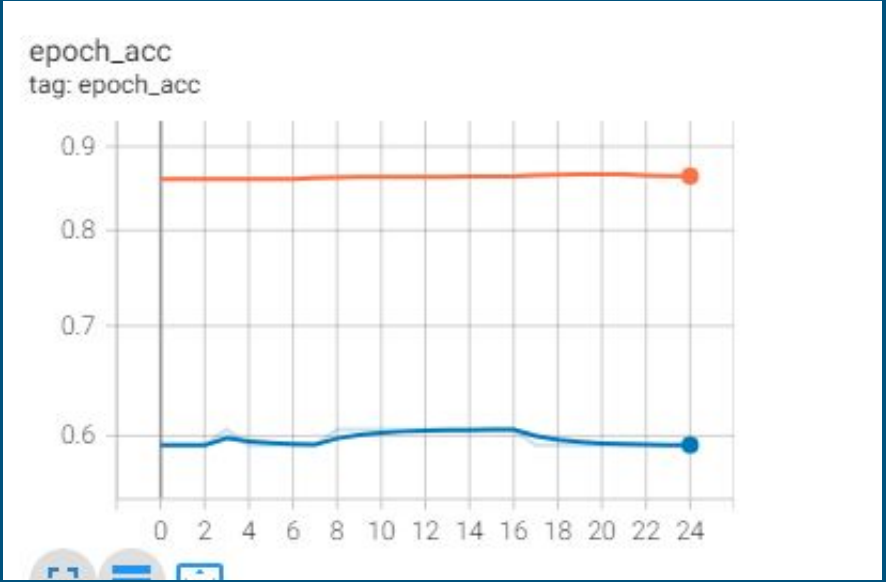
So what determines the number of epochs?

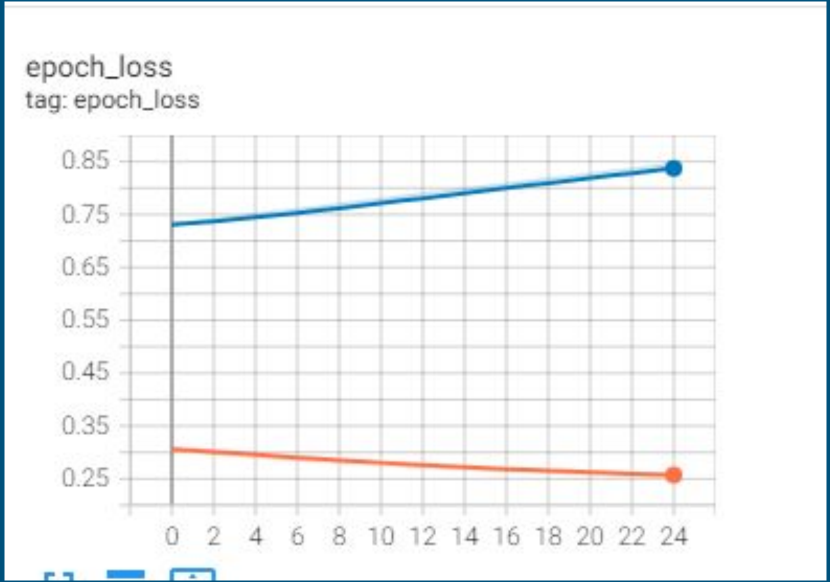
If the **training accuracy increases**(positive slope) while the **validation accuracy steadily decreases**(negative slope) then a situation of over fitting may have occurred. Time to stop Training.

Which means stop the number of epoch at that point.

What was the problem with my code last time?

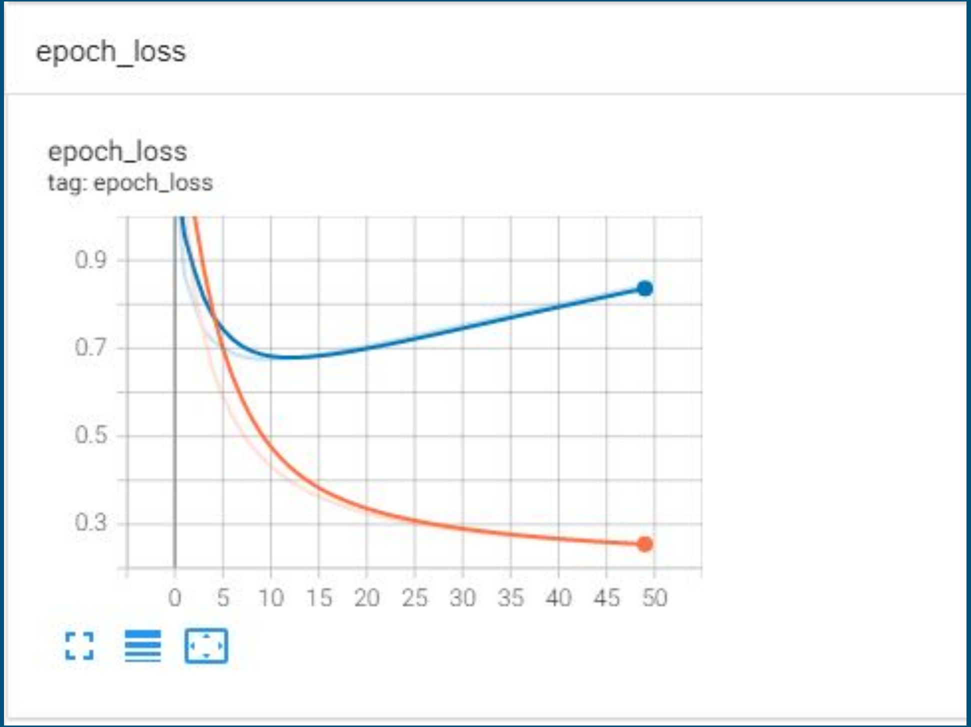
Hence this result..





How it has changed?

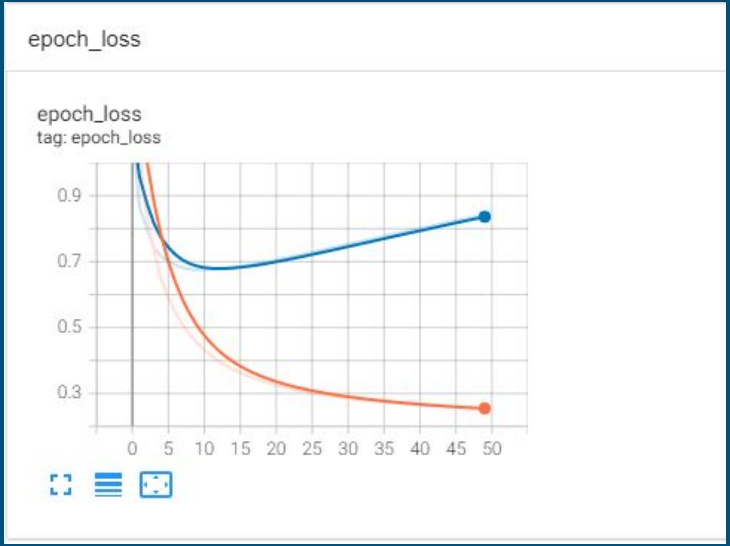
- The best learning rate ranges from 0.1 to 0.01
- The epoch number increased to 50 gives the following result.
- Since smaller learning rates require more training epochs.

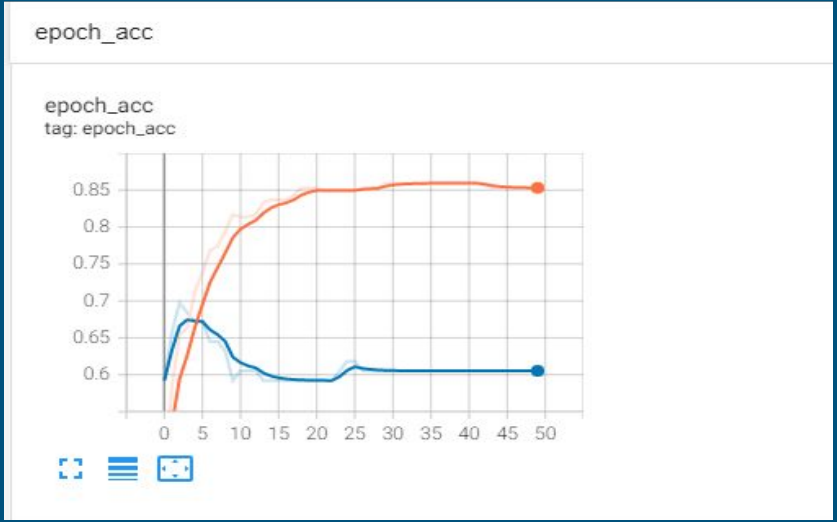


How it has changed?

EPOCH NUMBER INCREASED TO 50

● train
● test

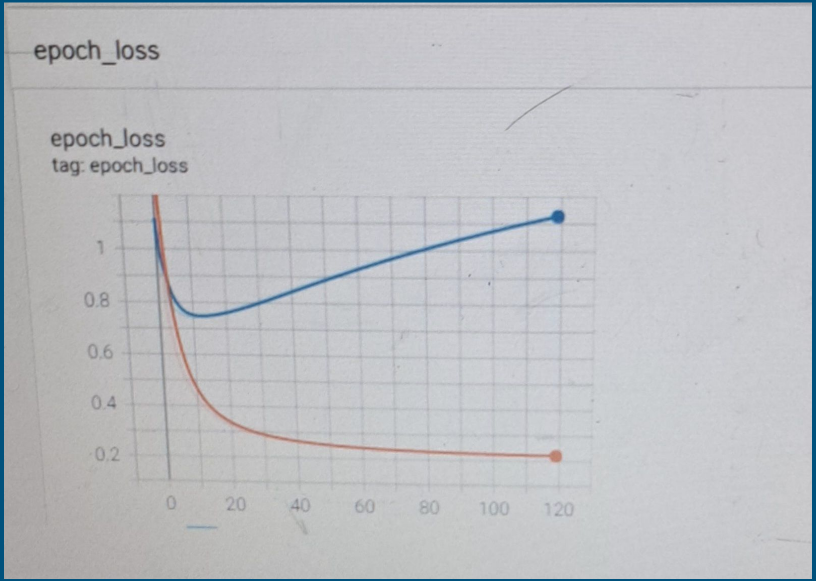


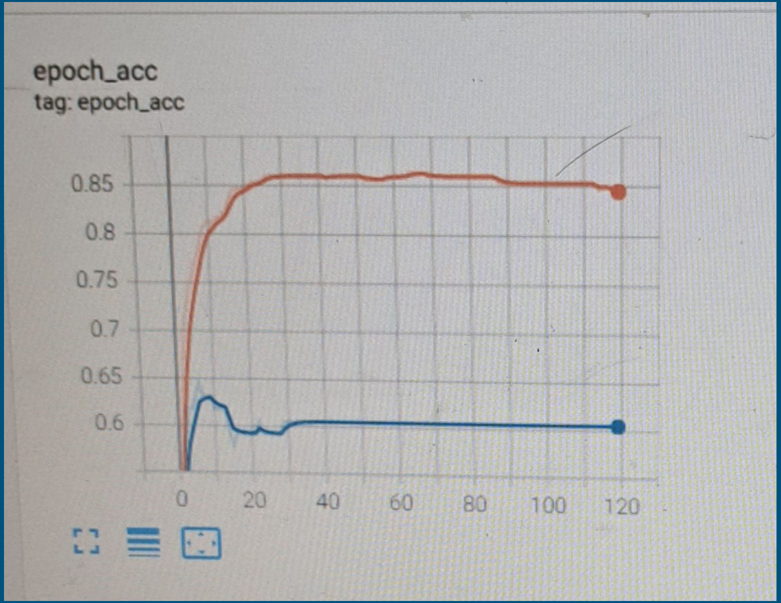


How it has changed?

EPOCH INCREASED TO 120

🟠	train
🔵	test





Conclusion...Is the problem solved.

The change in epoch number has helped the model to learn better but a few other things still need to be changed.

Regularization

Adding restrictions during training

Conservative training

We add some restrictions during training, so that the new model after training is not too different from the old model. This restriction is actually Regularization. We hope that the output of the new model and the old model will be the same.

Why do we need to use the conservative method

There are the possibilities that the model performs accurately on training data but fails to perform well on test data and also produces high error due to several factors such as collinearity, bias-variance impact and over modeling on train data.

Regularization

The features are estimated using coefficients while modelling. Also, if the estimates can be restricted, or shrinked or regularized towards zero, then the impact of insignificant features might be reduced and would prevent models from high variance with a stable fit.

It avoids overfitting by panelizing the regression coefficients of high value. More specifically, It decreases the parameters and shrinks (simplifies) the model. This more streamlined model will aptly perform more efficiently while making predictions..

Examples of regularization

K-means: Restricting the segments for avoiding redundant groups.

Neural networks: Confining the complexity (weights) of a model.

Random Forest: Reducing the depth of tree and branches (new features)

Types of regularization

L1, L2 and dropout regularization

L1 regularization: It adds an L1 penalty that is equal to the absolute value of the magnitude of coefficient, or simply restricting the size of coefficients. For example, Lasso regression implements this method.

L2 Regularization: It adds an L2 penalty which is equal to the square of the magnitude of coefficients. For example, Ridge regression and SVM implement this method.

Elastic Net: When L1 and L2 regularization combine together, it becomes the elastic net method, it adds a hyperparameter.

Using normalization and regularization at the same time

Normalization is used to speed up optimization and regularization is used to control overfitting.So the two can be used together since they don’t affect each other.

Layer weight regularizers

Regularizers allow you to apply penalties on layer parameters or layer activity during optimization. These penalties are summed into the loss function that the network optimizes.

Regularization penalties are applied on a per-layer basis. The exact API will depend on the layer, but many layers (e.g. Dense, Conv1D, Conv2D and Conv3D) have a unified API.

These layers expose 3 keyword arguments:

kernel_regularizer: Regularizer to apply a penalty on the layer's kernel

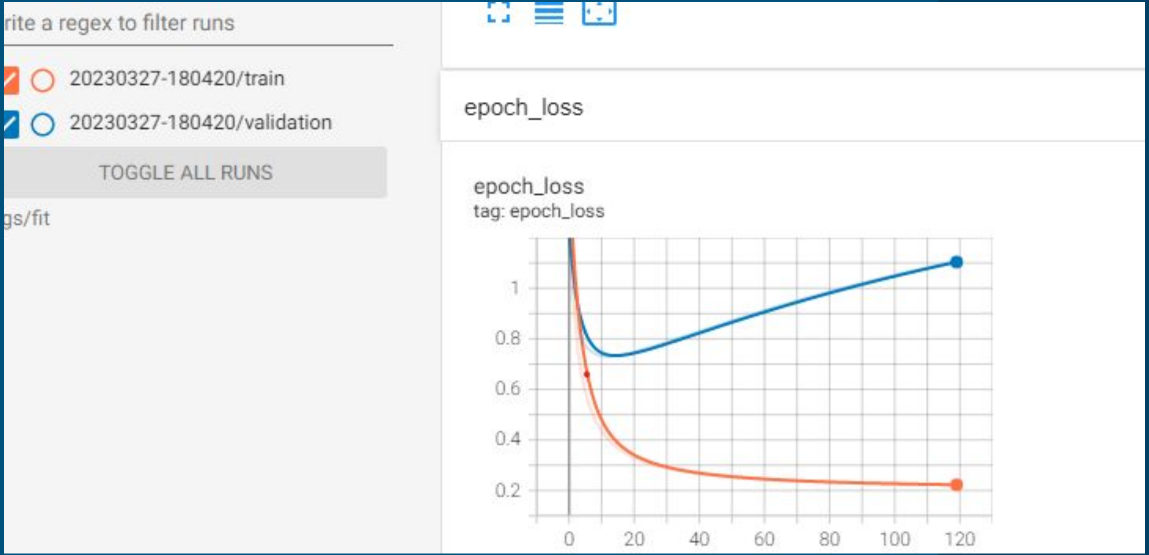
bias_regularizer: Regularizer to apply a penalty on the layer's bias

activity_regularizer: Regularizer to apply a penalty on the layer's output

So I have used L1,L2 and L1L2.As well as the kernel and activity regularizers.

I have noticed that even the penalty value also affect the results so I tried to see what is the best value using L1 at different values.

Results while using L1(2.0)regularization


rite a regex to filter runs

20230327-180420/train

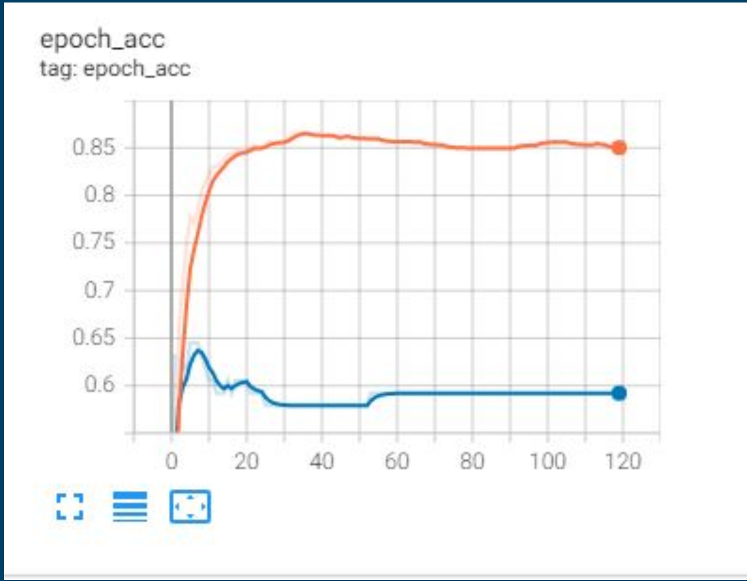
20230327-180420/validation

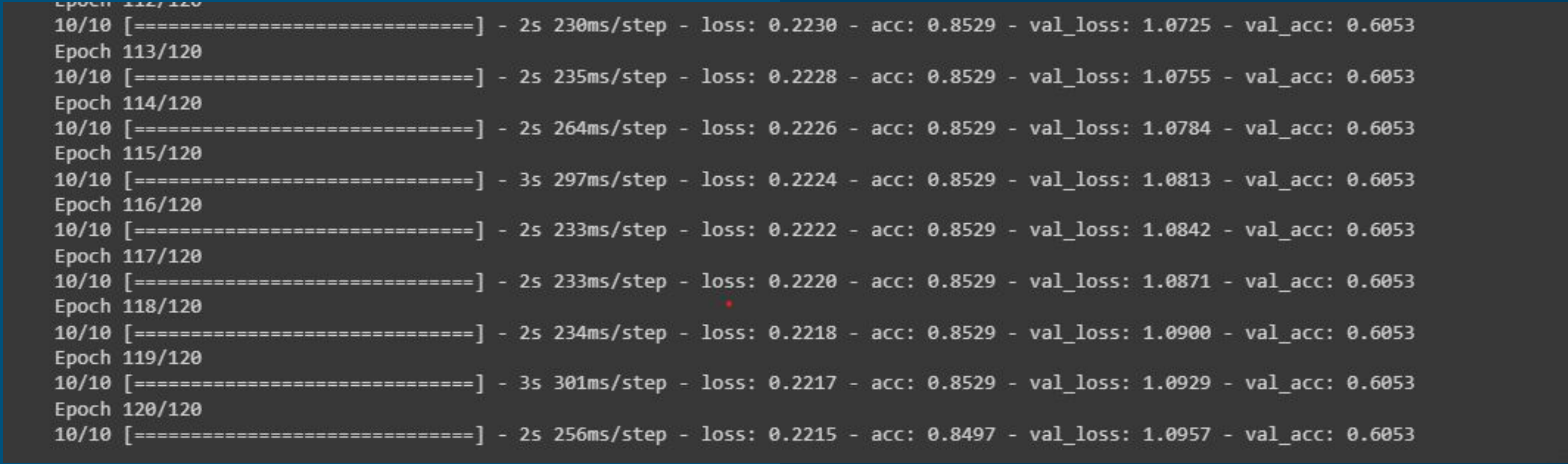
TOGGLE ALL RUNS

gs/fit

epoch_loss

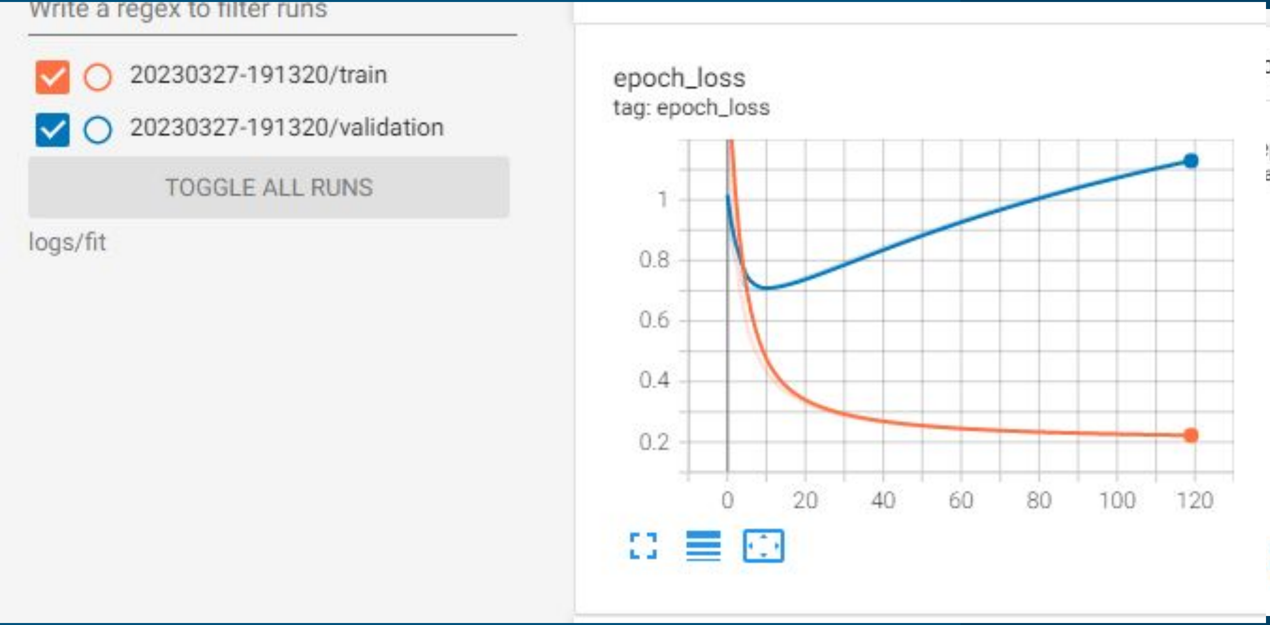
epoch_loss
tag: epoch_loss

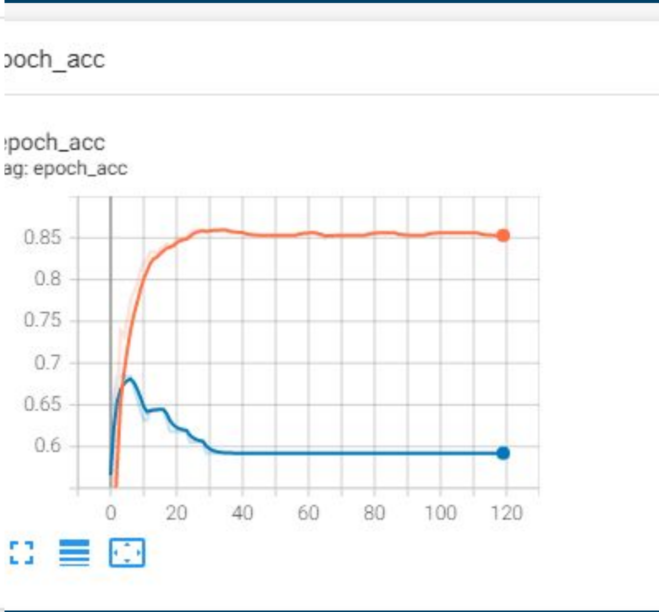

epoch_acc
tag: epoch_acc



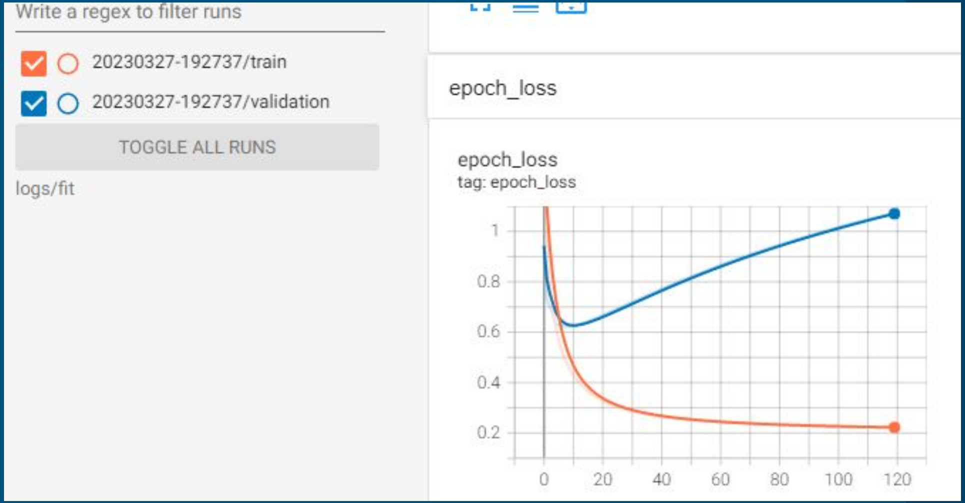
```
10/10 [==============================] - 2s 230ms/step - loss: 0.2230 - acc: 0.8529 - val_loss: 1.0725 - val_acc: 0.6053
Epoch 113/120
10/10 [==============================] - 2s 235ms/step - loss: 0.2228 - acc: 0.8529 - val_loss: 1.0755 - val_acc: 0.6053
Epoch 114/120
10/10 [==============================] - 2s 264ms/step - loss: 0.2226 - acc: 0.8529 - val_loss: 1.0784 - val_acc: 0.6053
Epoch 115/120
10/10 [==============================] - 3s 297ms/step - loss: 0.2224 - acc: 0.8529 - val_loss: 1.0813 - val_acc: 0.6053
Epoch 116/120
10/10 [==============================] - 2s 233ms/step - loss: 0.2222 - acc: 0.8529 - val_loss: 1.0842 - val_acc: 0.6053
Epoch 117/120
10/10 [==============================] - 2s 233ms/step - loss: 0.2220 - acc: 0.8529 - val_loss: 1.0871 - val_acc: 0.6053
Epoch 118/120
10/10 [==============================] - 2s 234ms/step - loss: 0.2218 - acc: 0.8529 - val_loss: 1.0900 - val_acc: 0.6053
Epoch 119/120
10/10 [==============================] - 3s 301ms/step - loss: 0.2217 - acc: 0.8529 - val_loss: 1.0929 - val_acc: 0.6053
Epoch 120/120
10/10 [==============================] - 2s 256ms/step - loss: 0.2215 - acc: 0.8497 - val_loss: 1.0957 - val_acc: 0.6053
```

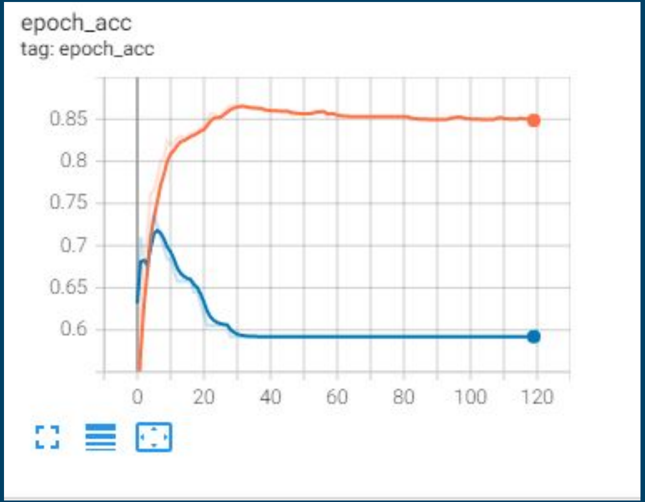
Results while using L1(0.01)regularization



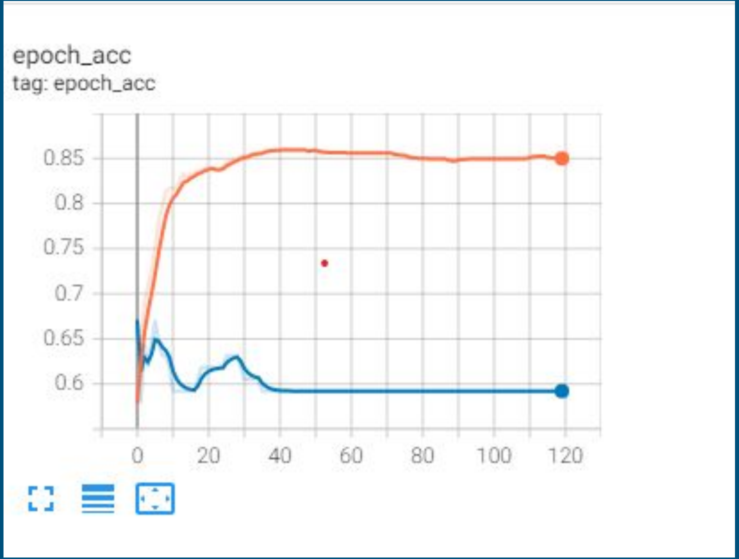


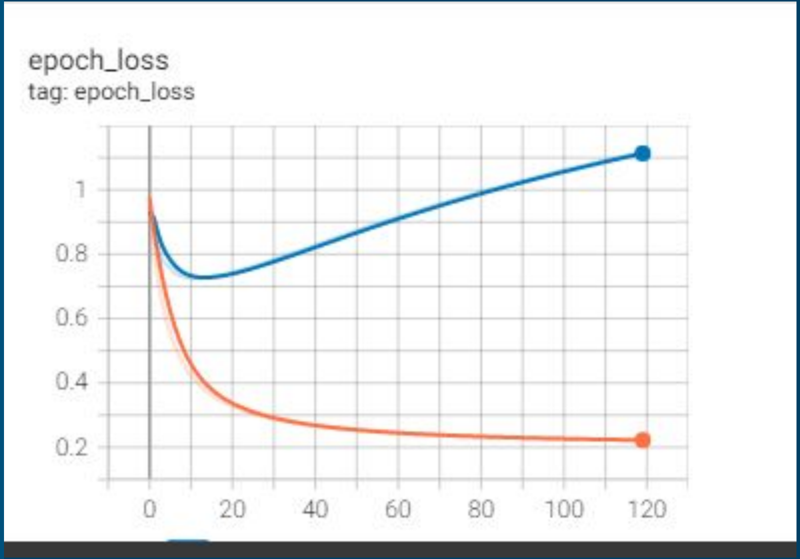
Results while using L1(0.001)regularization

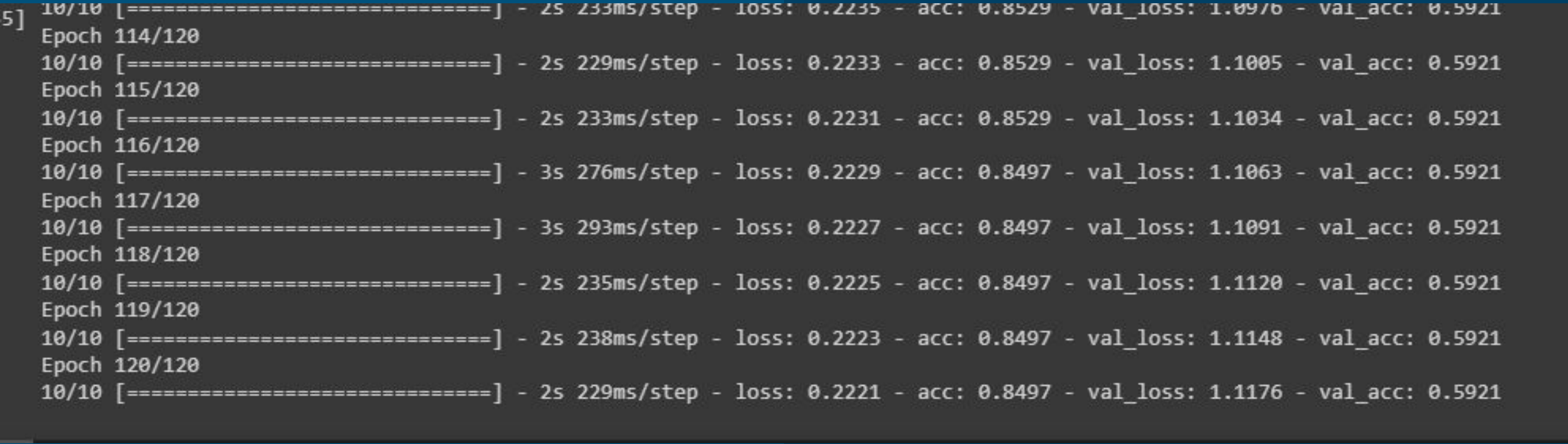




Results while using L2 regularization

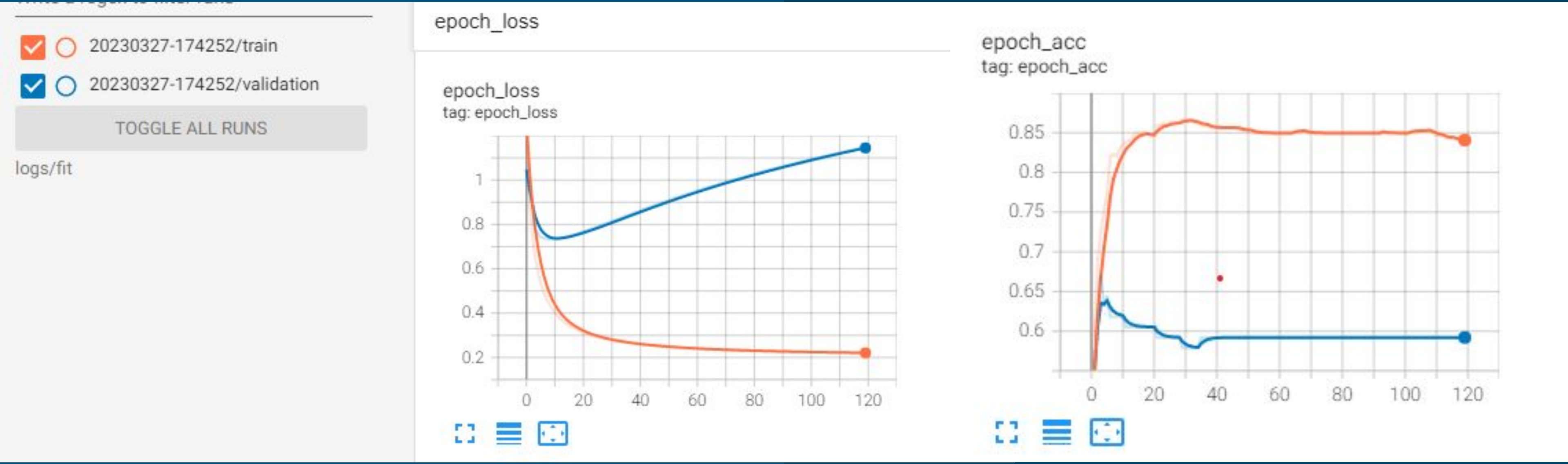


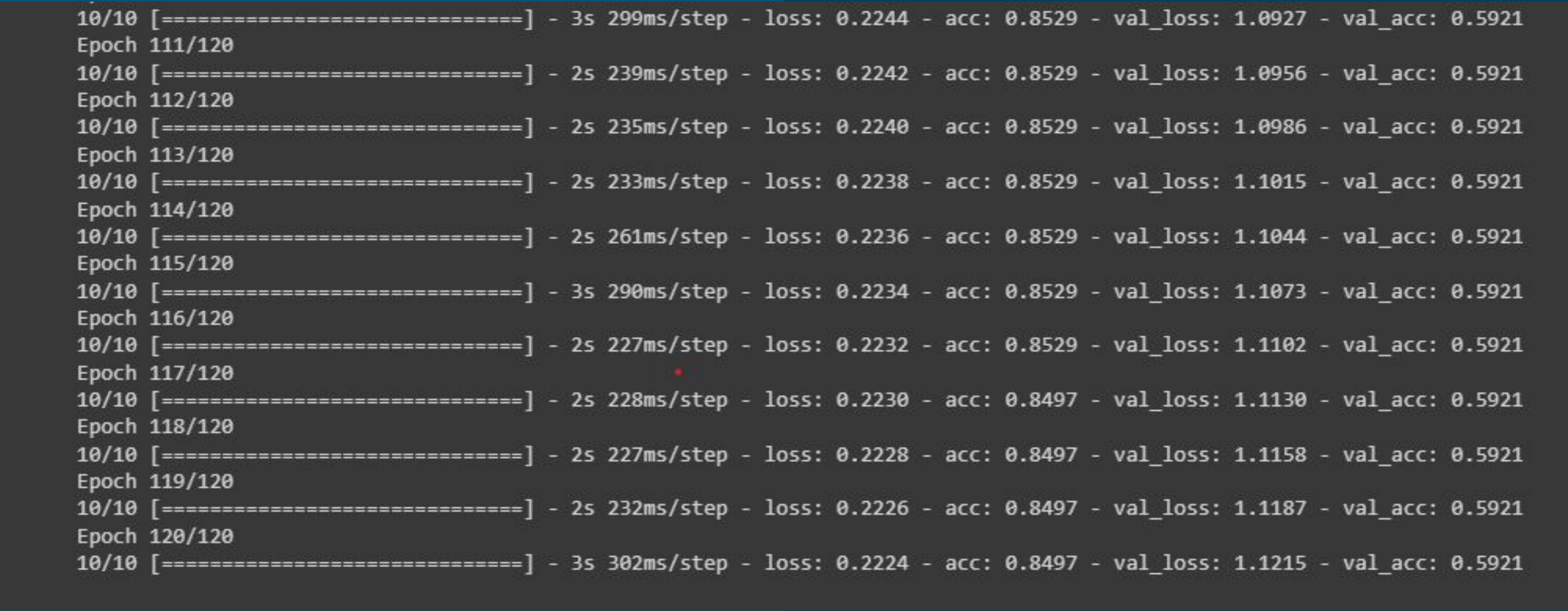




```
5]  10/10 [==============================] - 2s 233ms/step - loss: 0.2235 - acc: 0.8529 - val_loss: 1.0976 - val_acc: 0.5921
    Epoch 114/120
    10/10 [==============================] - 2s 229ms/step - loss: 0.2233 - acc: 0.8529 - val_loss: 1.1005 - val_acc: 0.5921
    Epoch 115/120
    10/10 [==============================] - 2s 233ms/step - loss: 0.2231 - acc: 0.8529 - val_loss: 1.1034 - val_acc: 0.5921
    Epoch 116/120
    10/10 [==============================] - 3s 276ms/step - loss: 0.2229 - acc: 0.8497 - val_loss: 1.1063 - val_acc: 0.5921
    Epoch 117/120
    10/10 [==============================] - 3s 293ms/step - loss: 0.2227 - acc: 0.8497 - val_loss: 1.1091 - val_acc: 0.5921
    Epoch 118/120
    10/10 [==============================] - 2s 235ms/step - loss: 0.2225 - acc: 0.8497 - val_loss: 1.1120 - val_acc: 0.5921
    Epoch 119/120
    10/10 [==============================] - 2s 238ms/step - loss: 0.2223 - acc: 0.8497 - val_loss: 1.1148 - val_acc: 0.5921
    Epoch 120/120
    10/10 [==============================] - 2s 229ms/step - loss: 0.2221 - acc: 0.8497 - val_loss: 1.1176 - val_acc: 0.5921
```

Results while using L1L2 regularization





```
10/10 [==============================] - 3s 299ms/step - loss: 0.2244 - acc: 0.8529 - val_loss: 1.0927 - val_acc: 0.5921
Epoch 111/120
10/10 [==============================] - 2s 239ms/step - loss: 0.2242 - acc: 0.8529 - val_loss: 1.0956 - val_acc: 0.5921
Epoch 112/120
10/10 [==============================] - 2s 235ms/step - loss: 0.2240 - acc: 0.8529 - val_loss: 1.0986 - val_acc: 0.5921
Epoch 113/120
10/10 [==============================] - 2s 233ms/step - loss: 0.2238 - acc: 0.8529 - val_loss: 1.1015 - val_acc: 0.5921
Epoch 114/120
10/10 [==============================] - 2s 261ms/step - loss: 0.2236 - acc: 0.8529 - val_loss: 1.1044 - val_acc: 0.5921
Epoch 115/120
10/10 [==============================] - 3s 290ms/step - loss: 0.2234 - acc: 0.8529 - val_loss: 1.1073 - val_acc: 0.5921
Epoch 116/120
10/10 [==============================] - 2s 227ms/step - loss: 0.2232 - acc: 0.8529 - val_loss: 1.1102 - val_acc: 0.5921
Epoch 117/120
10/10 [==============================] - 2s 228ms/step - loss: 0.2230 - acc: 0.8497 - val_loss: 1.1130 - val_acc: 0.5921
Epoch 118/120
10/10 [==============================] - 2s 227ms/step - loss: 0.2228 - acc: 0.8497 - val_loss: 1.1158 - val_acc: 0.5921
Epoch 119/120
10/10 [==============================] - 2s 232ms/step - loss: 0.2226 - acc: 0.8497 - val_loss: 1.1187 - val_acc: 0.5921
Epoch 120/120
10/10 [==============================] - 3s 302ms/step - loss: 0.2224 - acc: 0.8497 - val_loss: 1.1215 - val_acc: 0.5921
```

CONCLUSION

The accuracy is increased up to a third of the epoch number then drops again.The best performing has been L1 at 0.001 penalty value(lowering the penalty value increases accuracy in the test data).However i still need to try L2 and L1L2 at low penalty values,as well as the dropout regularizer.