Ronaldlee Ejalu

Kaggle UserId: Ronald_Novi

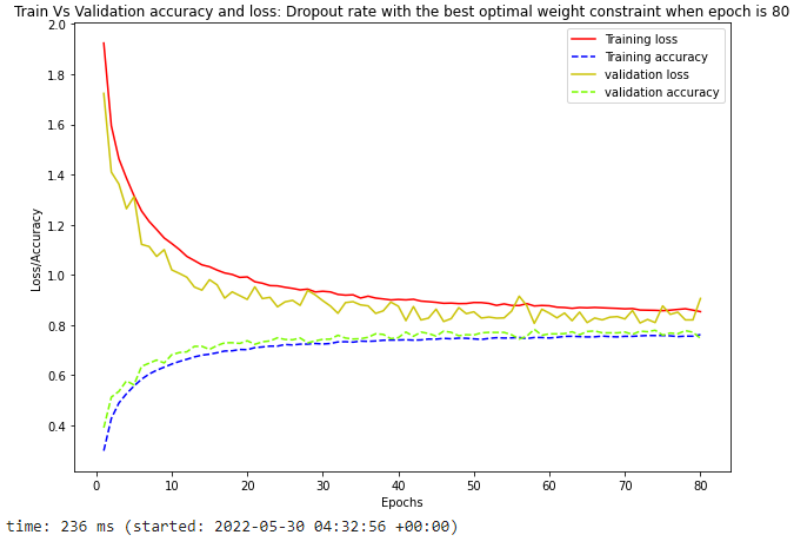
Private Score: 0.69965

Public Score: 0.71610

CSC 578 : 901

CSC 578 Homework 7:

A. Description of the best model:
After going through the process to do Random SearchCV, I found out that the best optimal epoch was 80 instead of 30 as I had previously guessed. Additionally, the best batch size value was 32 instead of 64. Also, I came across some research by Jason Brownlee talking about how you could use dropout rate with the weight constraint to limit over fitting. After finding the best parameter values namely Epoch, batch size, learning rate, which didn't change. I used these best parameters to run the best model and my scores improved because my validation loss had actually gone down as shown below in the charts.



time: 236 ms (started: 2022-05-30 04:32:56 +00:00)

We see that validation accuracy is slighly higher than the validation accuracy. At some epochs, they touch each other., but over this is the best model with a good validation accuracy and a low validation loss and it is the oen I used for my final submission.

B. Description of my journey of hyperparameter tuning:
Using 30 complete epochs, I begun experimenting the following parameters:

1. Mini-batch size:

These are the number of training examples at a time the network for training. The choice of the mini-batch size should relatively be independent of other hyper-parameters. Too small of a

size , you don't take full advantage of the benefits of the matrix libraries optimized for Hardware like how Michael Nielsen mentions in the NNDL book and too large, you are simply not updating your weights often enough. Using a for loop I built a variety of network models with a list of batch sizes of 8, 16, 32, 64 and 100.

When the mini-batch size is small, there is an increase in the processing time of the model to fit the data, and it decreases when the size is big as shown below in the table.

	Mini-batch-size	Time in seconds
0	8	482.807309
1	16	252.049198
2	32	141.760658
3	64	78.349052
4	100	50.542675

A minimum batch size of 100 took approximately 50 seconds to run and comparing it with other batch sizes it doesn't less overfits the training data as shown below:

Furthermore, it had the highest validation accuracy and less validation loss as shown below:

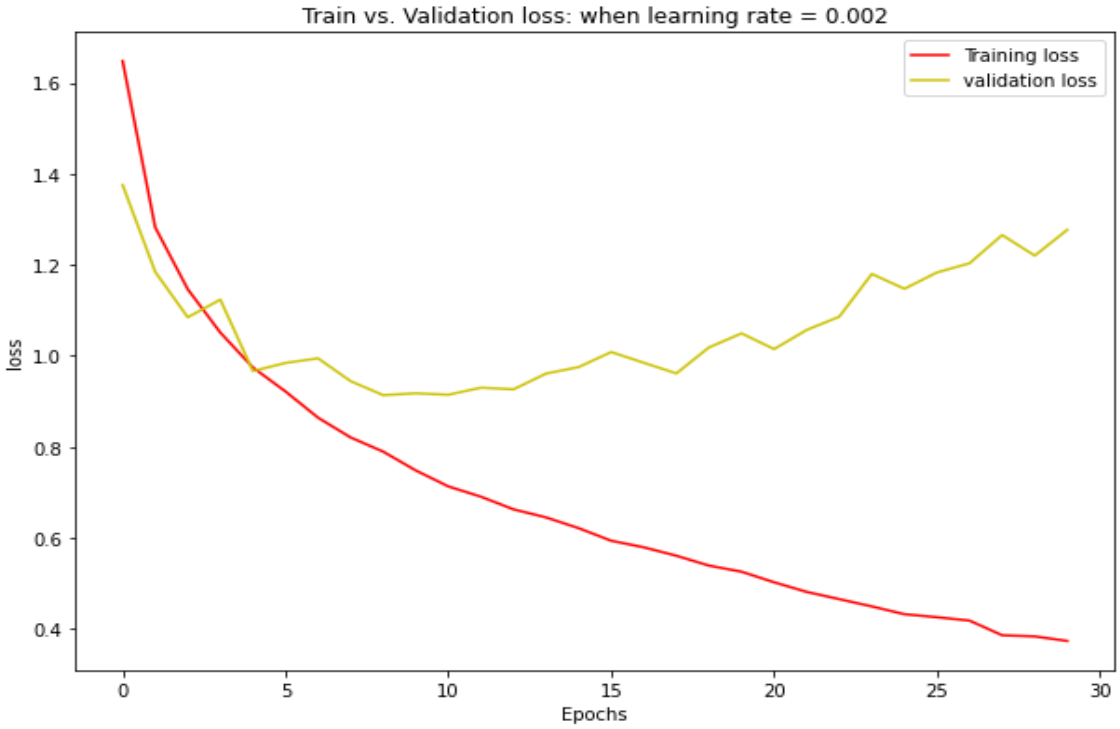
	Mini-batch-size	val_loss
0	8	1.952374
1	16	2.007511
2	32	1.507954
3	64	1.547393
4	100	1.228263

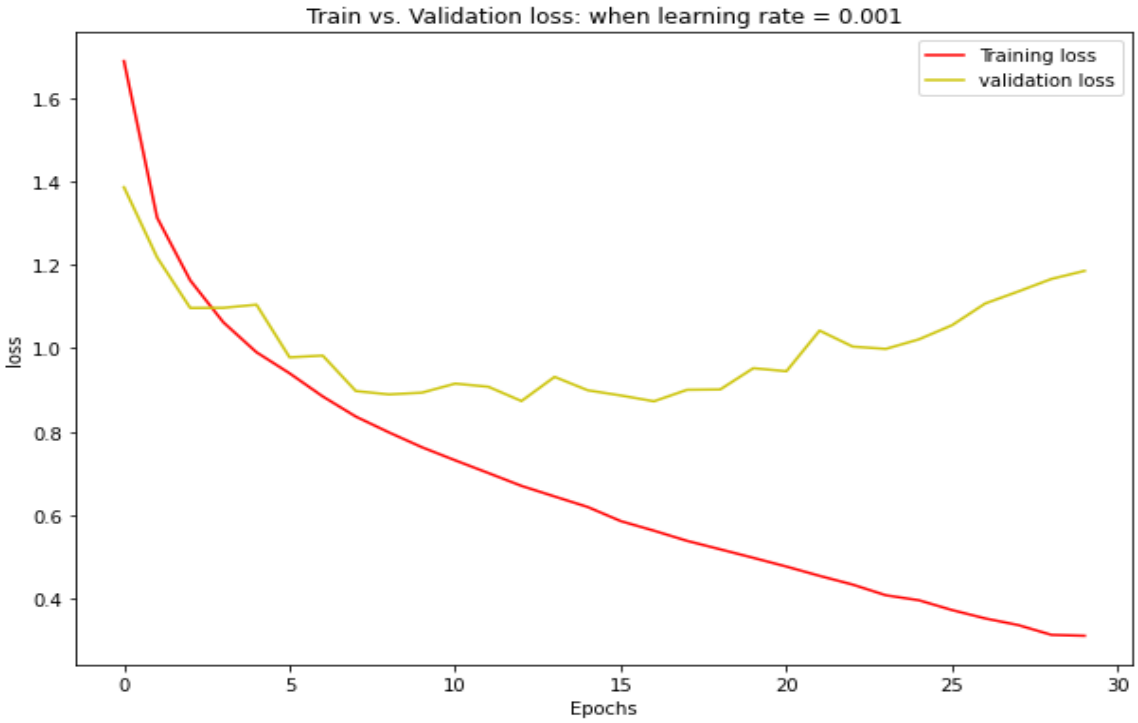
	Mini-batch-size	val_accuracy_L
0	8	0.6496
1	16	0.6542
2	32	0.6844
3	64	0.7024
4	100	0.6991

2. Learning rate:

This was my second hyperparameter I optimized. Using the best epoch of 100 within a for loop, I ran different networks, compiling the model with a list of learning rates: 0.5, 0.25, 0.025, 0.002, and 0.001 and fitted the training data with the network model within each iteration. The learning rates of 0.5, 0.25, 0.025 performed poorly.

Comparing 0.001 with 0.002 as shown by the Train vs. Validation loss ;line charts below:





We observe that the learning rate of 0.001 is the best value since the network experiences less overfitting when compared to 0.002.

Additionally, we see that 0.001 has the highest validation accuracy and less validation loss as shown below:

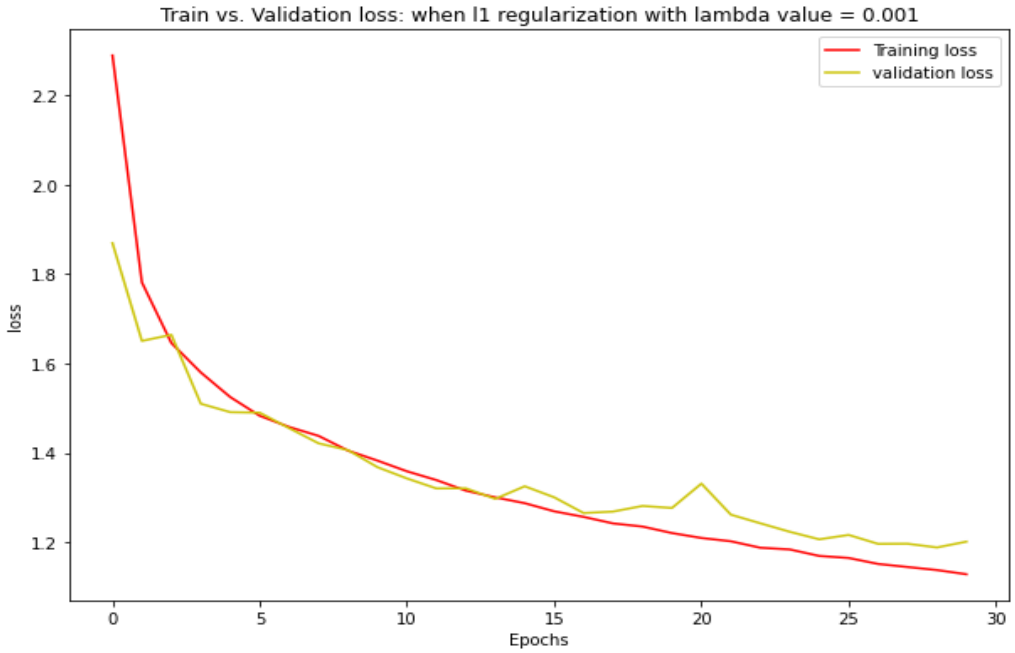
	learningRate	val_loss
0	0.500	2.313286
1	0.250	2.314390
2	0.025	2.303901
3	0.002	1.276839
4	0.001	1.185140

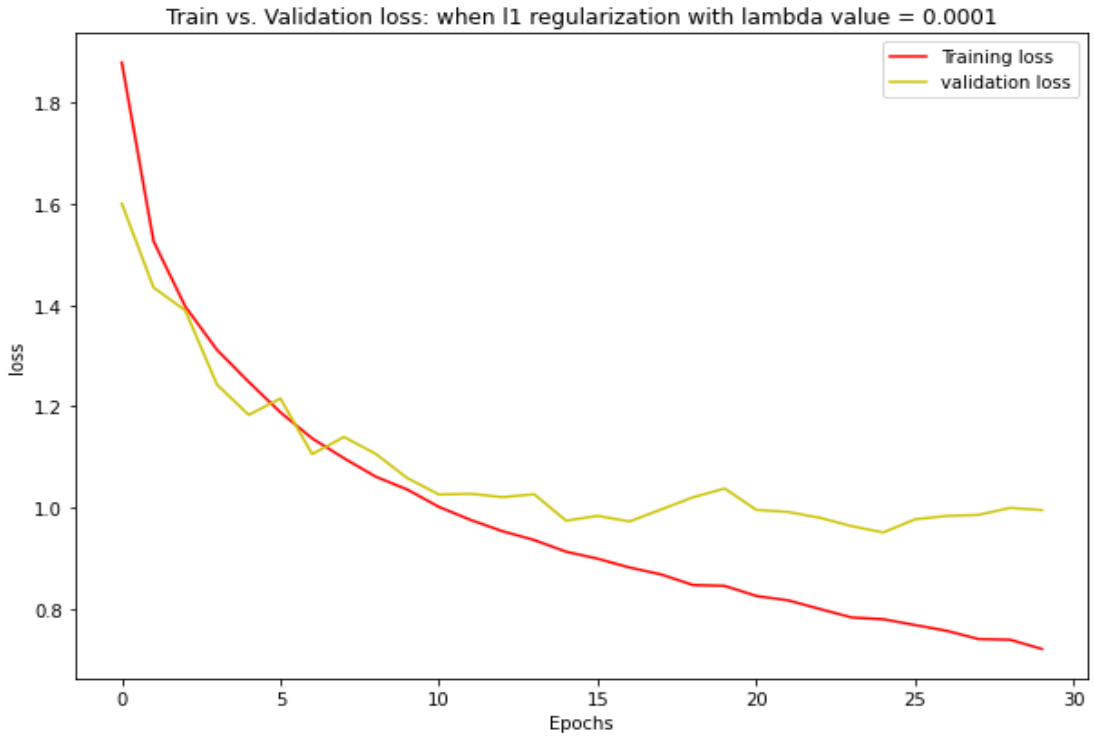
	learningRate	val_accuracy_L
0	0.500	0.1000
1	0.250	0.1000
2	0.025	0.1000
3	0.002	0.6768
4	0.001	0.7056

3. Regularization methods

Using a list of values of 1, 0.1, 0.01, 0.001 and 0.0001, I used both l2 and l1 regularization. Using a for loop, I built various networks where I compiled and fitted the network model with the training data.

With l1 regularization, values of 1, 0.1 and 0.01 performed poorly. 0.001 was the best lambda value when compared with 0.0001 because it has less overfitting as shown below in the line charts:





However, 0.0001 had the highest validation accuracy and low validation loss as shown below:

	lambda	val_loss
0	1.0000	9.268972
1	0.1000	2.993785
2	0.0100	1.892821
3	0.0010	1.200979
4	0.0001	0.995192

	lambda	val_accuracy_L
0	1.0000	0.1000
1	0.1000	0.1000
2	0.0100	0.3605
3	0.0010	0.6350
4	0.0001	0.7056

With l2 regularization, values of 1, and 0.1 performed poorly. 0.001 was the best lambda value when compared with 0.0001 and 0.1 because it had the highest validation accuracy and low validation loss as shown below:

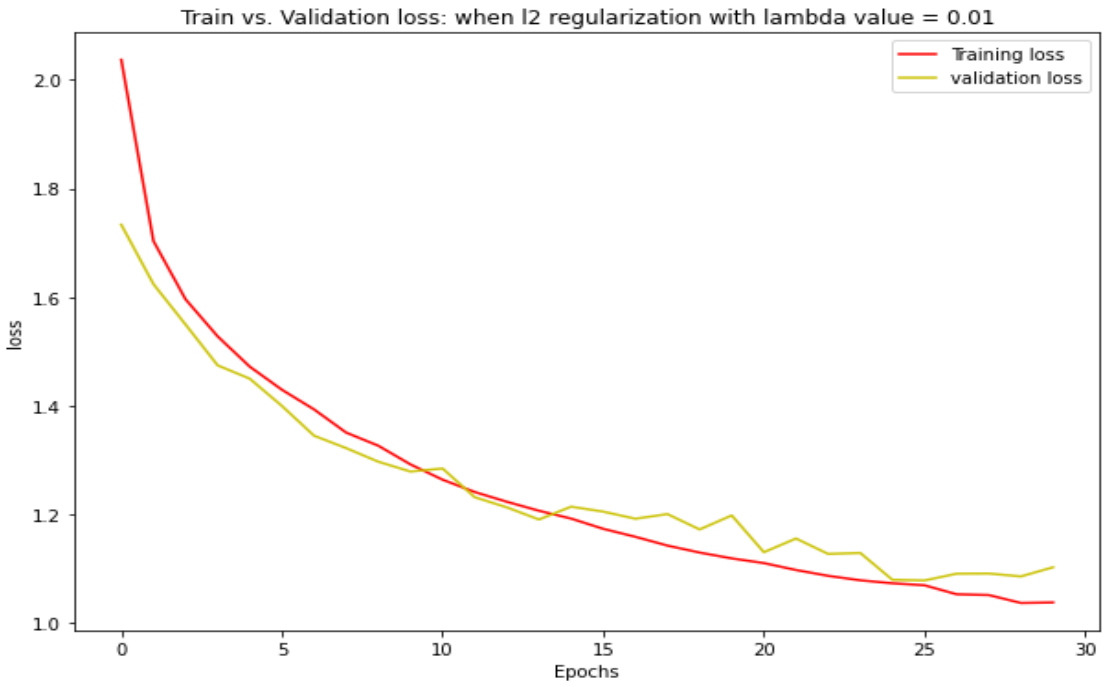
```
val_loss_df
```

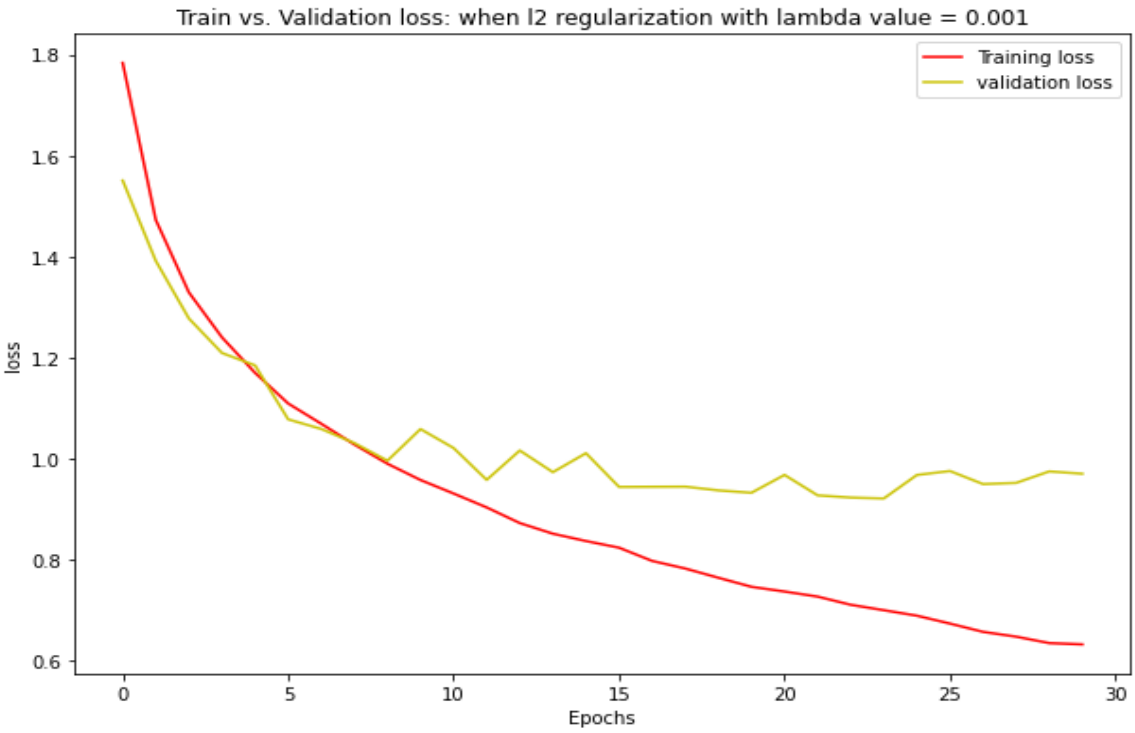
	lambda	val_loss
0	1.0000	9.116474
1	0.1000	2.236322
2	0.0100	1.234797
3	0.0010	1.022068
4	0.0001	1.097710

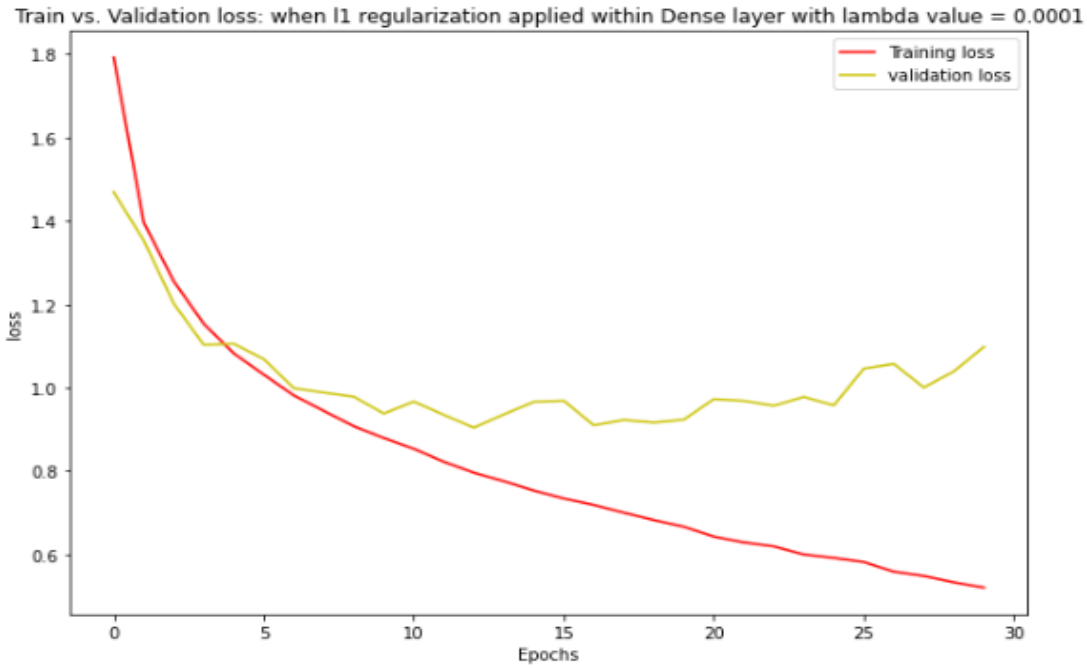
```
val_accuracy_df
```

	lambda	val_accuracy_L
0	1.0000	0.3782
1	0.1000	0.4854
2	0.0100	0.6252
3	0.0010	0.6945
4	0.0001	0.7072

0.01 had the less overfitting when compared to 0.001 and 0.0001 as shown below:







Comparing both l2 and l1 regularization lambda values, 0.001 is the best value with less overfitting.

Furthermore,I explored different values of l1 and l2 regularization within the Convolutional and dense layers and the value of lambda of 0.001 with l2 regularization applied within the Convoluti onal layer is the best lambda value because it experiences less overfitting when the network star t fitting the training data to when it finishes.

4. Dropout

Dropout randomly selects nodes and removes them from the network when the model is training to limit overfitting and increase the level of generalization for the model. I explored different values of 0.2, 0.3, 0.4, and 0.5. Using a l2 regularization value of 0.001, learning rate of 0.001 and a mini-batch size of 100, we modified the network with the list of 20, 30, 40 and 50 percent values where those percent of the hidden neurons in the network are randomly deleted in a for loop as the multiple networks were built.

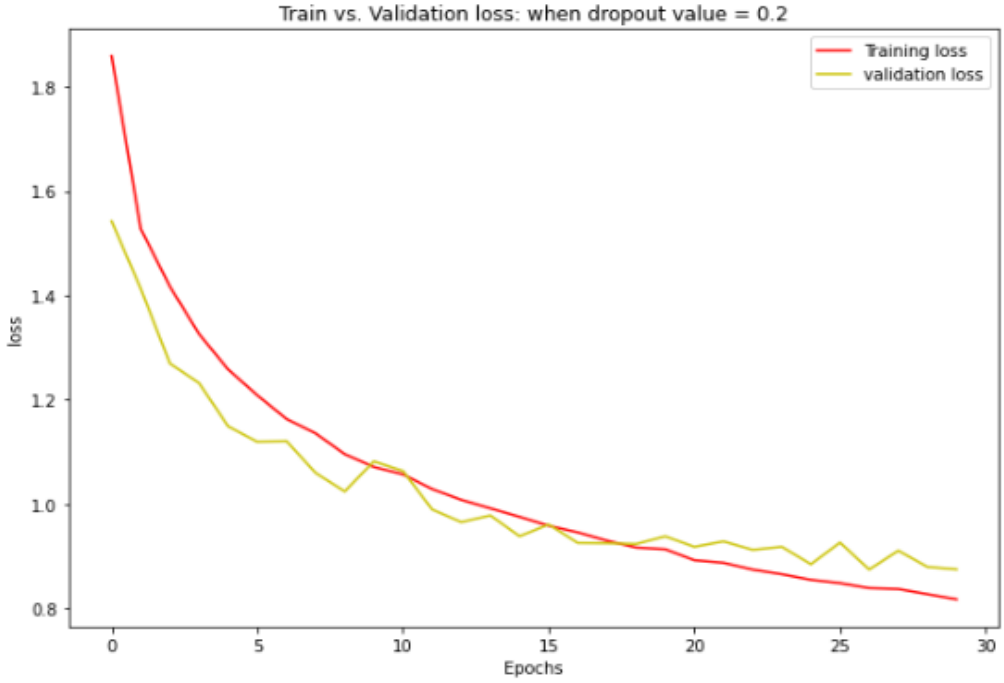
With comparison of other values explored, I discovered that 0.2 had the highest validation accuracy and lowest validation loss with less overfitting as shown below:

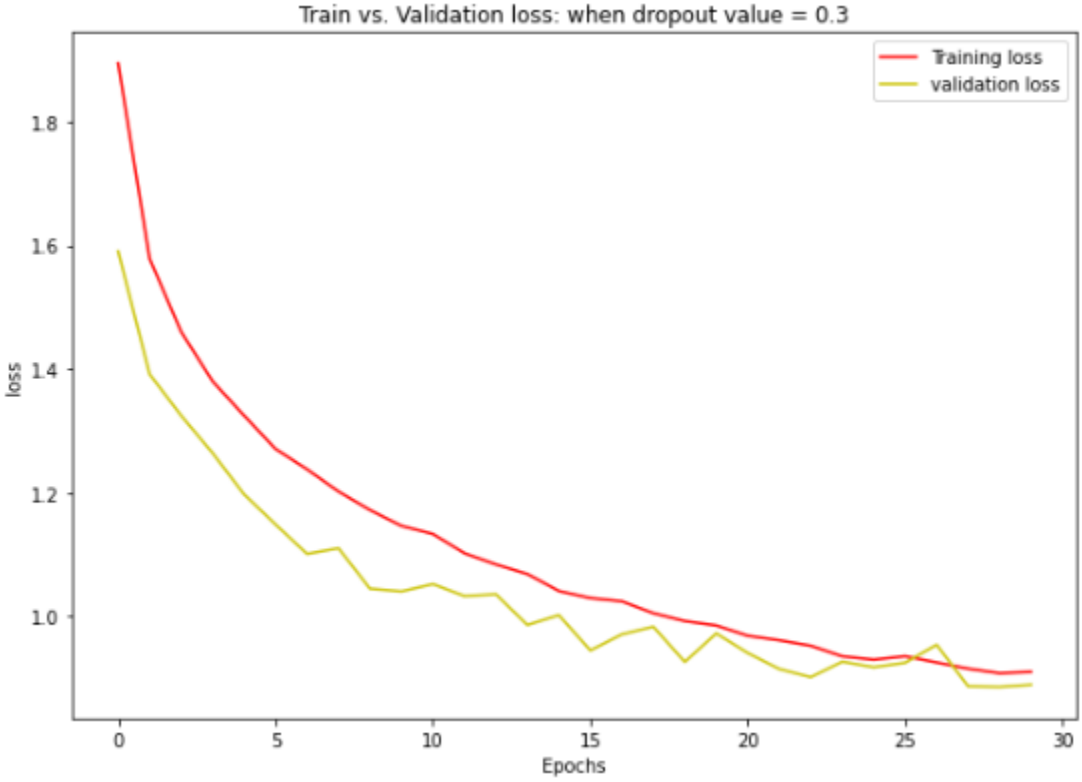
```
val_loss_df
```

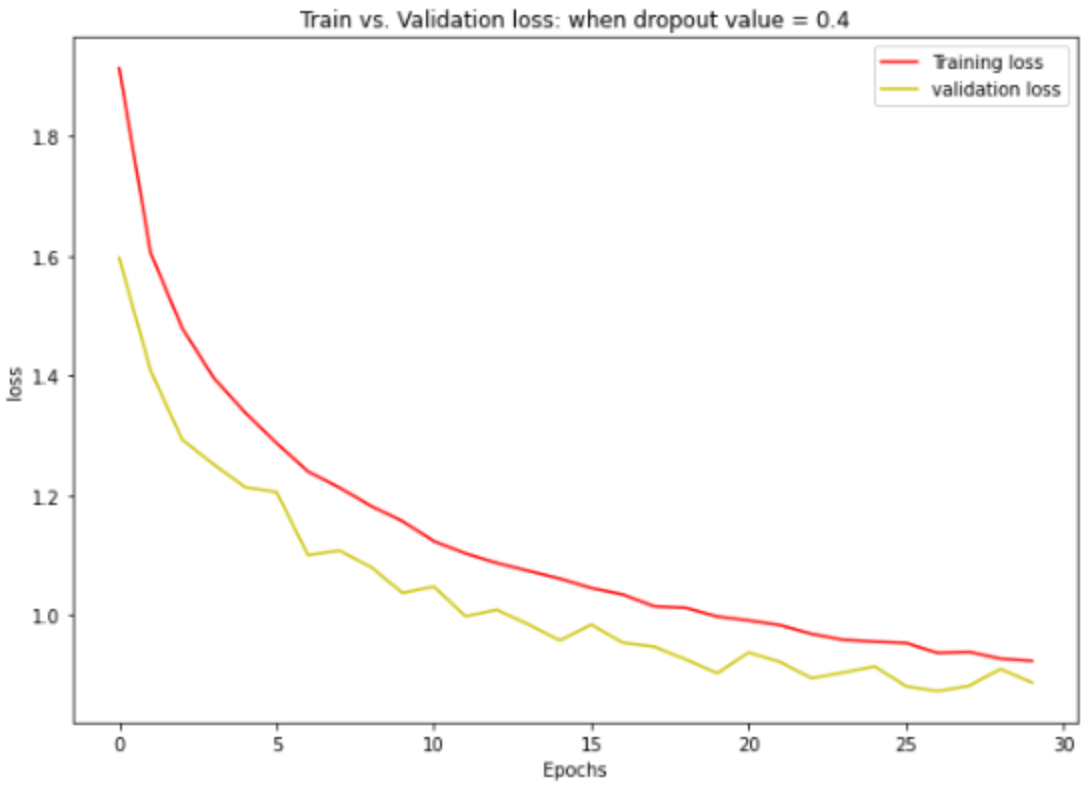
	dropout	val_loss
0	0.2	0.875205
1	0.3	0.889680
2	0.4	0.886582
3	0.5	0.939425

```
val_accuracy_df
```

	dropout	val_accuracy_L
0	0.2	0.7318
1	0.3	0.7263
2	0.4	0.7312
3	0.5	0.7036


Train vs. Validation loss: when dropout value = 0.2
Training loss
validation loss
loss
Epochs
1.8
1.6
1.4
1.2
1.0
0.8
0
5
10
15
20
25
30

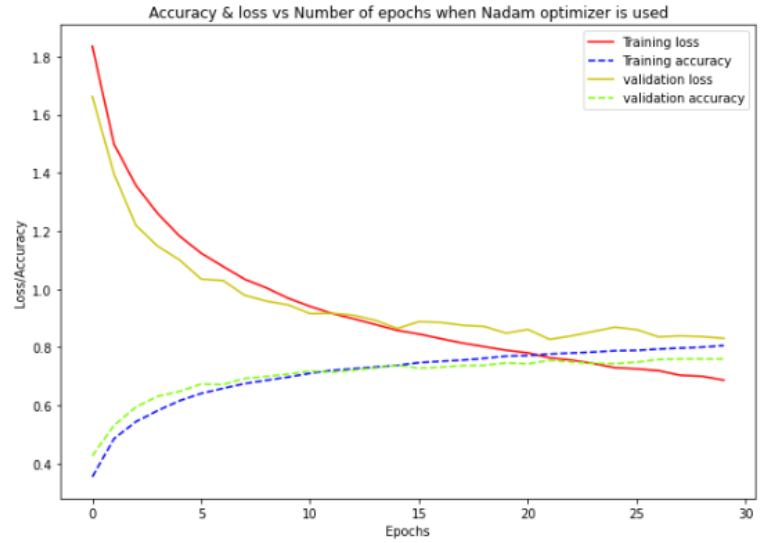

Train vs. Validation loss: when dropout value = 0.3
Training loss
validation loss
loss
Epochs
1.8
1.6
1.4
1.2
1.0
0
5
10
15
20
25
30



5. Optimizers

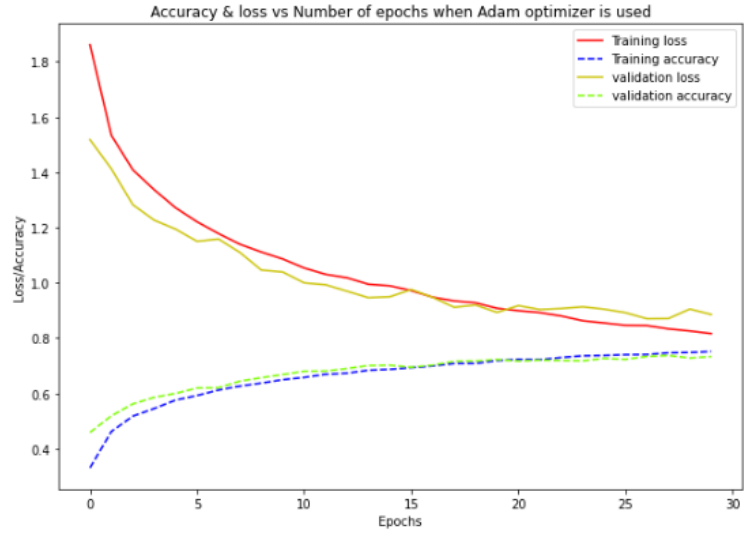
I built and compiled the two networks with two different optimizers namely Adam and Adam, and fitted the training data on the two different network models.

The results below:



time: 258 ms (started: 2022-05-28 15:56:37 +00:00)

We notice that at training epoch 27 is where there is validation loss is at its lowest. Also we notice at epoch 27 the gap between training and validation accuracy is small, which means that there is less overfitting.



time: 235 ms (started: 2022-05-28 15:57:33 +00:00)

We notice that at training epoch 27 is where validation loss is at its lowest. Also we notice at epoch 27 the gap between training and validation accuracy is small, which means that there is less overfitting.

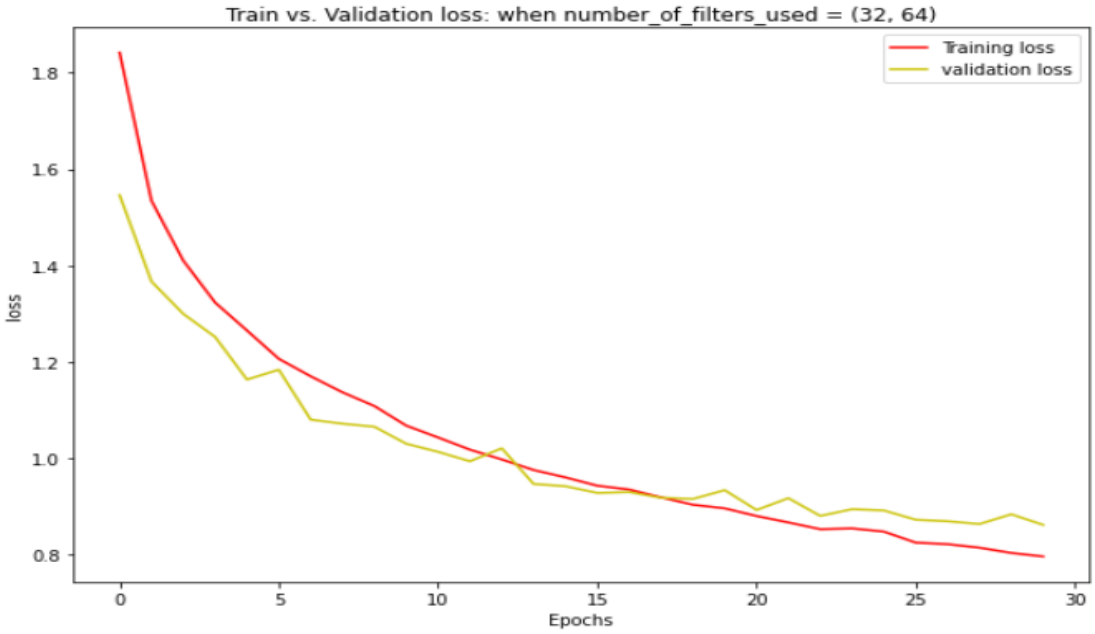
show that the Adam optimizer performs better since there is less overfitting as we just witnessed from the above charts and going forward, I will be using the Adam optimizer.

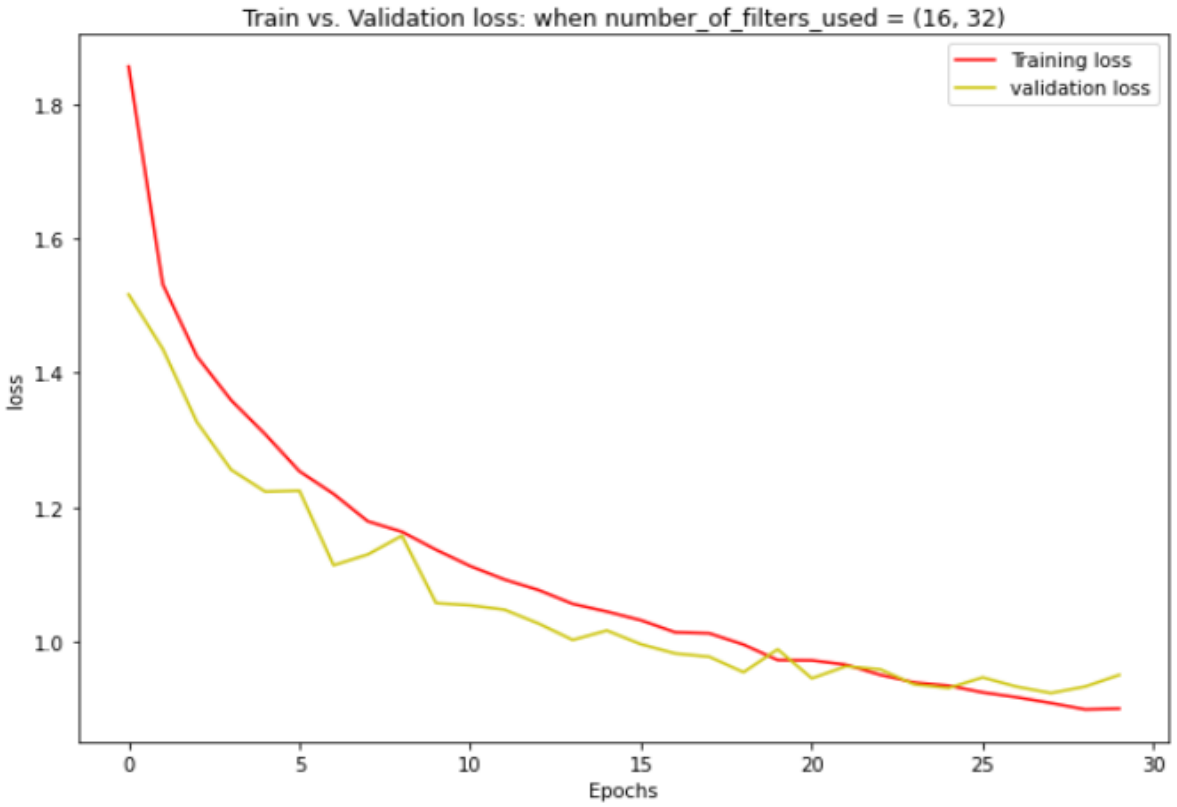
6. Number of filters

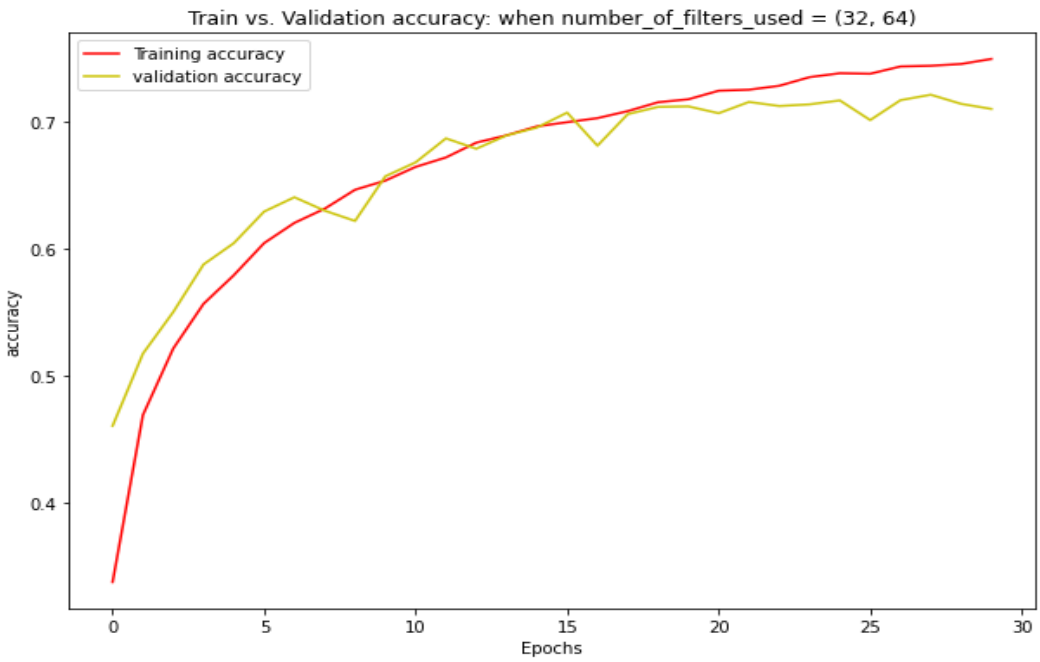
Using a list of tuples of pairs of numbers namely, (8,16),(16,32), (24, 48), (32, 64), (40, 80), (48, 96) with the best already found hyper parameters above, I created a for loop where I was

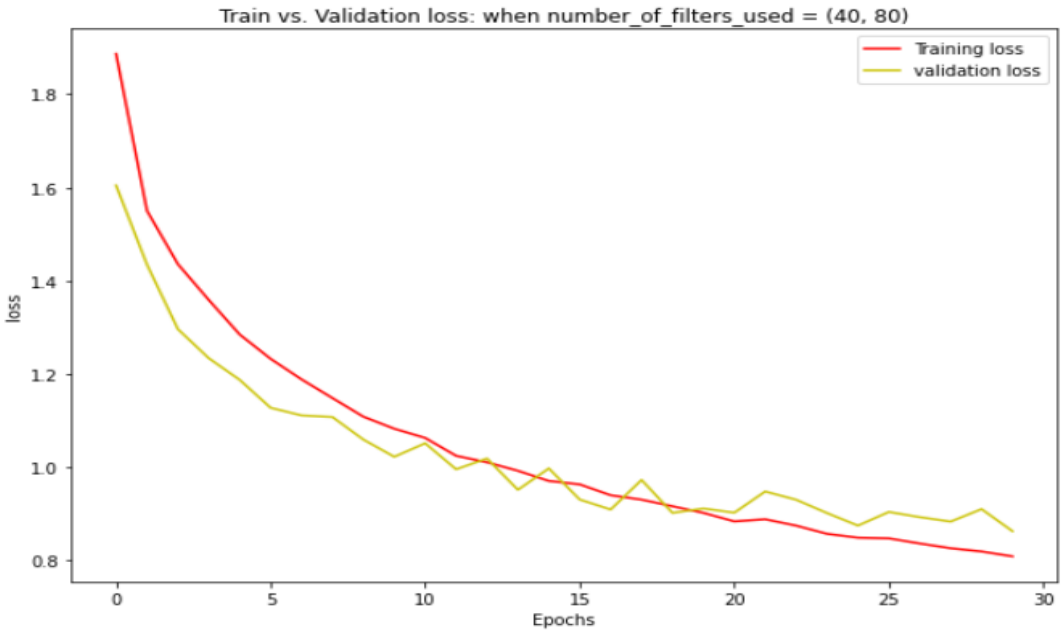
looping through the list and using each pair of tuples creating and compiling a network then fitting the training data to the network model.

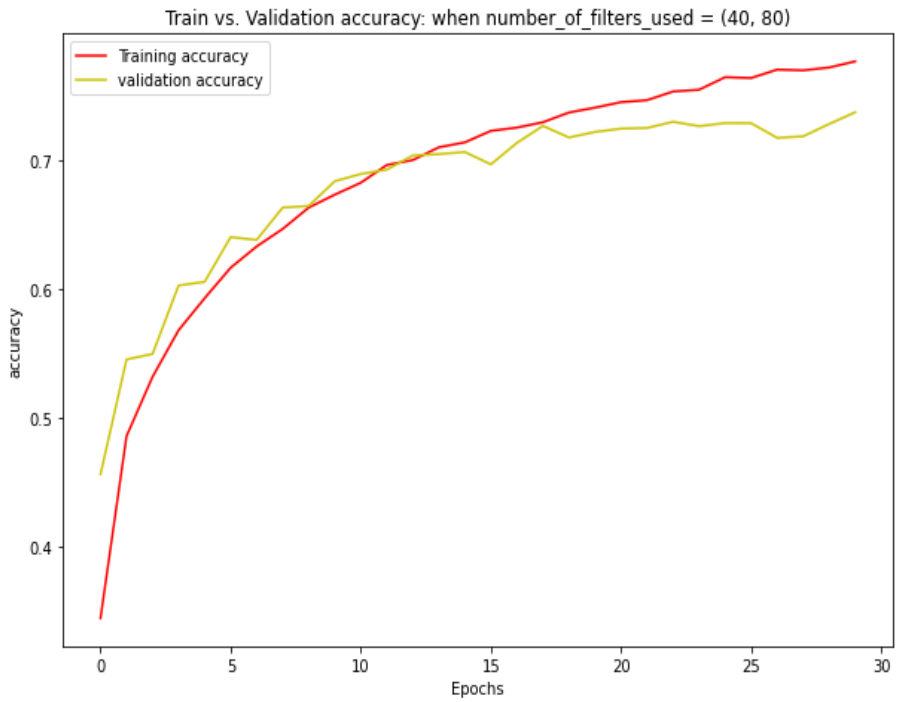
From the chats below:






Train vs. Validation accuracy: when number_of_filters_used = (32, 64)
Training accuracy
validation accuracy
accuracy
Epochs
0.4
0.5
0.6
0.7
0
5
10
15
20
25
30


Train vs. Validation loss: when number_of_filters_used = (40, 80)
Training loss
validation loss
loss
Epochs
0.8
1.0
1.2
1.4
1.6
1.8
0
5
10
15
20
25
30



We observe that (40, 80) and (32, 64)seem to be the optimal number of filters to be applied at t he Convolution layers, but we decide to with the (32, 64).

32 is applied at the first Convolution layer. 64 is applied at both the second and third Convolutio n layers. This pair has the highest accuracy of 0.74 and the lowest validation loss of 0.87.
So, comparing the two lists of the number of filters, we justify that (32, 64) is the optimal numbe r of filters to be used by the Convolution layers. This is because it has less overfitting with a high accuracy and low validation loss value.

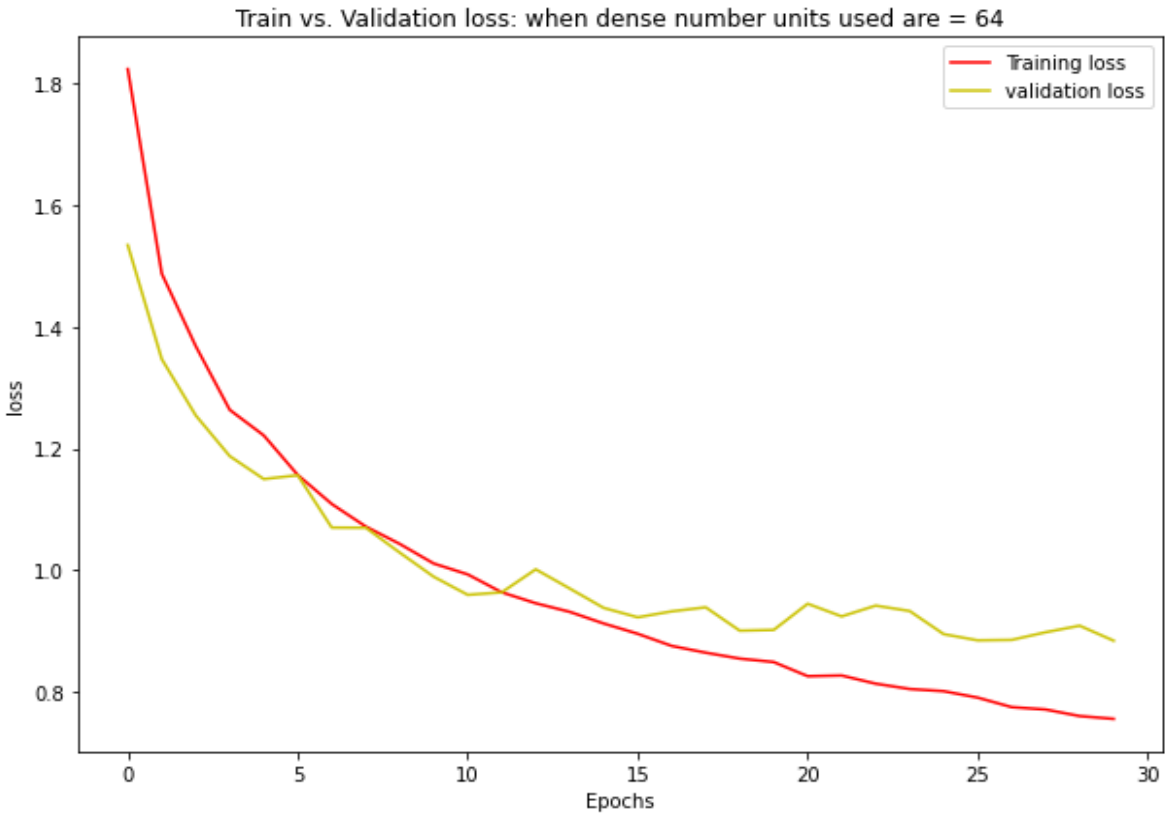
7. Number of units to be used in the Dense layer (fully connected layer)

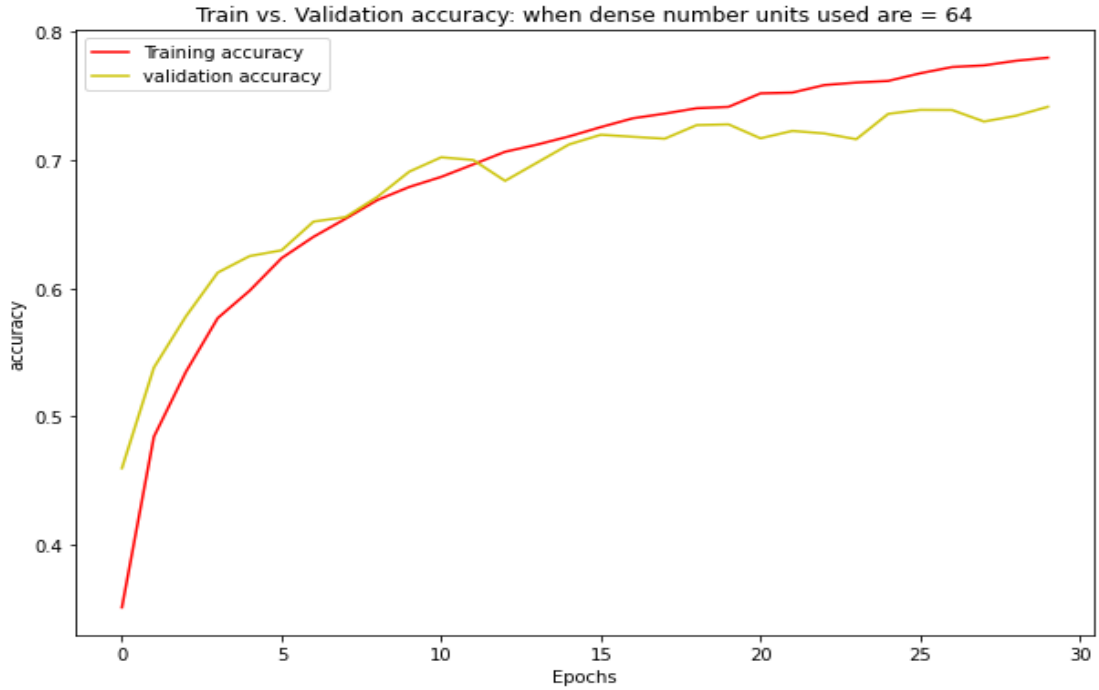
Using a list of numbers namely: 16, 32, 64, 128, 256, 512, 1024 and 2048 with the best already found hyper parameters above, I created a for loop where I was looping through the list and using each value as a unit in the fully connected layer creating and compiling a network then fitting the training data to the network model.
Amongst the different units we have explored, 64 is the optimal number of units to be used in th e dense layer because it has the highest validation accuracy and a low validation loss.

	number_of_units_used	val_loss
0	16	0.972134
1	32	0.911219
2	64	0.872164
3	128	0.876130
4	256	0.860306
5	512	0.881093
6	1024	0.891137
7	2048	1.089687

	number_of_units_used	val_accuracy_L
0	16	0.7002
1	32	0.7265
2	64	0.7356
3	128	0.7388
4	256	0.7505
5	512	0.7514
6	1024	0.7576
7	2048	0.7367

Also, we see that there is less overfitting as shown
below so we shall leave the number of units in the dense layer as is.


Train vs. Validation loss: when dense number units used are = 64
Training loss
validation loss
loss
Epochs
1.8
1.6
1.4
1.2
1.0
0.8
0
5
10
15
20
25
30

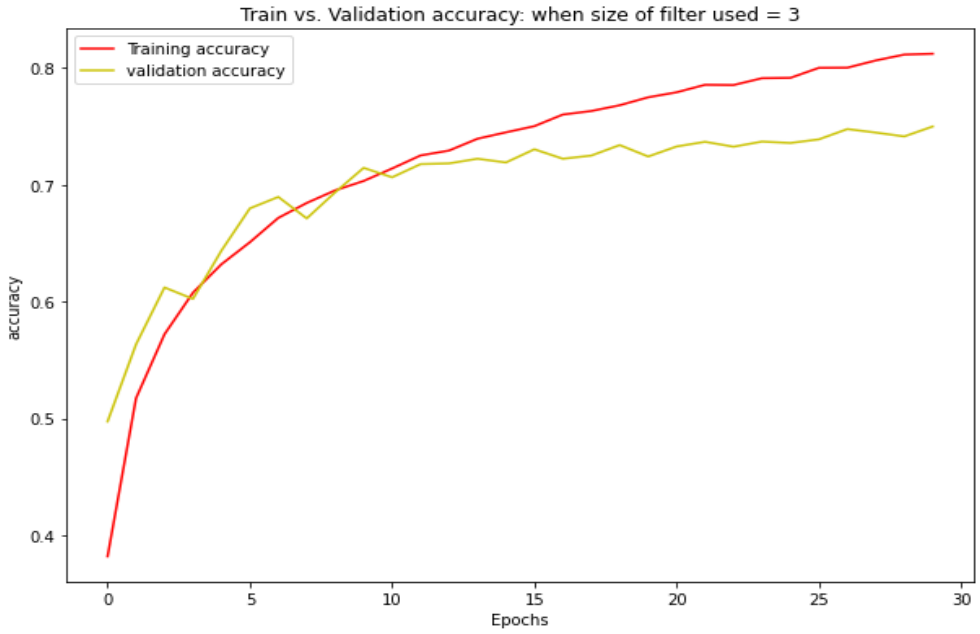

Train vs. Validation accuracy: when dense number units used are = 64
Training accuracy
validation accuracy
accuracy
Epochs
0.8
0.7
0.6
0.5
0.4
0
5
10
15
20
25
30

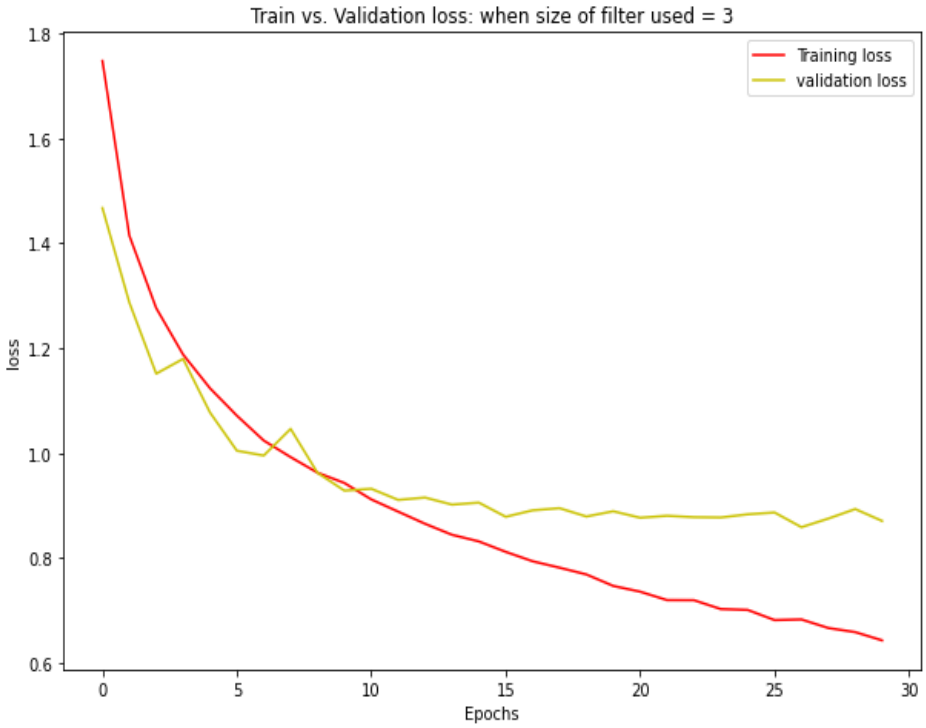
8. Size of filters

Using a list of numbers namely: 1, 3, 5, 7, 9 and 11 with the best already found hyper parameters above, I created a for loop where I was looping through the list and using each value as a filter size in the Convolution layer creating and compiling a network then fitting the training data to the network model.

We observe that as training epochs increases, both training and validation accuracy for the filter sizes 1, 3, 5, 7, and 11 increase. For filter size 1, the the validation accuracy is higher than the tra ining accuracy, which might be a sign of underfitting.

For 3, 5, 7, in the initial stages as the model is training the input neurons, the validation accuracy is high than training but at some point as the training epochs increase, training accuracy is highe r than validation accuracy. Compared to 1, 5, 7 and 11, we see that the filter size 3 is the optimal size since it has a high validation accuracy with less overfitting as shown below in the line charts:

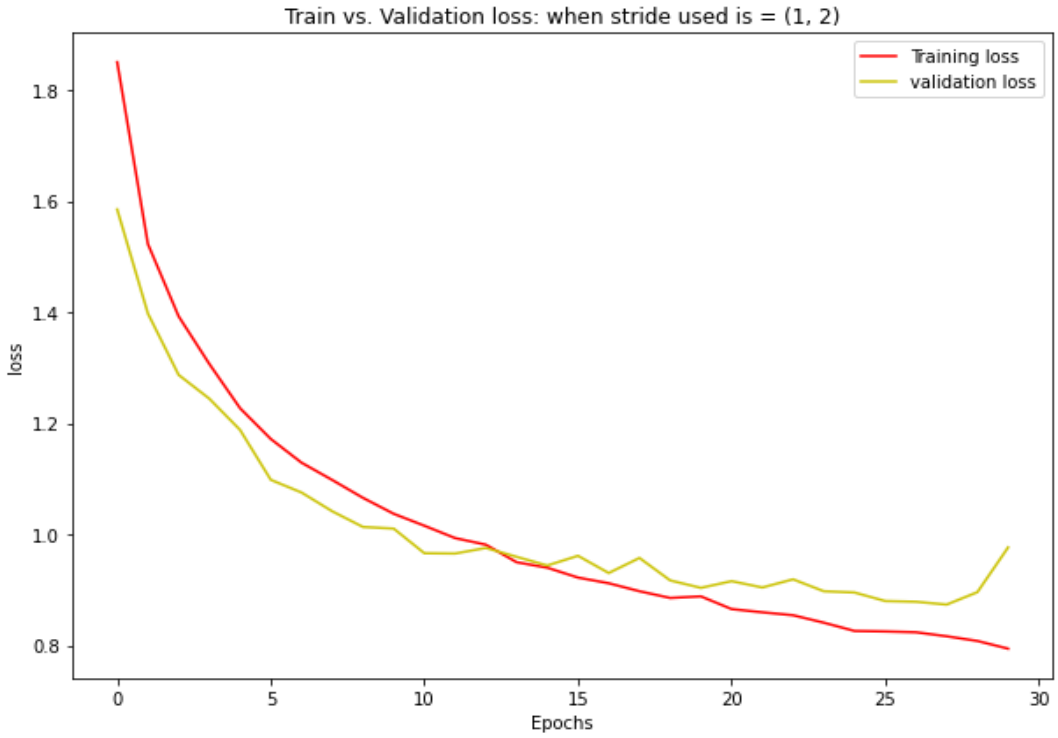


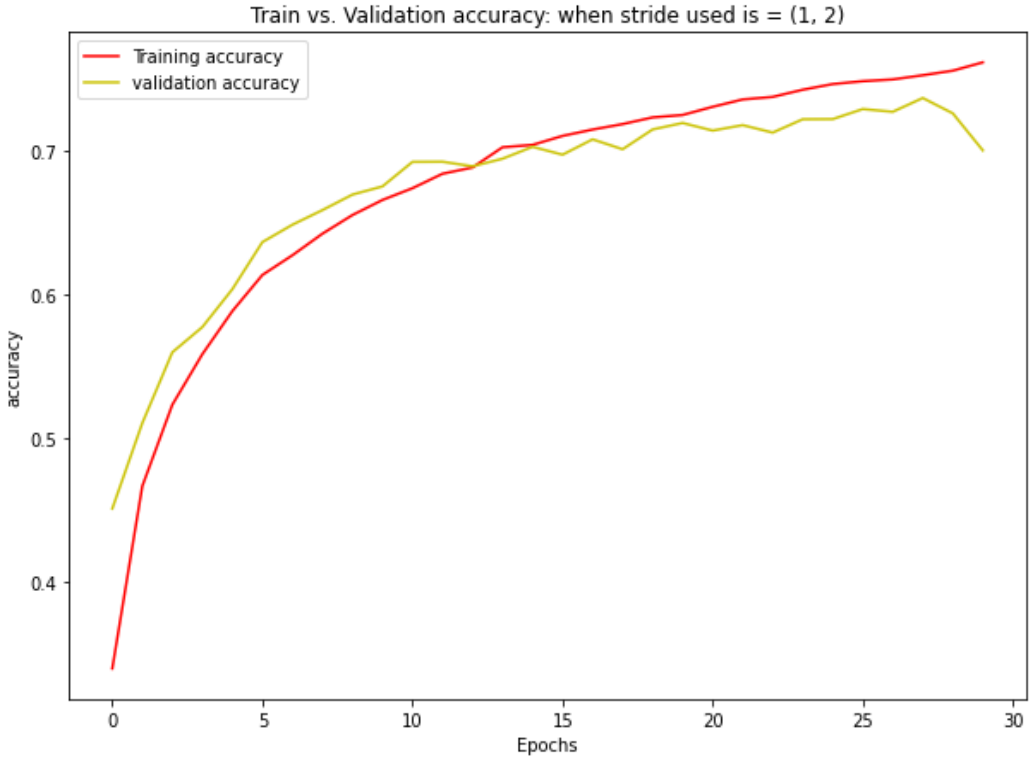


9. Size of Stride

Using a list of tuples namely: (1,1), (2, 2), (3, 3), (1, 2), (2,3) and (1,3) with the best already found hyper parameters above, I created a for loop where I was looping through the list and using each tuple as a stride in the Convolution layer creating and compiling a network then fitting the training data to the network model.

We observe that in the initial stages when the model is training, training loss is higher than valid ation loss but as the training epoch increases validation loss is higher than training loss, and amo ngst all the strides which have been used stride, (1, 1) has the best validation accuracy and low v alidation loss but with high overfitting when compared to stride (1, 2), which experiences less ov erfitting. This explain that stride (1, 2) is the best optimal stride.
and this is demonstrated in the line charts below:



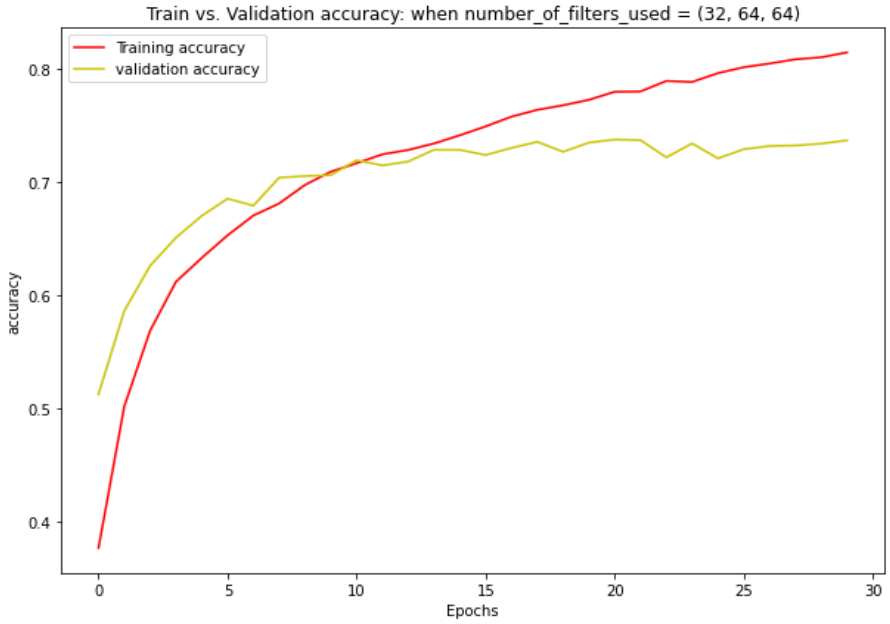


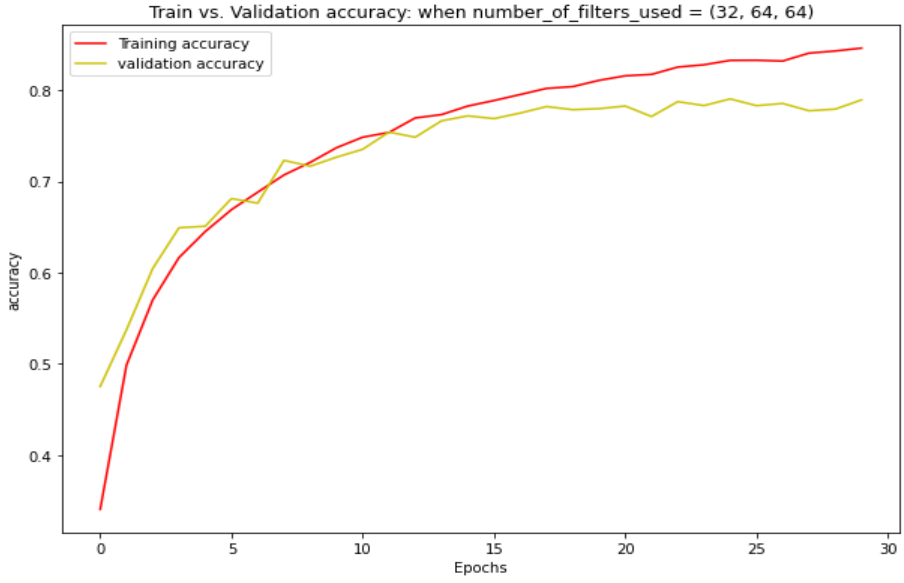
10. Add more Convolution layers:

I had a quite number of experiments I explored here:

- I added one Convolution layer at each layer and determined the best number of filters.
- Removed one Convolution layer from the last CNN and determining the number of filters
- Removed the middle Convolution layer from the network and determining the number of filters

Through the different experiments carried out, we discovered that the set (32, 64, 64) had the highest validation accuracy and lowest validation loss amongest all pairs we used. 32 was applied to the first set of Convolution layer, 64 was applied to the second (in the middle) and third set of Convolution layer so I decided to go with the option where one Convolution layer was added to the existing CNN layers.



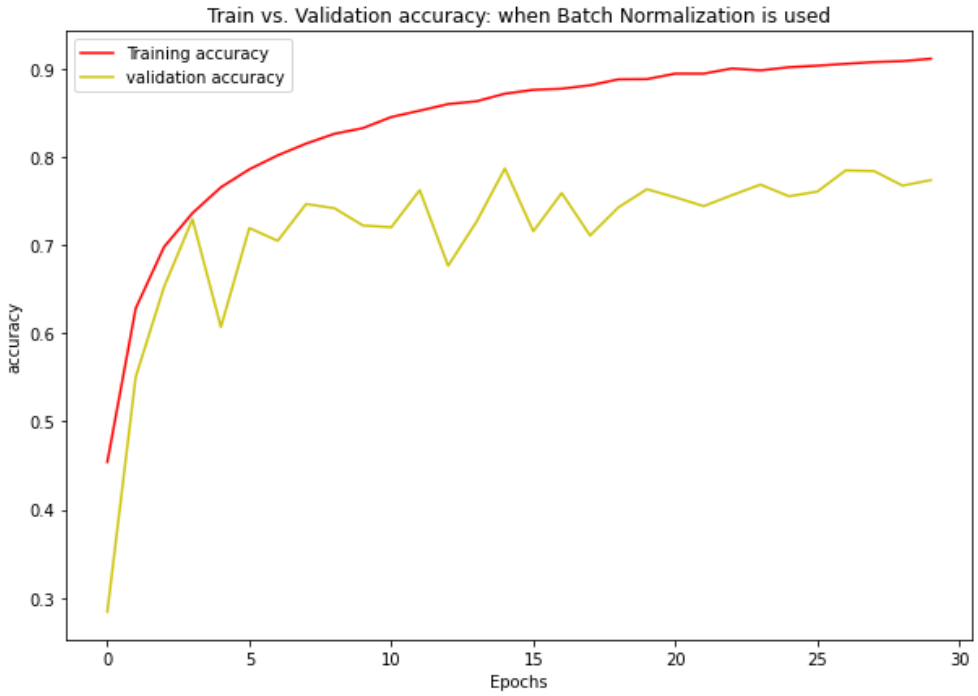


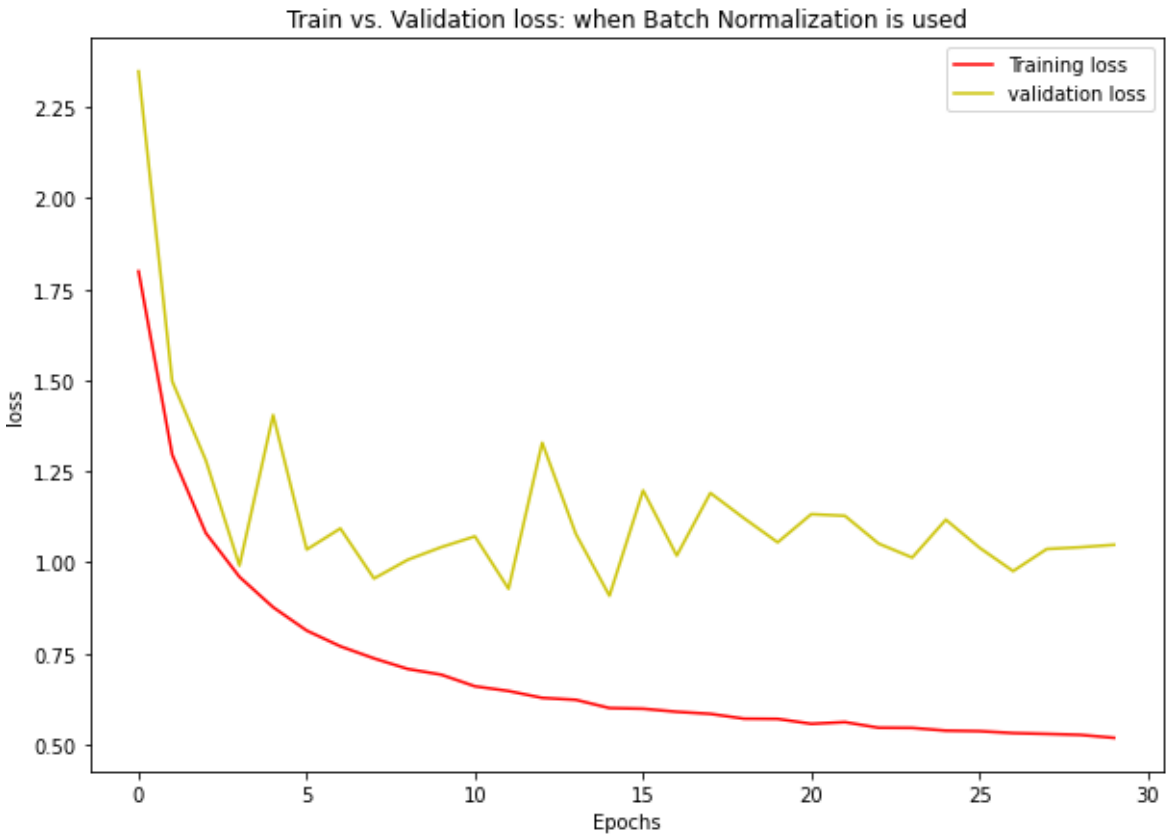
Here, that this pair, (32, 64, 64), had less overfitting compared to other pair of filters.

11. Data Augmentation

    I did play around with data argumentation, and I found it something interesting to explore but my results were disappointing.

12. Added Batch Normalization to standardize the input images after the activation function of the previous layer in order to get the best performance. I didn't find the model (non-best model) built using batch normalization worthy using because it was overfitting the training data as shown below:





13. GridSearchCV/RandomSearchCV

I was supposed to have used GridSearchCV, but the peformance of the module is dissap ointing so I decided to use RandomSearchCV, which also ran for hours because it constructs and evaluates one model for each combination of parameters.

First, I ran random searchCV with a list of different batch size values namely: 32, 64, 96 , and 100 and with a list of different epoch values namely: 10, 30, 80, 100 to determine the optimal parameters. The results I got were different from the above experiments used. Epoch and batch size resulted into 80 and 32 respectively.

Second, using values of 0.001, 0.01, 0.1 and 0.3, we determined the learning rate and 0.001 was the optimal value, which was not different from what I had before.

Third, using values of 24, 32, 48, 64, 96, 80 and 128, we determined the optimal neurons to be used in the fully connected layer and 64 was the best parameter value determined by the Random SearchCV.

Fourth, using values of dropout rate: 0.0, 0.1, 0.2, 0.3, 0.4, and 0.5 and for weight constraint: values from 1 to 5, we shall use Random SearchCV to find the best parameter values. The reason why I decided to take this approach is because to get good results, it's always advisable to combine dropout with weight constraints like the max norm constraint and this involves fitting both the dropout percentage and weight constraint as suggested by Jason Brownlee PhD. I ran the Random SearchCV which took 1 hour and 50 mins, I walked away from my machine and came back when google colab had timed out my session so I decide to do it the manual way. So, doing

this manually with different combination of values I found that dropout rate of 0.2 and the weight constraint of 4 were the optimal parameter values.

C. **Final Conclusion on the best model**:
   The best model had a low validation loss and high validation accuracy with less overfitting.
D. **My reaction and reflection:** I found this assignment to be interesting and challenging because it taught me how to think like a Data Scientist especially when you are building and fine tunning models.
   I also learnt how to use GridSearchCV/ RandomSearchCV with the keras Deep learning module.