COL341 Assignment-4 Report

By: Divyanshu Agarwal Entry No. 2020CS10343

3: Part 1 CNN from scratch

Experimental setup :

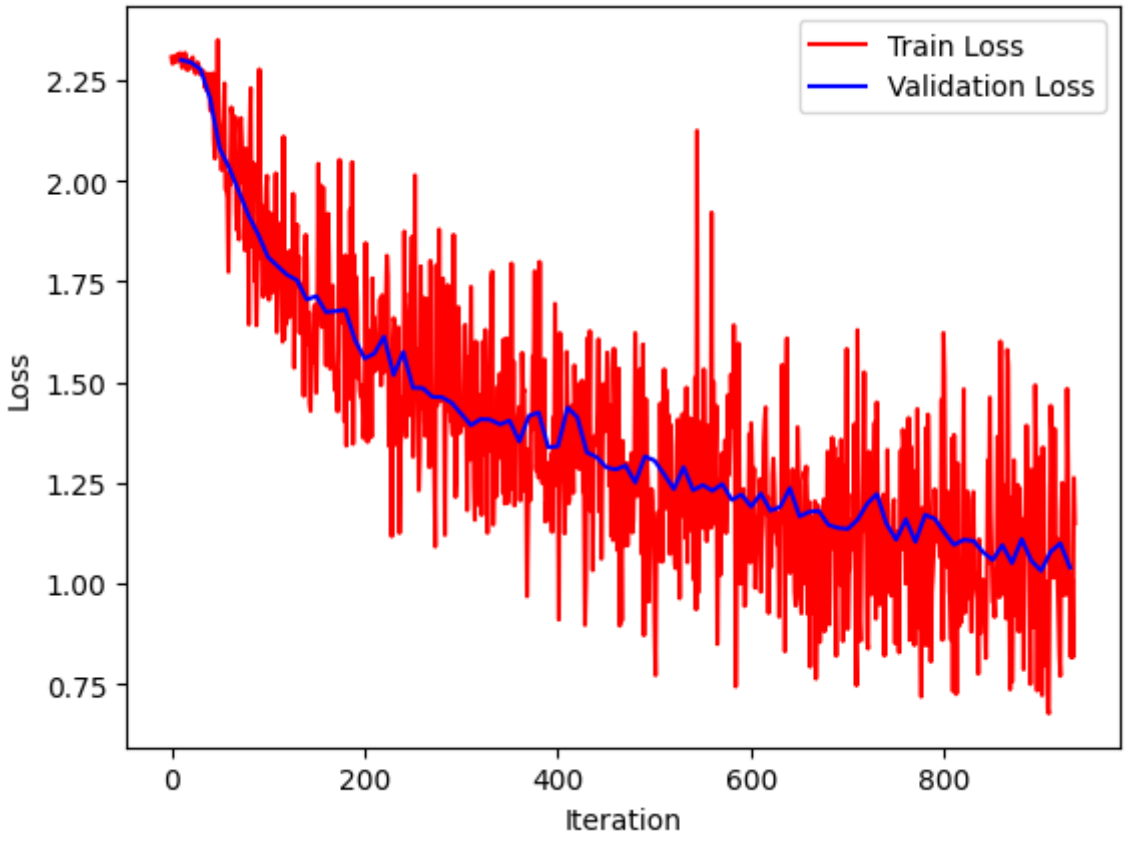
I constructed the CNN by making various layers as python classes with inputs, outputs and weights/kernels, biases. Each layer had its' own forward and backward propagation functions.

Since my model took considerable time on each epoch , I reduced the number of training epochs to a fifth of what was asked. Moreover I sampled for training accuracy on batches after 5 iterations and measure validation accuracy and loss after 50 batches. This helped me obtain validation accuracies at around 90 points and train accuracy at around 900 points .
I obtained the following training curve :

Training time for 1 epoch : 3 hrs.

Note the here each epoch consisted of about 200 iterations##. Hence the x-axis is interpreted to be over 5 iterations.

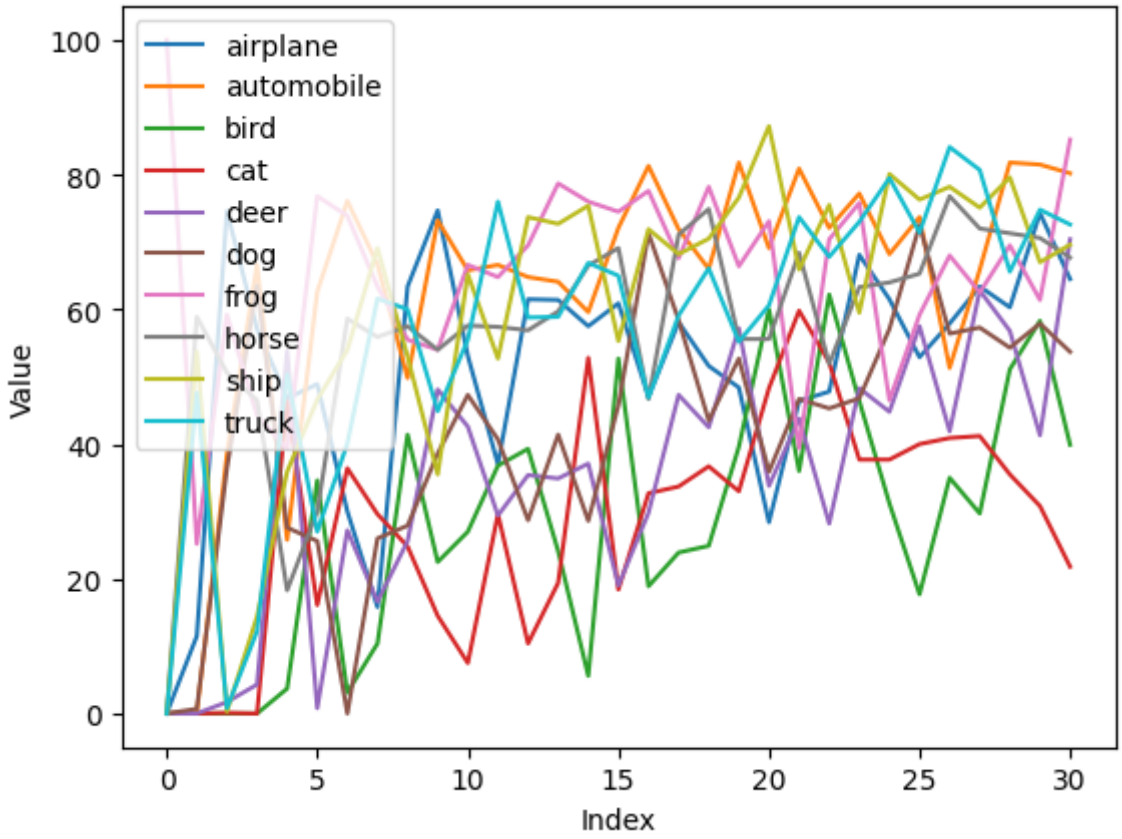
Training and Validation Loss



From the above curve we can see that the train accuracy is reducing in general however it ocassionally shoots up to a large value for some batches. Validation loss also decreases. Had I allowed it to train for a longer period I could have obtained a better loss.

Note : Each epoch had 6 samples for validation accuracy chosen out of all the 90 points available . Thus the x-axis is to be interpreted to distributed over 5 epochs.

Class wise Validation Accuracy



We can see that the class wise validation accuracy reaches above **40%** for most of the classes except cat. And the highest it reaches is for frog which is around 80% .

The overall validation accuracy of my model was 53% after 5 epochs.

---

4 : Part 2 CNN using Pytorch

I made a CNN with the following layers using pytorch with the following layers :

- Conv2d -> Relu() -> MaxPool->Conv2D->Relu->Maxpool->Conv2d->Relu->Reshape->Dense->Relu->Dense->Softmax

  I conducted the follwing experiments and obtained these results :

Varying Learning Rate :

In all these experiments I used

Optimiser : Adam

Epochs : 12

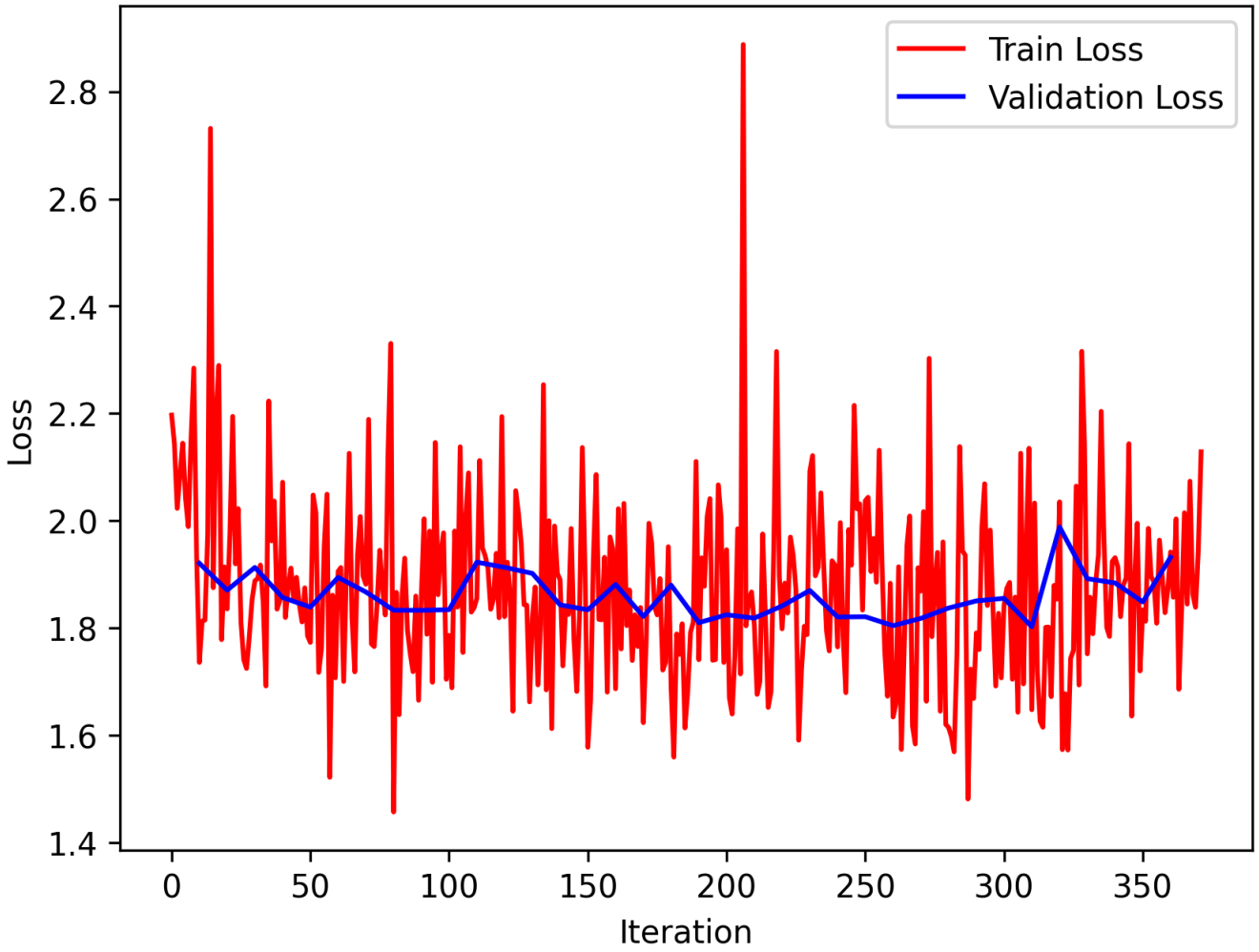
Batch_size=32

Fixed Learning Rate (no scheduler)

Note : x axis here shows the iteration value. In each epoch I sampled training loss after 50 batches and validation loss after 500 batches. Hence each epoch had (50k/50*32 = 31 iterations). Hence we had a total of 31 * 12 = 372 iterations for training loss which can be seen on the x axis.

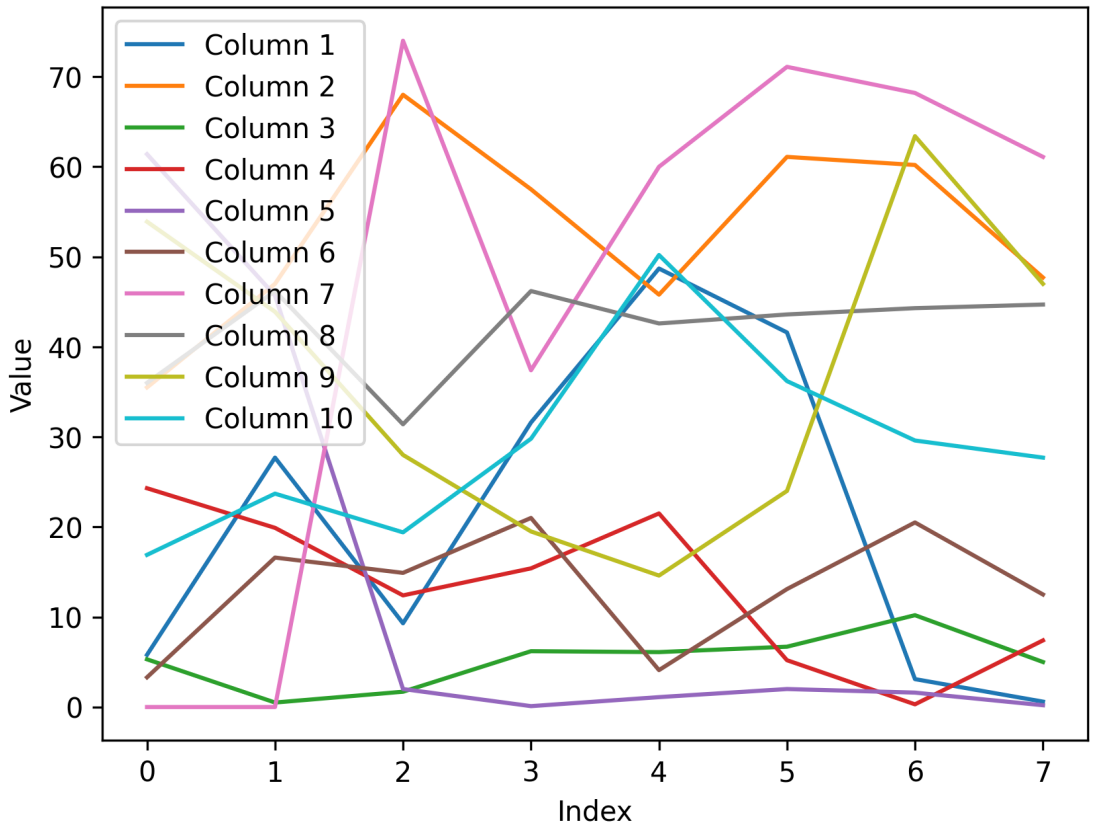
Learning Rate = 0.01

Train Loss and Validation Loss



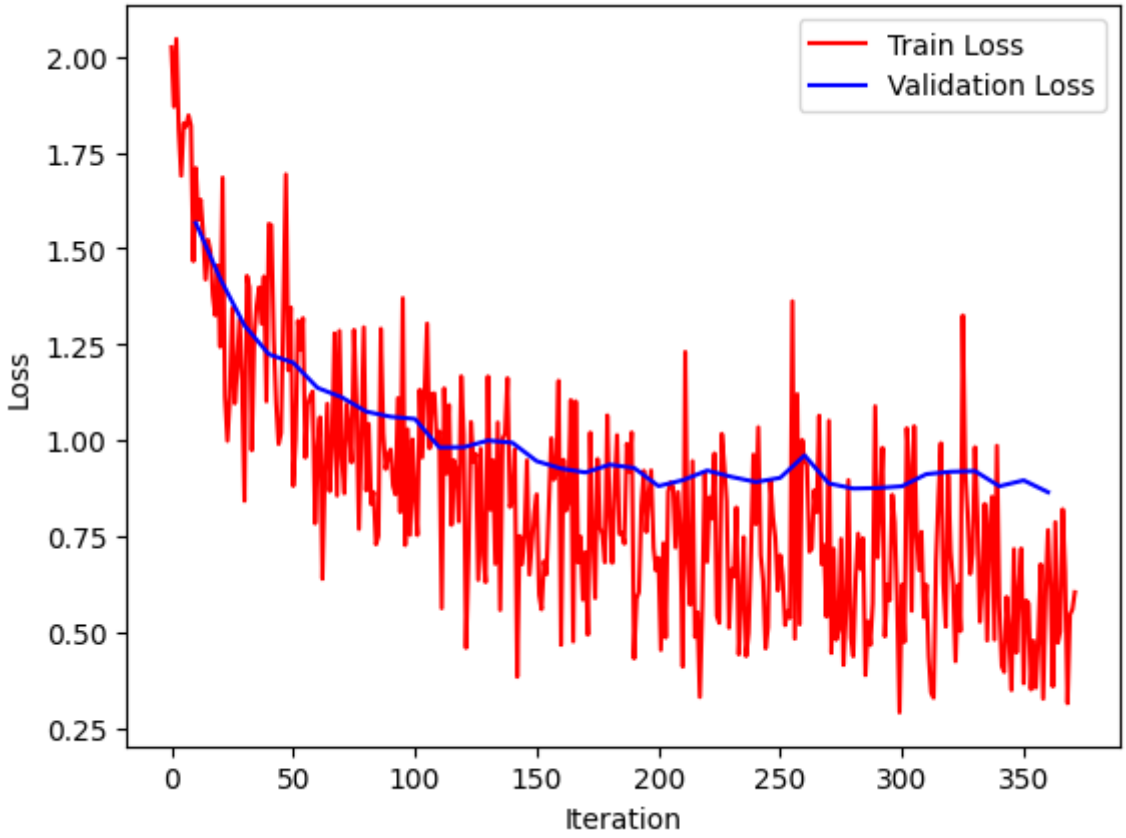
Class wise Validation accuracy

Note that from the calculation above we had 372/10 = 37 points where validation accuracy was measured. I had computed class wise validation accuracy for all these iterations. However, plotting all those points and 10 lines made it very untidy on this graph, so I uniformly chose 37/5 = 7 points from those and plotted them.



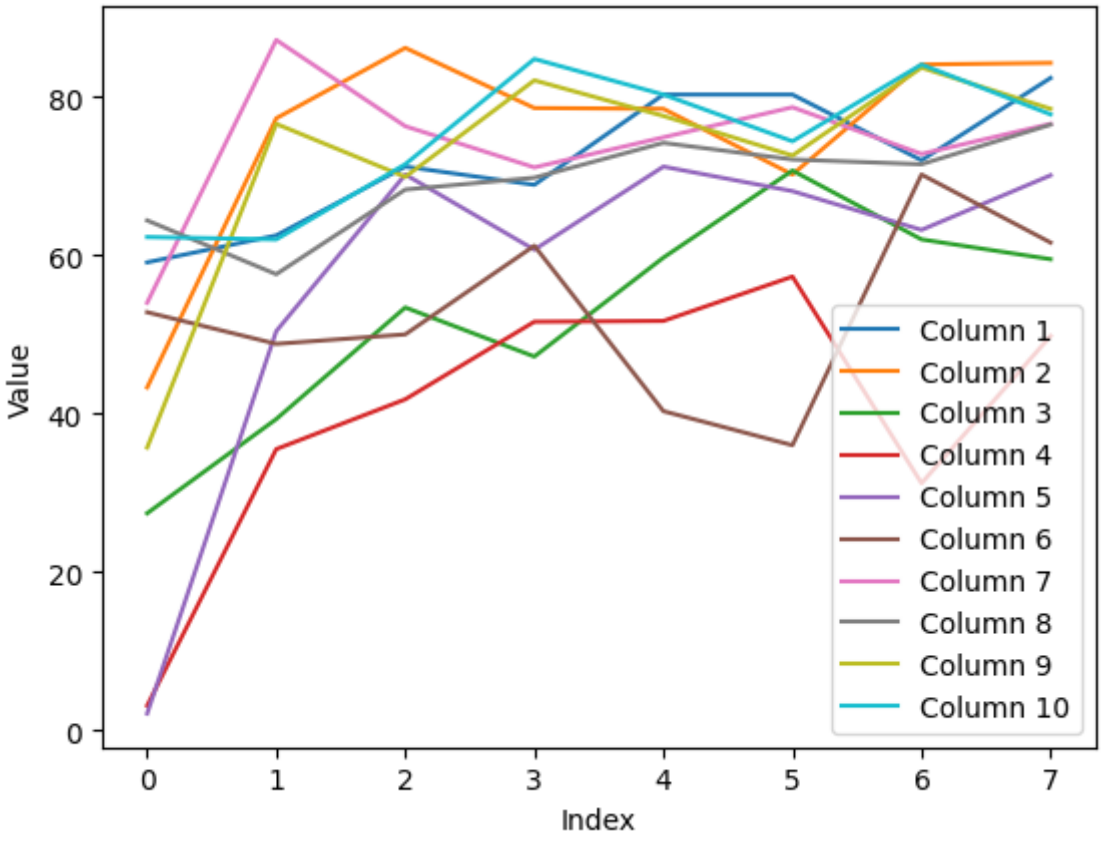
Learning Rate = 0.001

Train Loss and Validation Loss



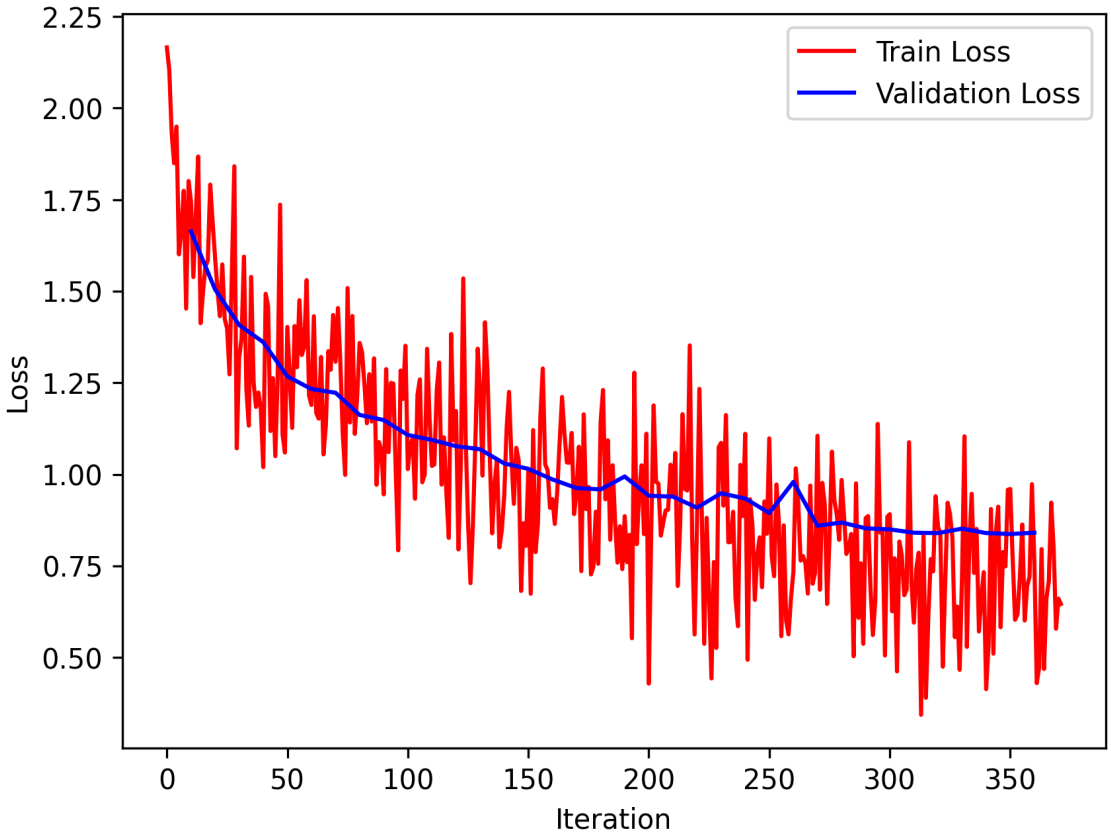
Class wise Validation accuracy

Note that from the calculation above we had 372/10 = 37 points where validation accuracy was measured. I had computed class wise validation accuracy for all these iterations. However, plotting all those points and 10 lines made it very untidy on this graph, so I uniformly chose 37/5 = 7 points from those and plotted them.



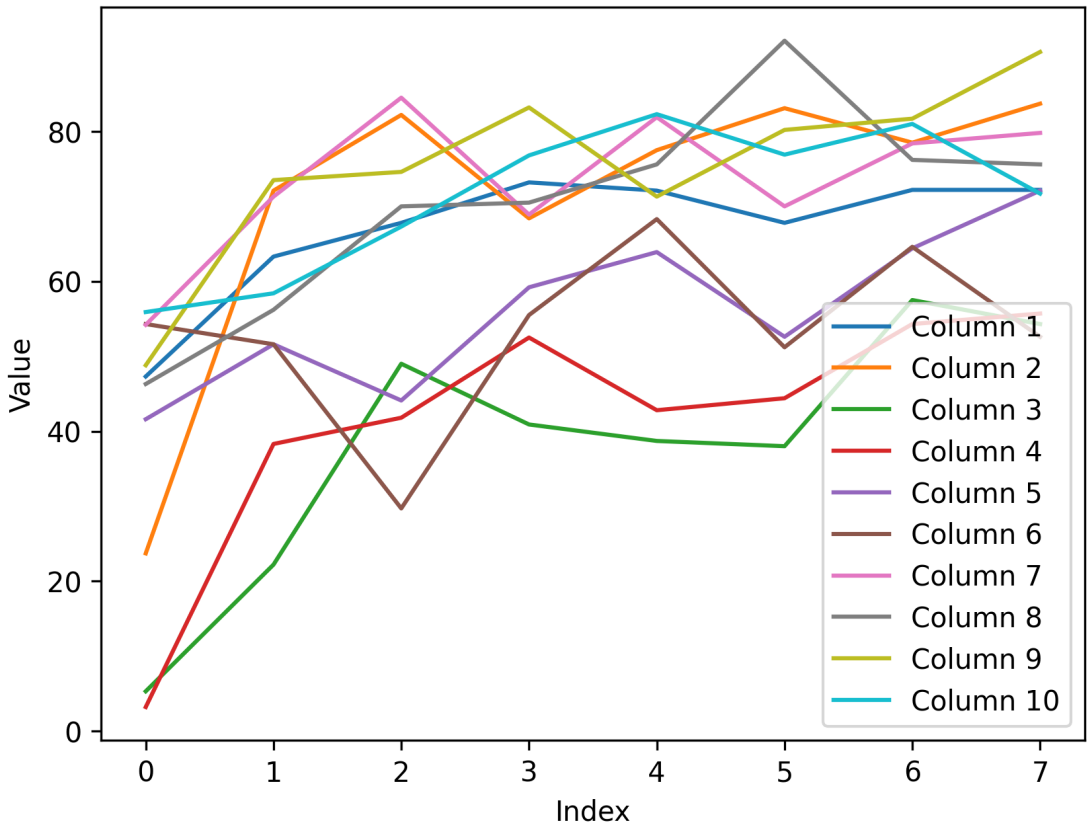
Learning Rate = 0.0003

Train Loss and Validation Loss



Class wise Validation accuracy

Note that from the calculation above we had 372/10 = 37 points where validation accuracy was measured. I had computed class wise validation accuracy for all these iterations. However, plotting all those points and 10 lines made it very untidy on this graph, so I uniformly chose 37/5 = 7 points from those and plotted them.



Varying Learning Rate :

In all these experiments I used

Optimiser : Adam

Epochs : 12

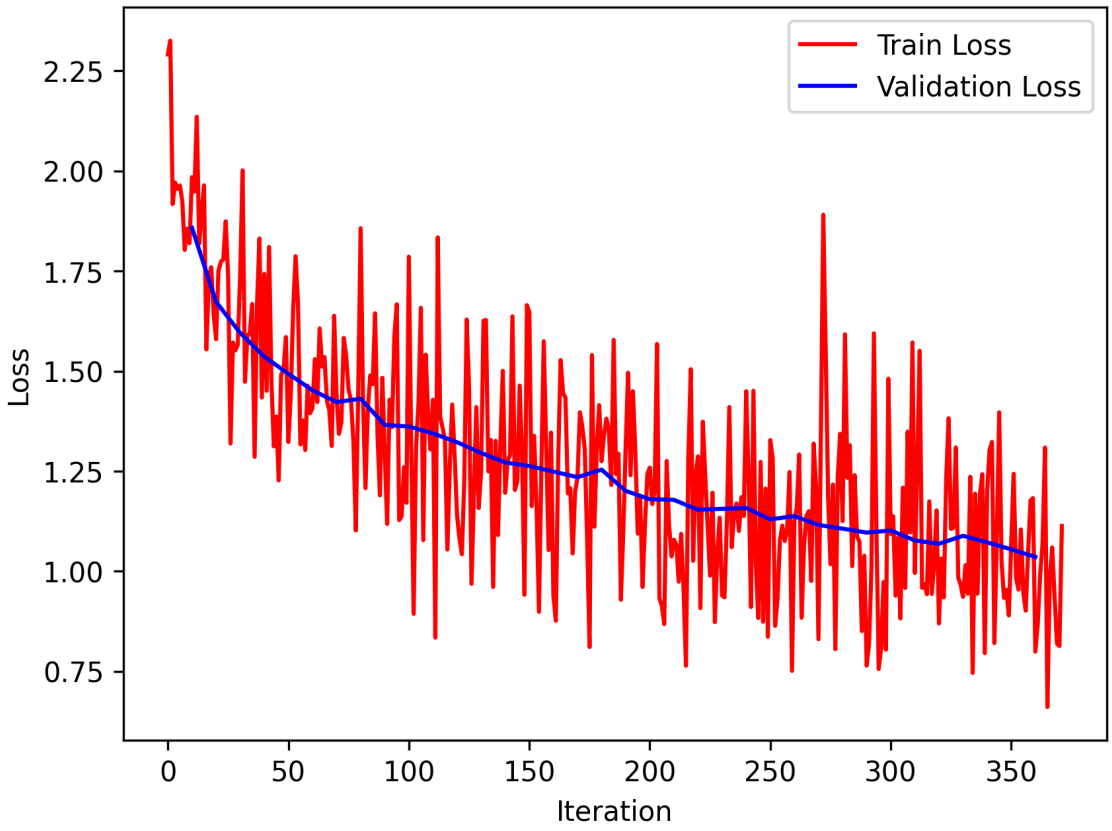
Batch_size=32

Fixed Learning Rate (no scheduler)

Note : x axis here shows the iteration value. In each epoch I sampled training loss after 50 batches and validation loss after 500 batches. Hence each epoch had (50k/50*32 = 31 iterations). Hence we had a total of 31 * 12 = 372 iterations for training loss which can be seen on the x axis.

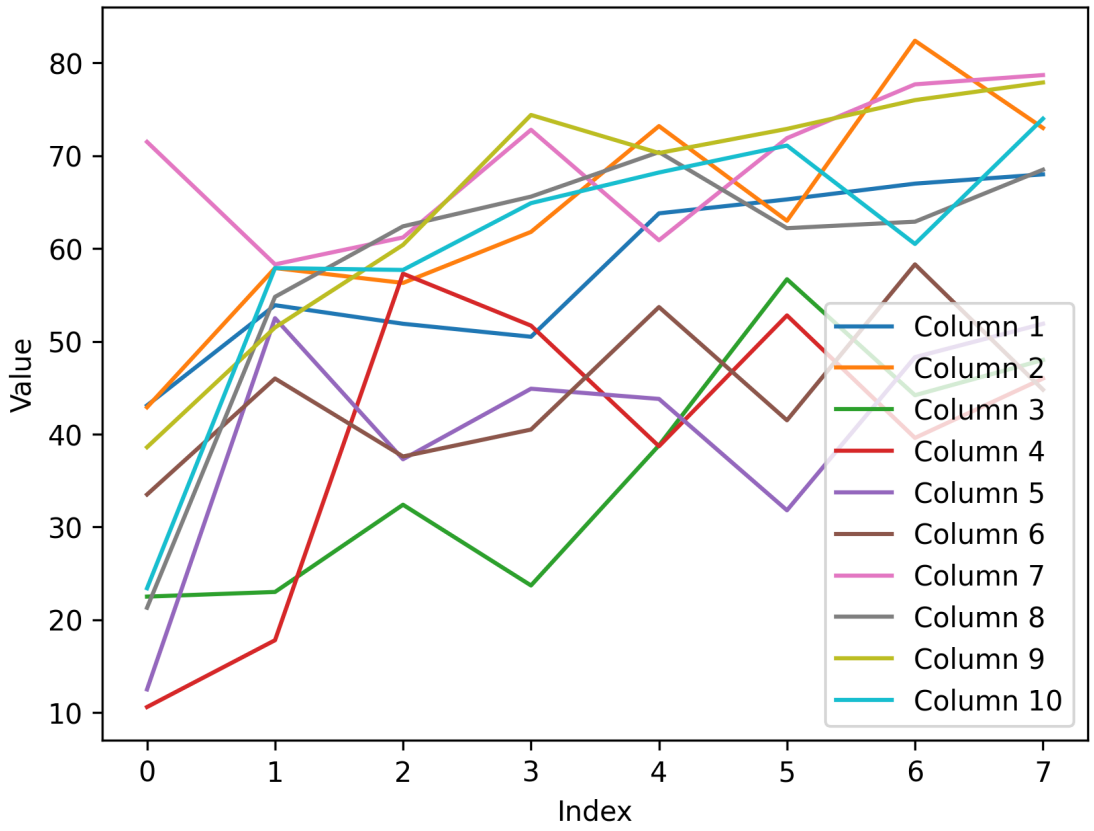
Learning Rate = 0.0001

Train Loss and Validation Loss



Class wise Validation accuracy

Note that from the calculation above we had 372/10 = 37 points where validation accuracy was measured. I had computed class wise validation accuracy for all these iterations. However, plotting all those points and 10 lines made it very untidy on this graph, so I uniformly chose 37/5 = 7 points from those and plotted them.



Analysis on Varying Learning Rate :

We can see that when the learning rate is very high = 0.01 then the training and the validation accuracy remain almost constant, which means that the model osciallates around minimal regions but does not converge due to large learning rate, every time it tends in the direction of the minimum it takes a large step and skips the minimum , hence there is almost no decrease in training loss and validation loss. The same can also be inferred from class wise accuracy plots which display an oscillating behaviour, when one class achieves high accuracy the other achieves a lower accuracy and they keep fluctuating, the overall accuracy remains poor.

When we start to decrease the learning rate to a better value = 0.001 and 0.0003 we can see that the model almost monotonically converges, and reaches a validation loss after around 150 iters in the first case and 350 in the second. We can also see that class-wise accuracies in the first case oscillate more while in the second case they seem to increase all together. This suggests that when lr=0.0003 we are taking the right amount of steps so that each step doesn't penalises accuracy on other classes while improving current accuracy.

Finally when lr=0.0001 which is very low , we can see a very smooth decrease in validaiton accuracy, however we see that even after 350 iterations = 13 epochs the valdiation loss is still above 1, which other learning rates were able to achieve after 3-4 epochs, this shows the slow rate of convergence.

Learning Rate Scheduler

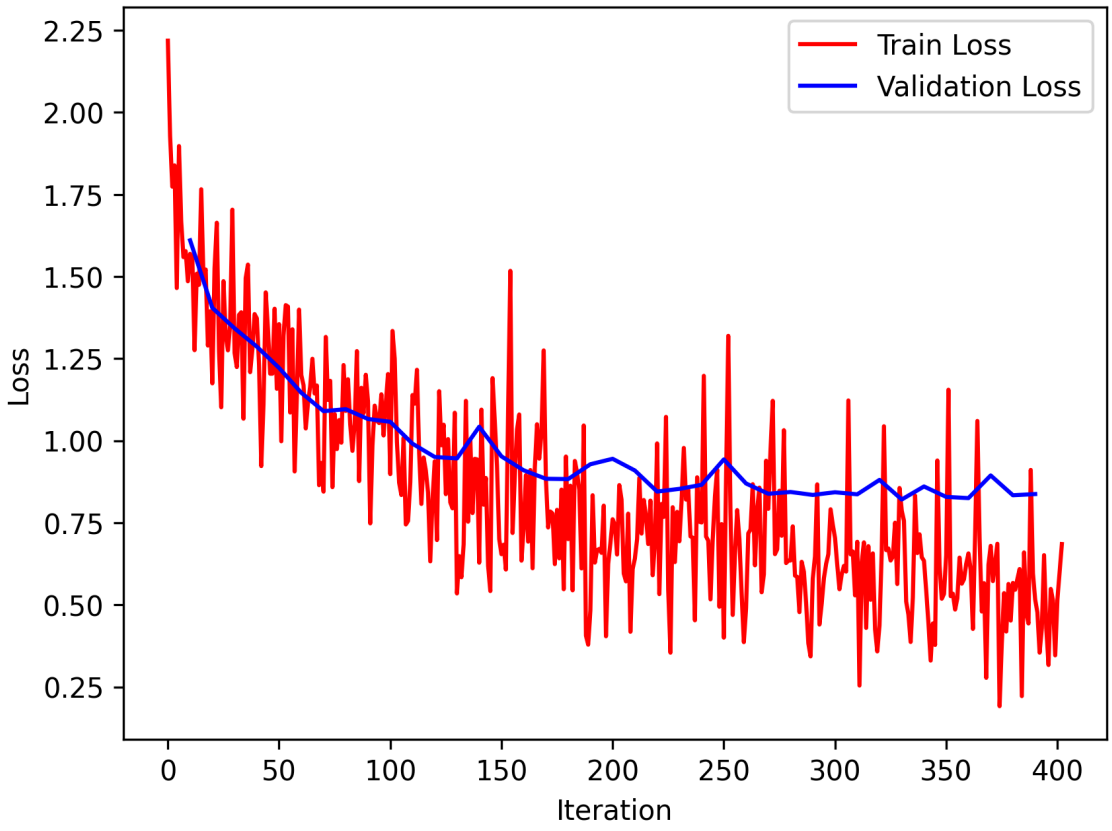
I used the following parameters

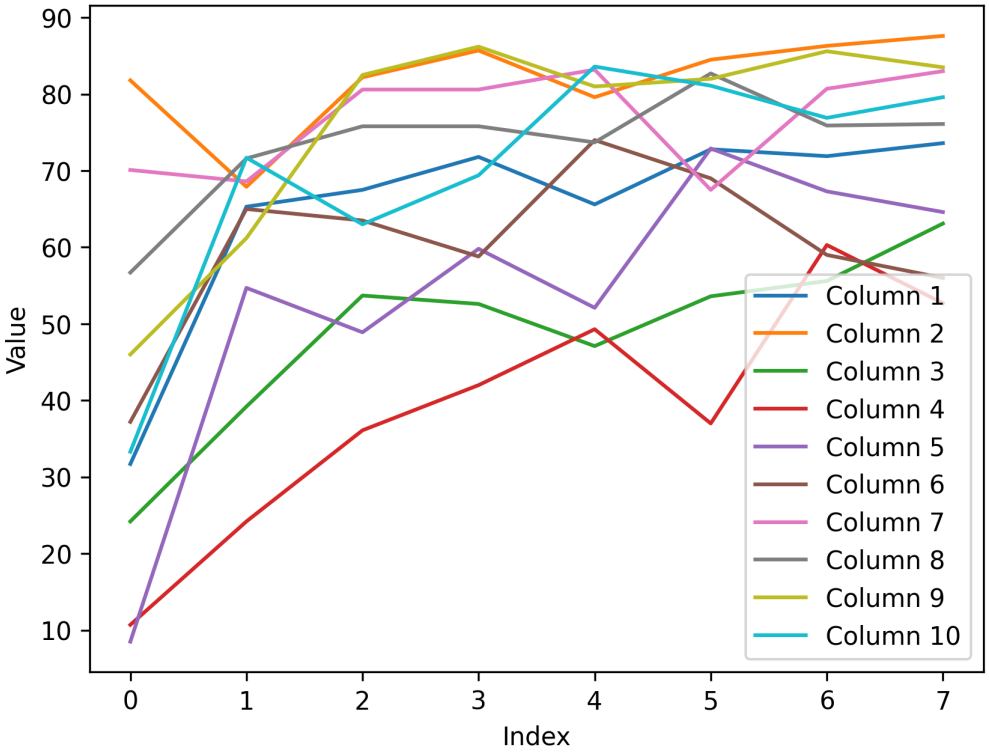
Start learnign rate = 0.001

batch_size=32

epochs= 13

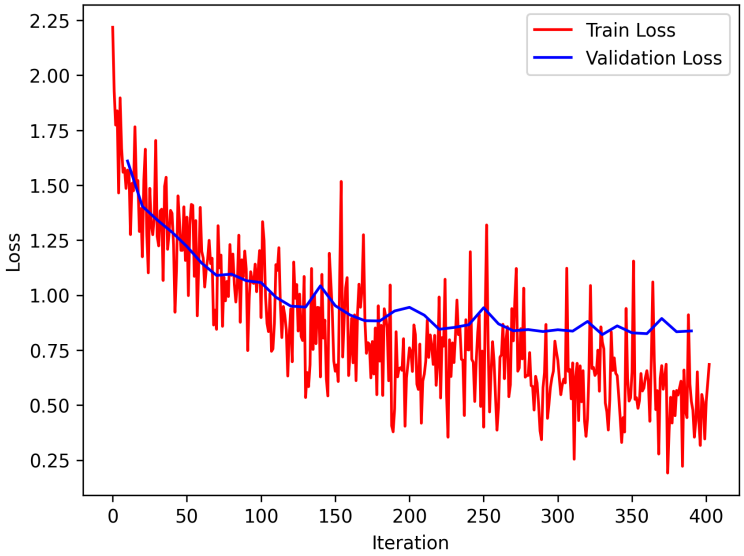
and used a the cosineAnnealingLR with T_max = 32, eta_min = 1e-6 and obtained the following results:

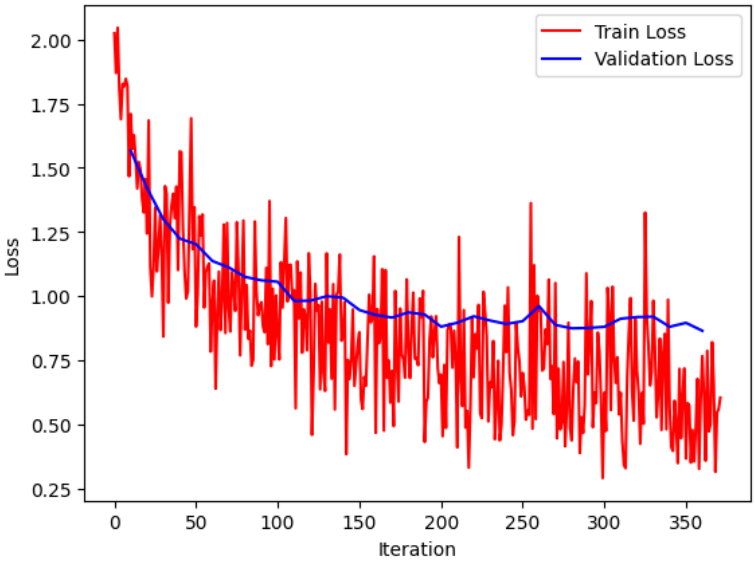




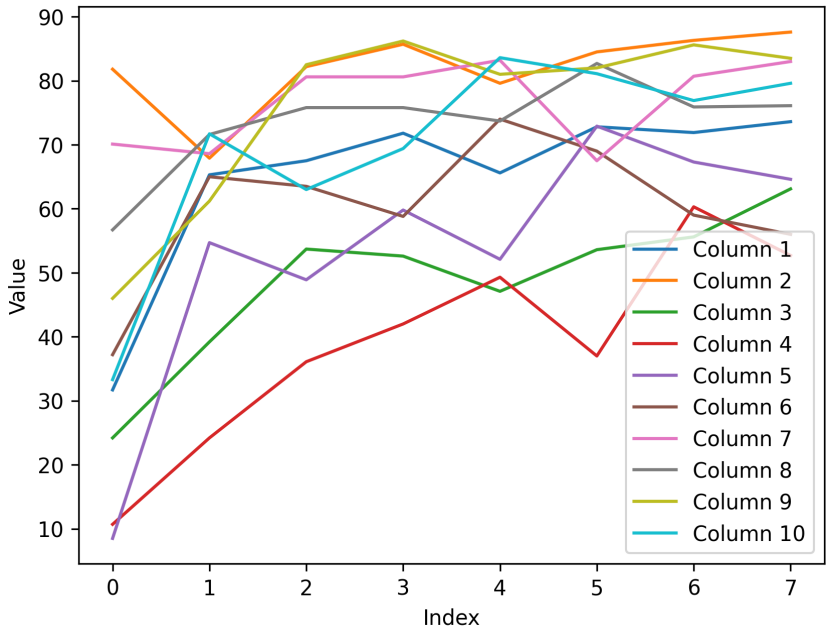
Comparison with constant learning rate :

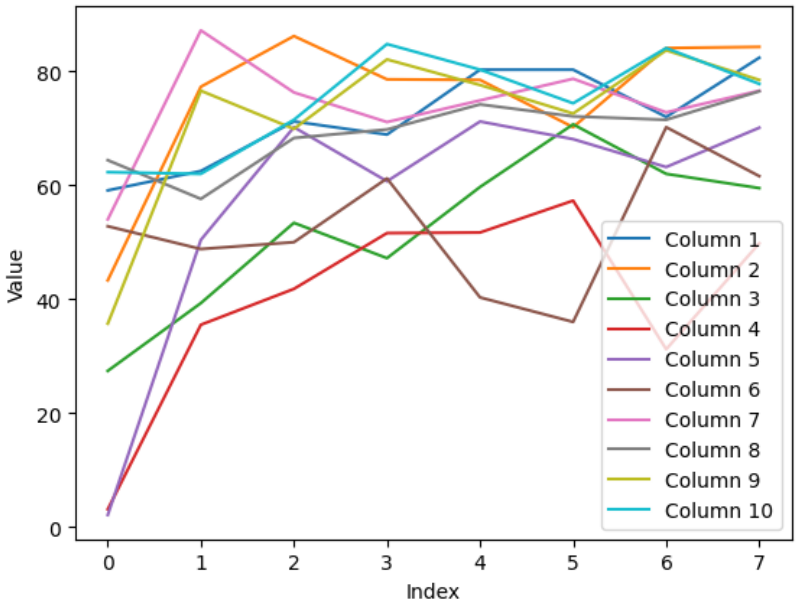
If I compare it with the same but with constant learning rate on the same parameters we get :





The left plot is cosine annealing lr in both the cases





Analysis :

We can see that the left plot has less variation in training loss, this is possible because since it decreases the learning rate after some iterations it prevents taking large steps and only takes large steps in the first few iterations when the model has not learnt anything . Other than that we obtain almost similar validation. We just see lss variance in training accuracy.

Varying Number of Epochs

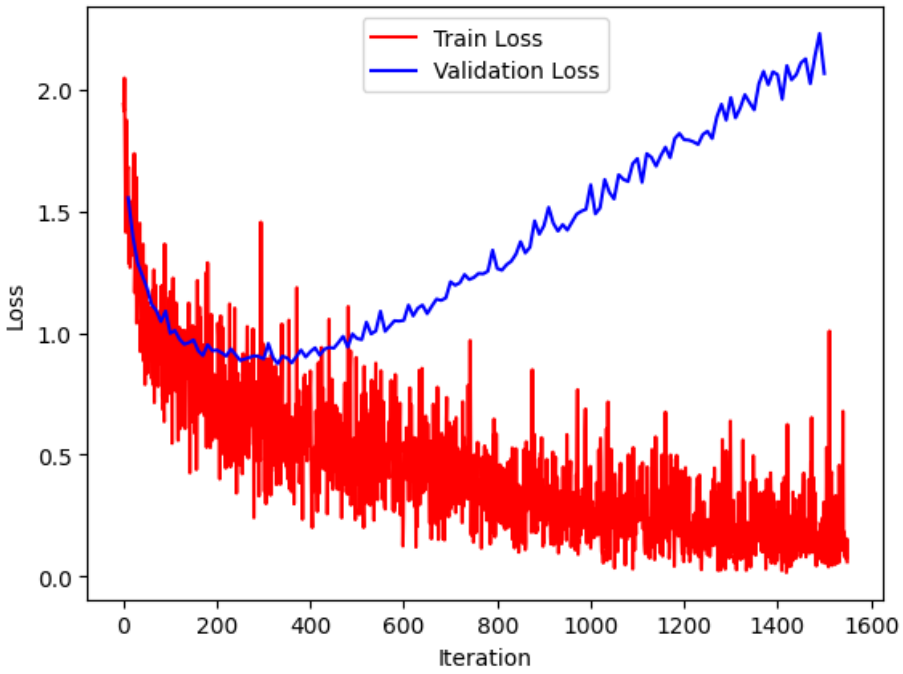
Learning Rate = 0.001 (fixed)

batch size = 32

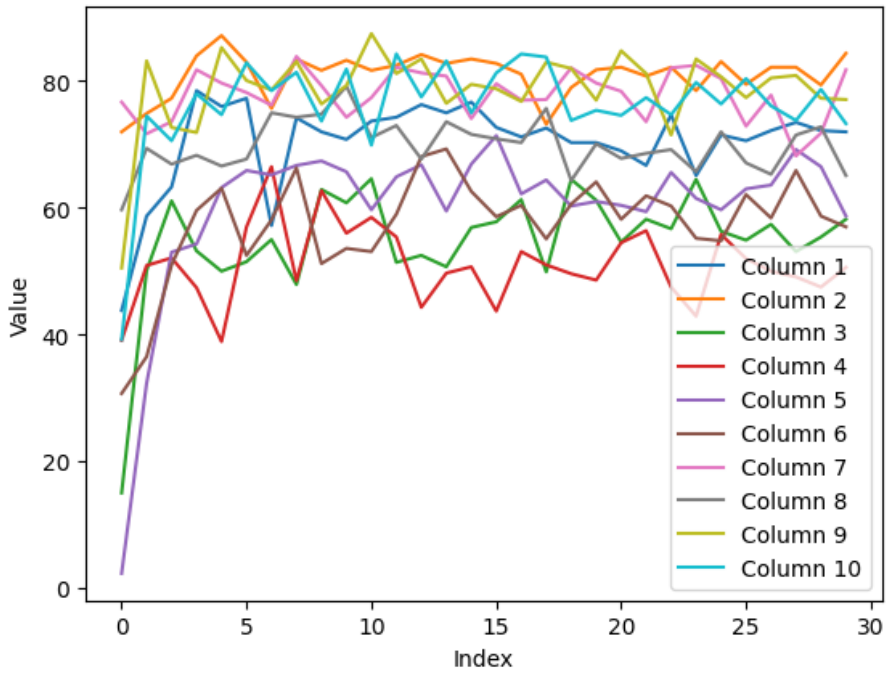
Adam Optimiser

I set the number of epochs to a large number =50 and observed how the training loss and validation loss vary. I obtained the following plots :

Note : **x axis here shows the iteration value. In each epoch I sampled training loss after 50 batches and validation loss after 500 batches. Hence each epoch had (50k/50*32 = 31 iterations). Hence we had a total of 31 * 50 = 1550 iterations for training loss which can be seen on the x axis.**



Note that from the calculation above we had 1550/10 = 155 points where validation accuracy was measured. I had computed class wise validation accuracy for all these iterations. After uniformly choosing one-fifth points 155/5 = 31 points from those and plotted them.



Analysis :

We can see that after about 400 iterations (=12 epochs = index 7) , the valdiation loss starts to increase while the training loss still keeps decreasing. This shows that after these many iterations our CNN starts to **overfit** on the training set and the validation accuracy tends to decrease. This is also reflected in the class-wise accuracy plots where we can see that in the first few epochs there is a large improvement in class wise accuracies, however after a ceratn number class accuracies start to dwindle with increase in one class leading to a decrease in other.

Varying Batch Size

For all these experiments I used the following parameters

Learning Rate : 0.001 (fixed)

Epochs =13

Adam Optimiser

I tried batch sizes : 4,8,16,32 .

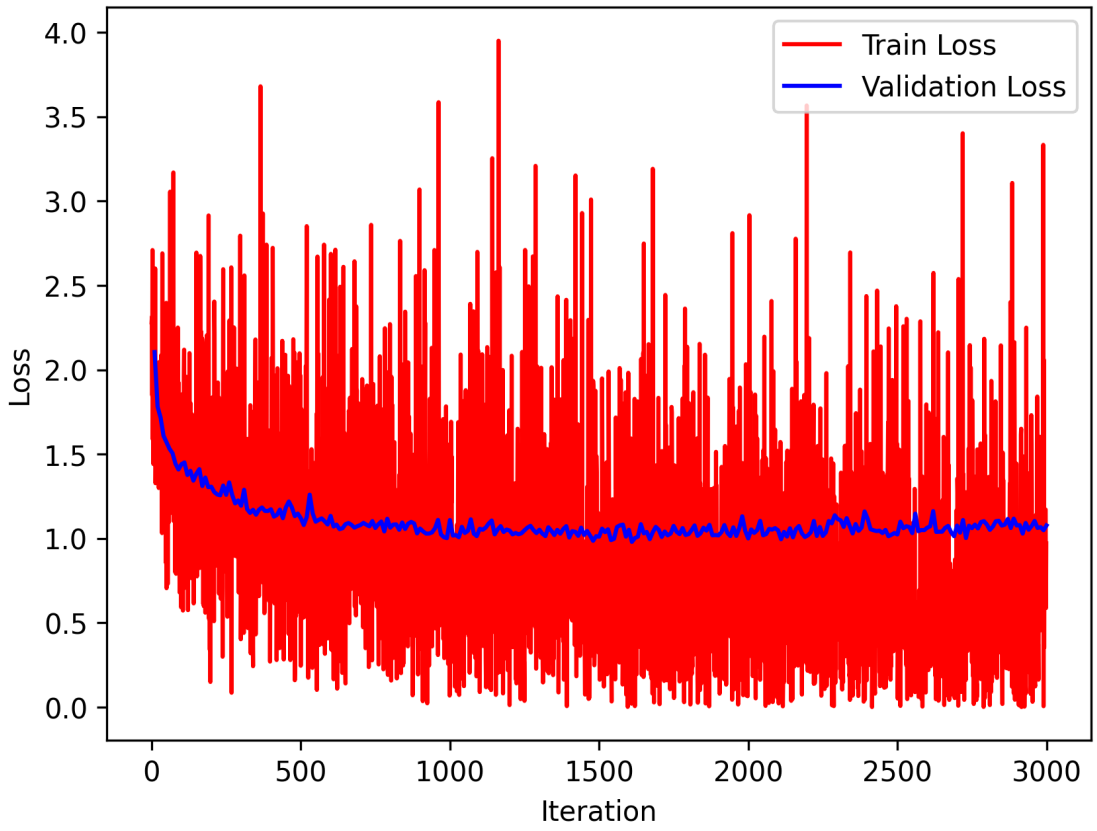
One immediate observation was that the **training time depended heavily on the batch size** . The training time across batches almost halved from 4 to 8 and 8 to 16 .. and so on. The following were the plots obtained :

Note: that iteration is measured after 50 batches. So total iterations = (50k * epochs) (50 * batchsize) = (13 k / batchsize).

Similarly after 500 batches vaildation accuracy was calculated and one fifth of points were chosen so index values in cross validation curves are about (**13k/(5 *10 * batchsize)**) = (260 / batchsize) .

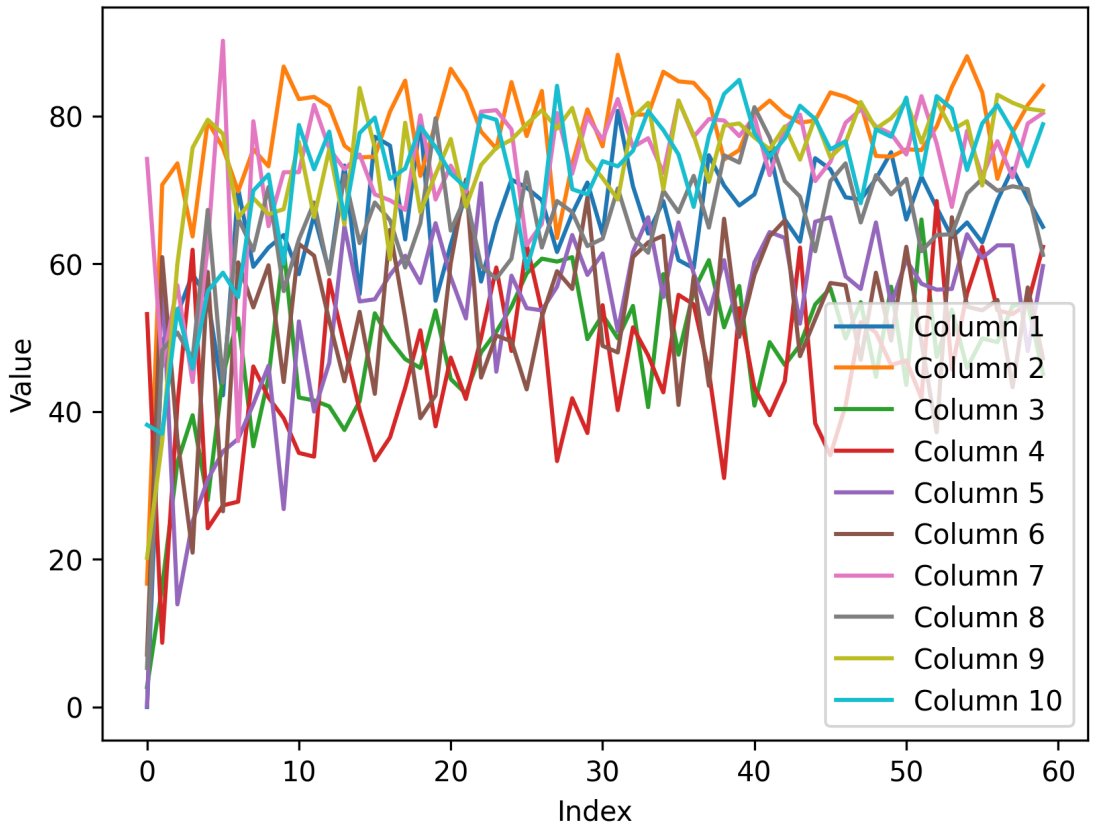
Train Loss and Validation Loss

Iterations =13k / 4 ~ 3000

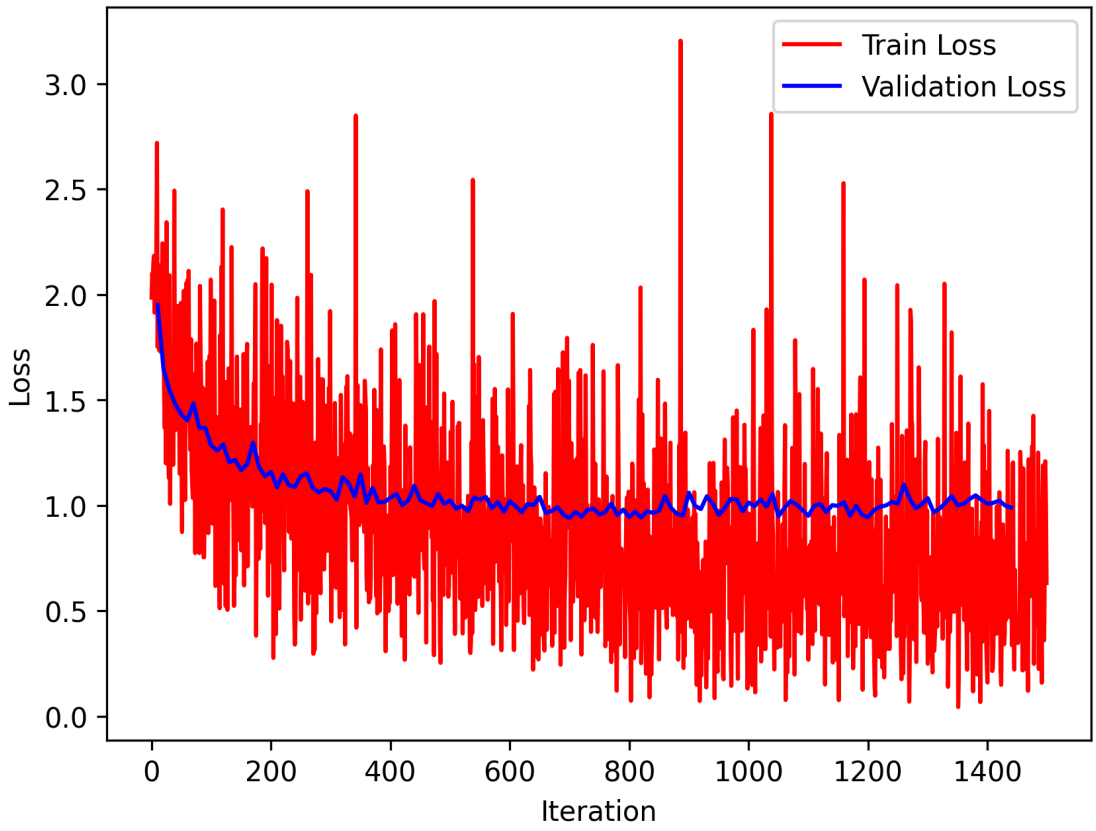


Class wise Validation accuracy

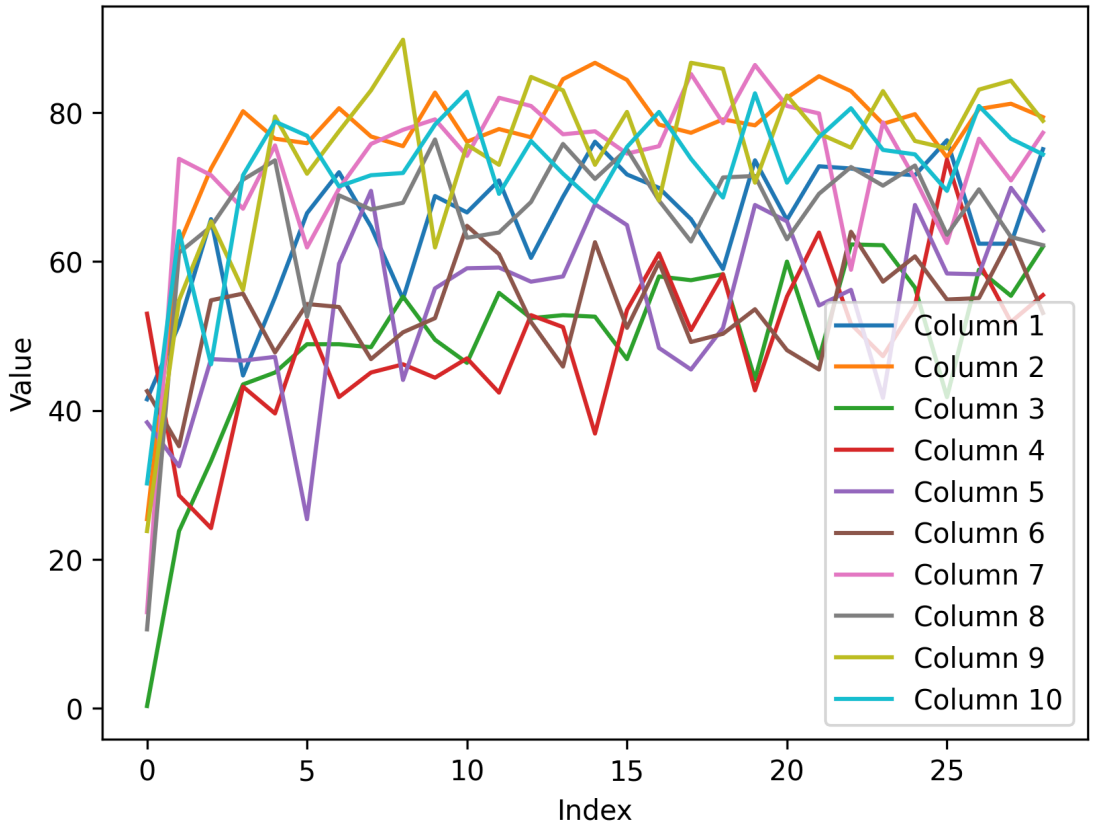
Index = 260/4 ~ 65



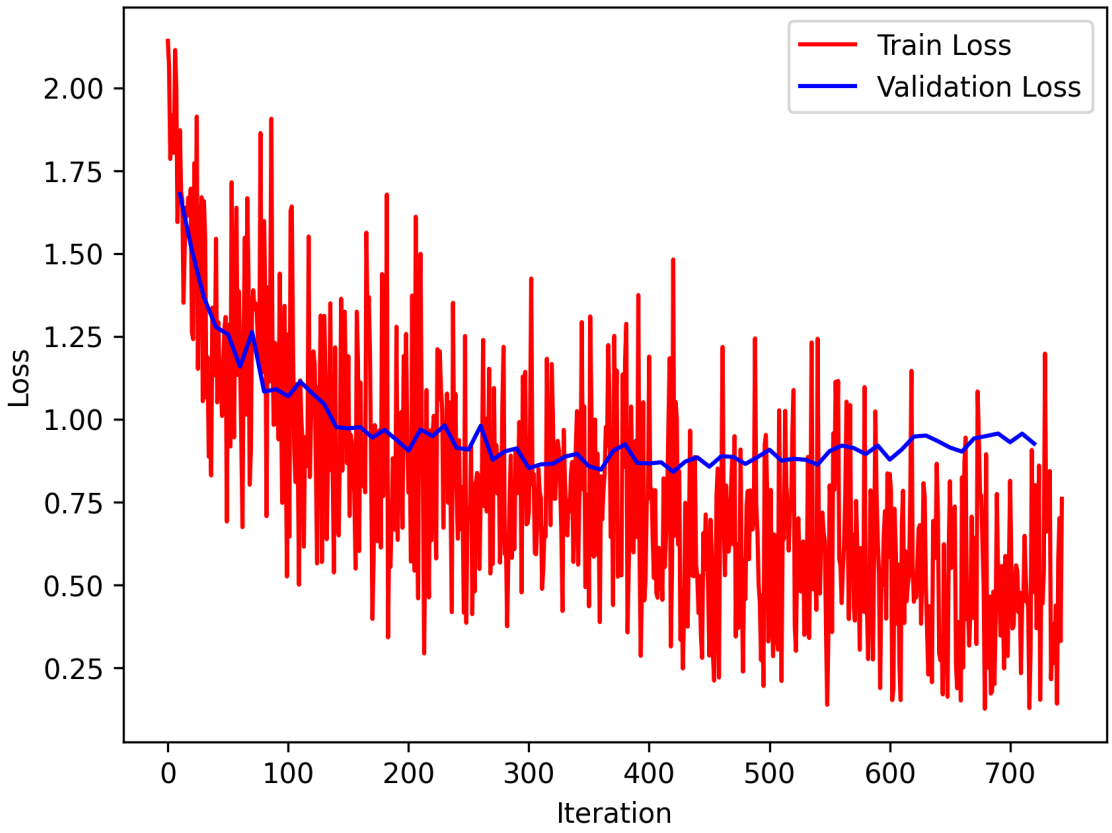
Train Loss and Validation Loss



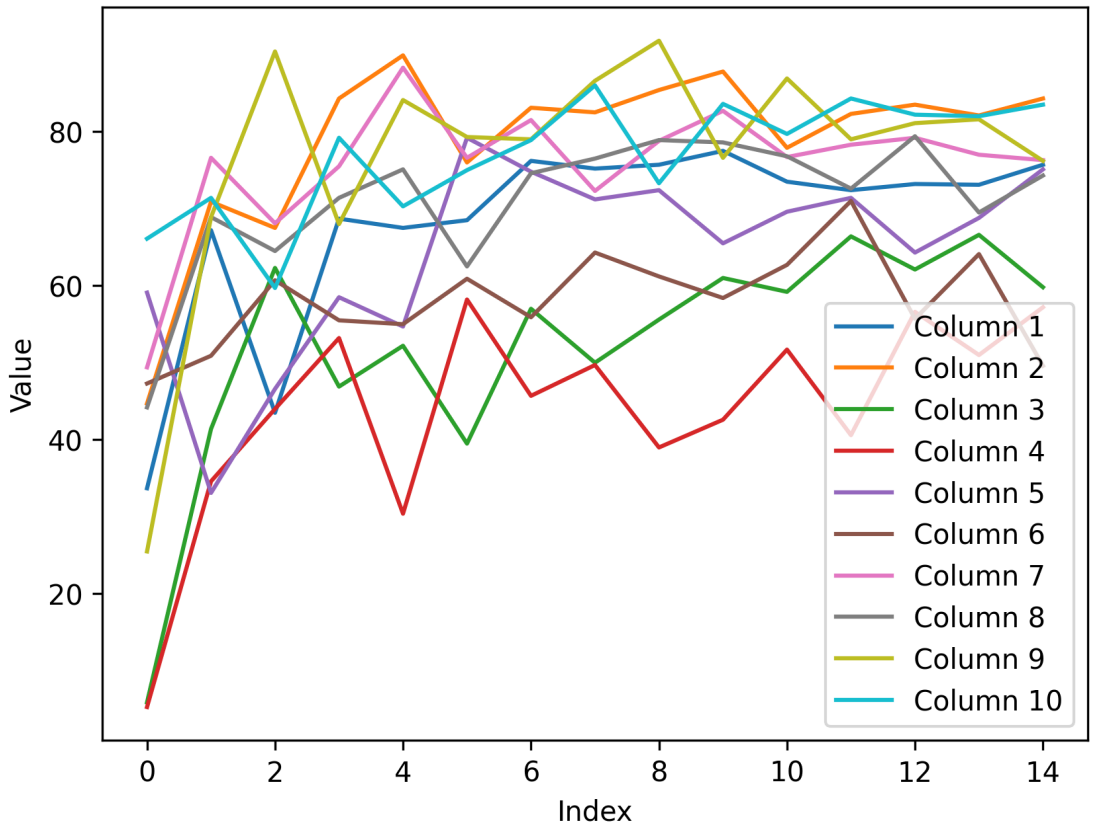
Class wise Validation accuracy



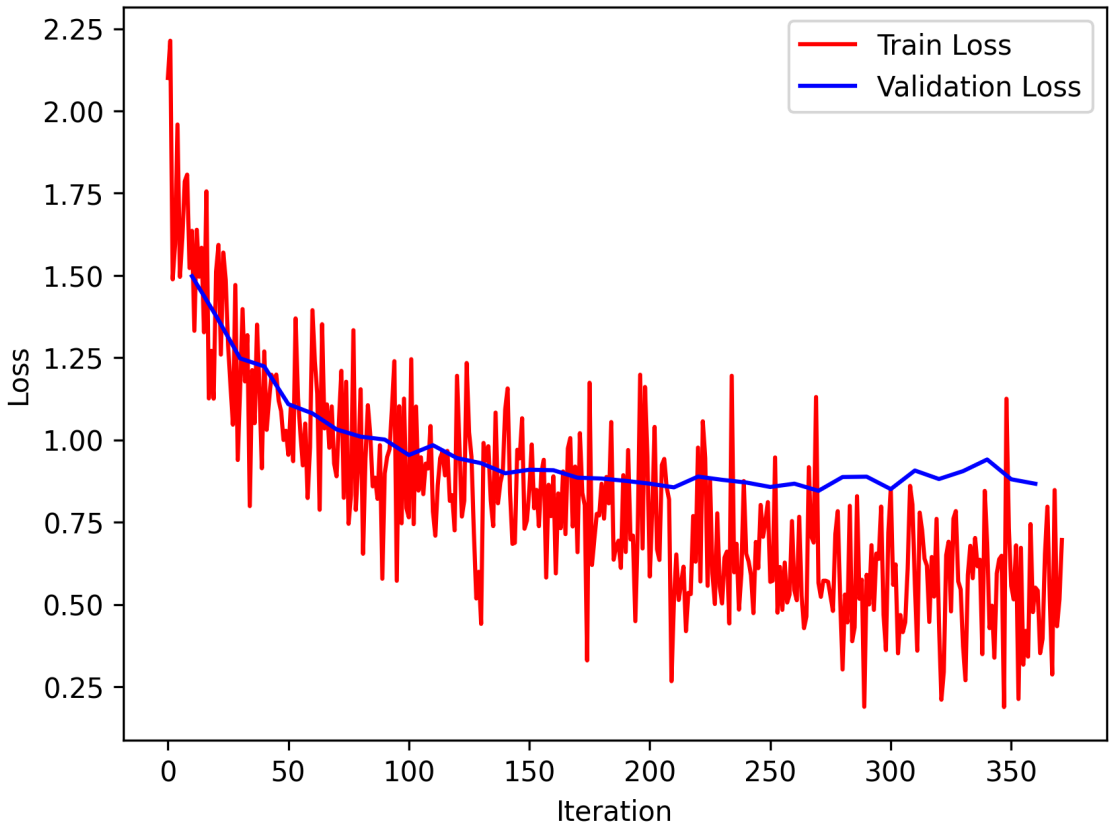
Train Loss and Validation Loss



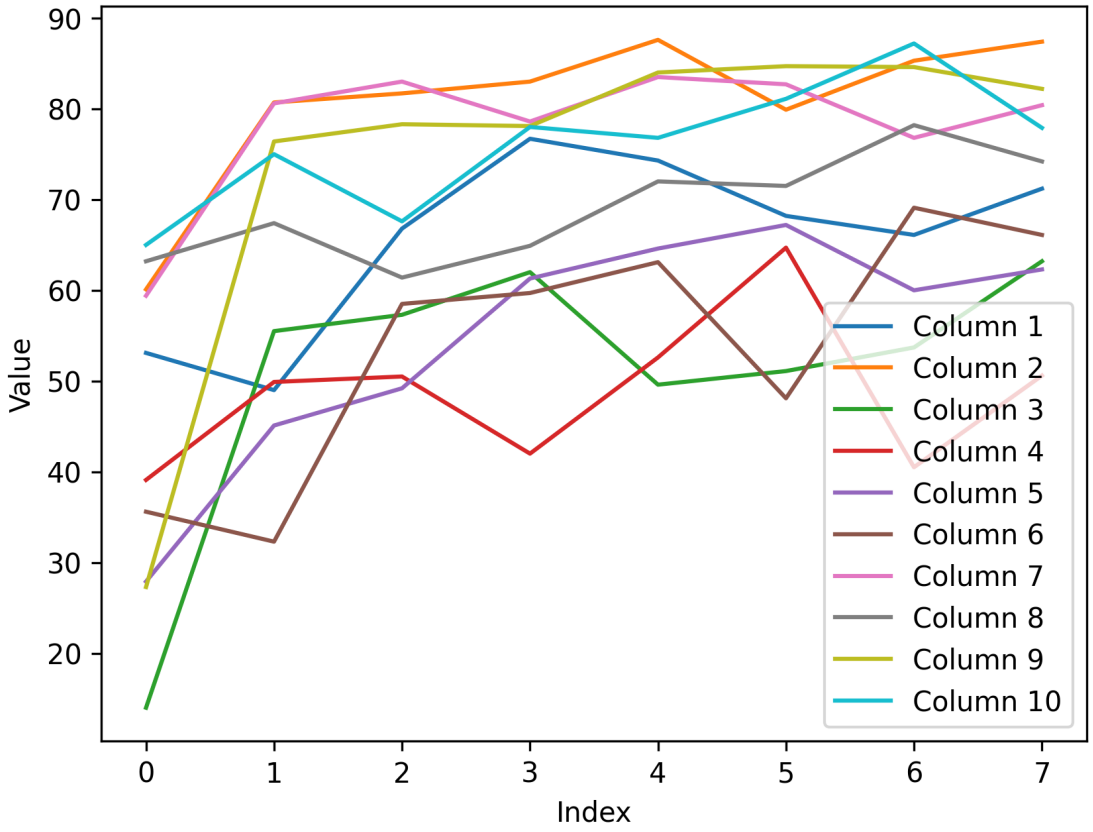
Class wise Validation accuracy



Train Loss and Validation Loss



Class wise Validation accuracy



Analysis on Varying batch size

One important observation is that when the batch size becomes larger , the training loss on each batch becomes more and more uniform and we can see that we get rid of the large variation in training loss that can be seen when batch size is 4,8.

We can also see that the model converges to its' minimum validation accuracy in lesser number of epochs when the batch size is small , however this minimum is not the optimal and when the batch size increases we see lower minimas. This is because a lower batch size means that we learn the output errors from that small batch and improve the model according to that, this leads

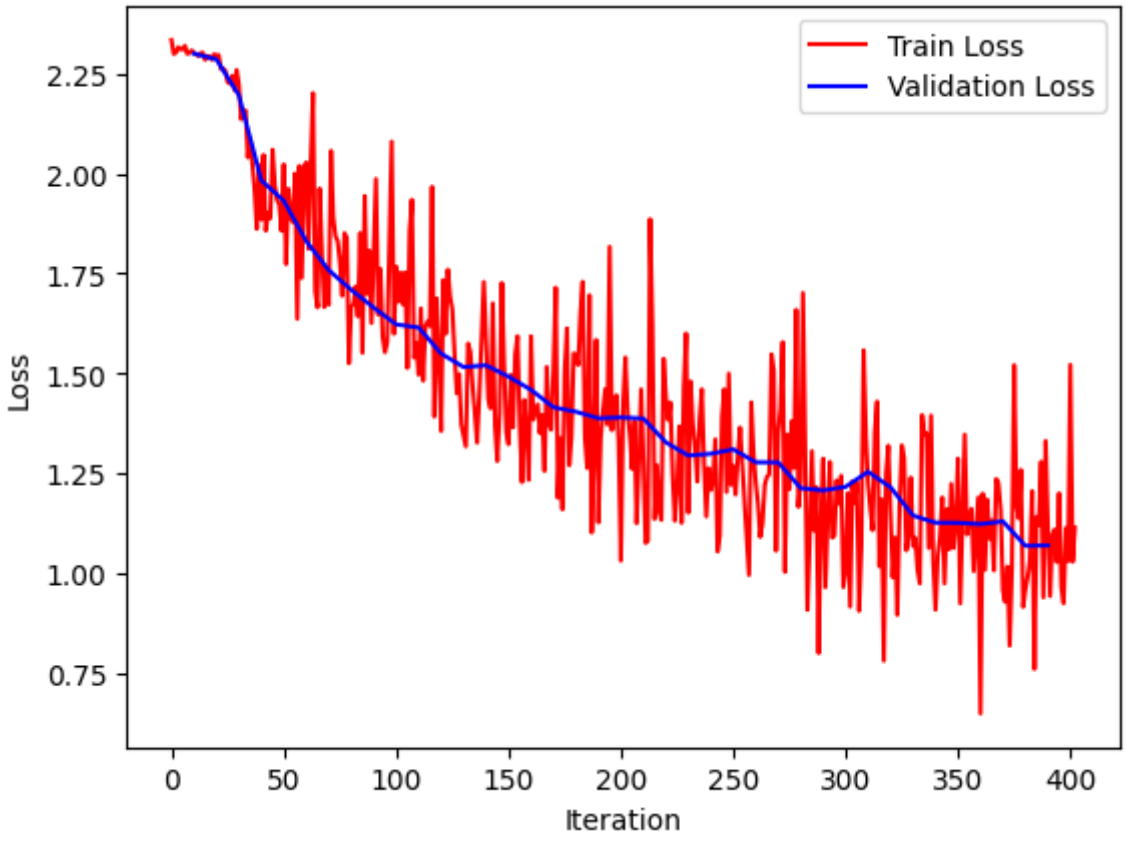
In general Larger batch sizes improves the accuracy of classes with more examples, while decreasing the accuracy of classes with fewer examples. This is because the larger batch size allows the model to better learn the patterns in the more common classes, while the smaller classes may not be well represented in each batch. In the CIFAR1-10 dataset all the classes have the same number of examples, hence we see that we obtain highest class-wise validation accuracy when the batch_size is 32.

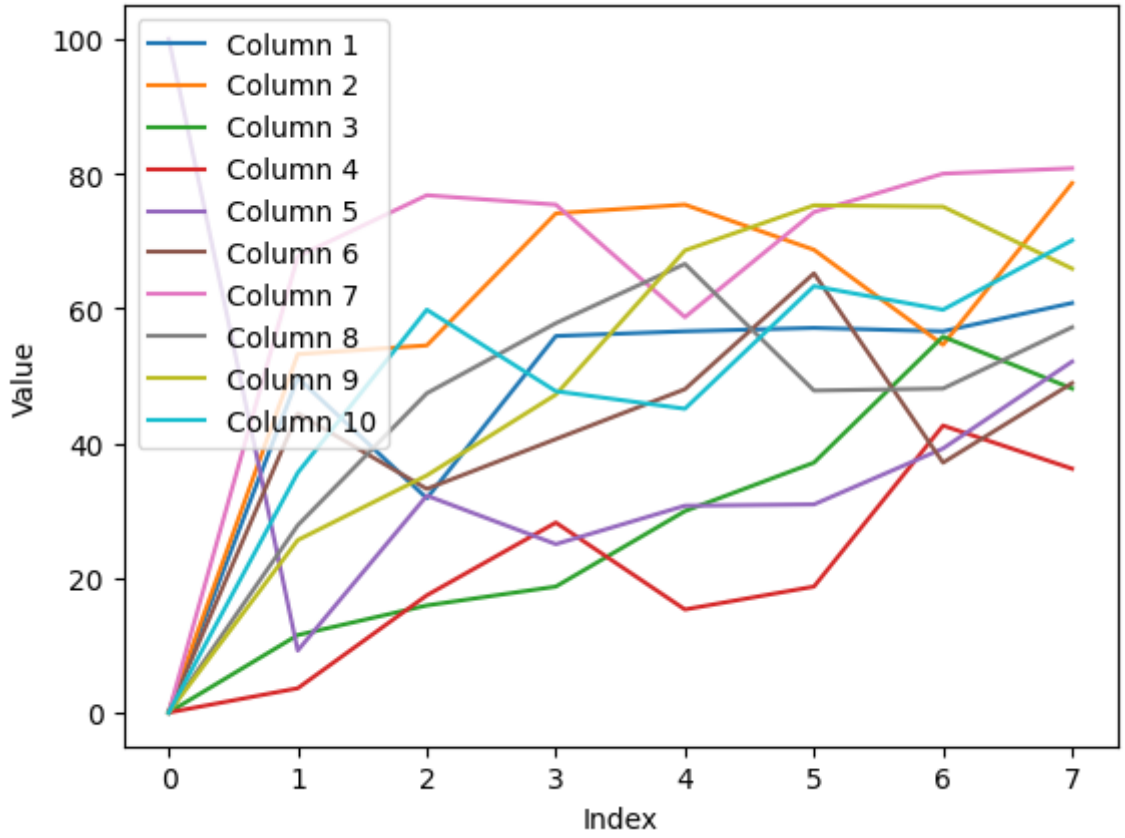
Overall best parameters

From the above analysis , I found out that learning rate=0.001, epochs=13, batch_size=32 with cosine annealing scheduler gave me good results.

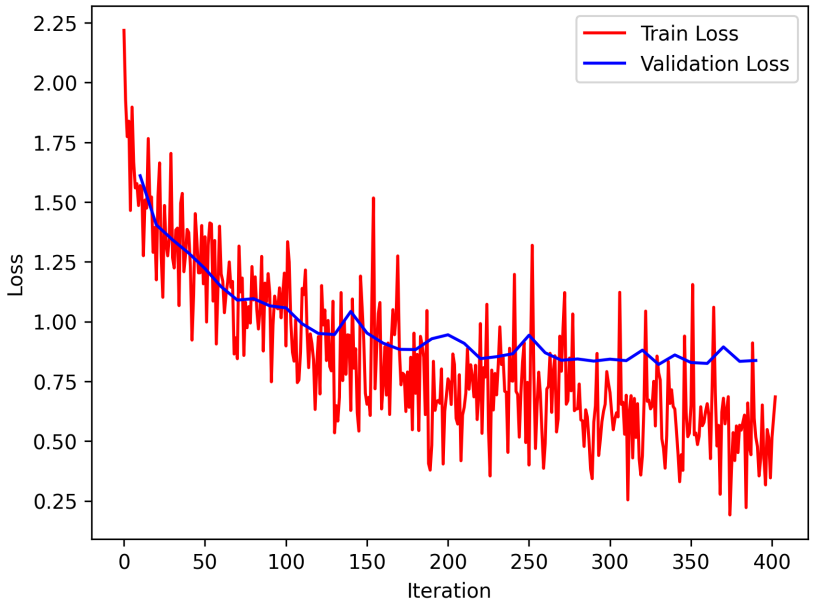
Varying Optimiser : SGD Optimiser

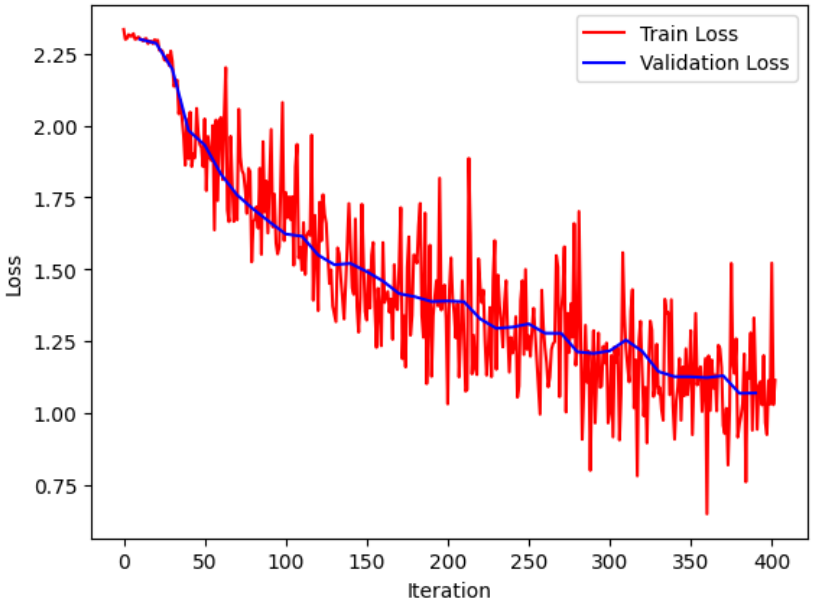
I use learning rate = 0.001, epochs=13 and SGD optimiser (momentum=0.9) and Cosine Annealing scheduler.
I obtained the following plot Training vs Validation Loss curve :


Train Loss
Validation Loss
Loss
Iteration
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0
50
100
150
200
250
300
350
400


Column 1
Column 2
Column 3
Column 4
Column 5
Column 6
Column 7
Column 8
Column 9
Column 10
Value
Index
100
80
60
40
20
0
0
1
2
3
4
5
6
7

Comparing with Adam Optimiser on the same parameters :



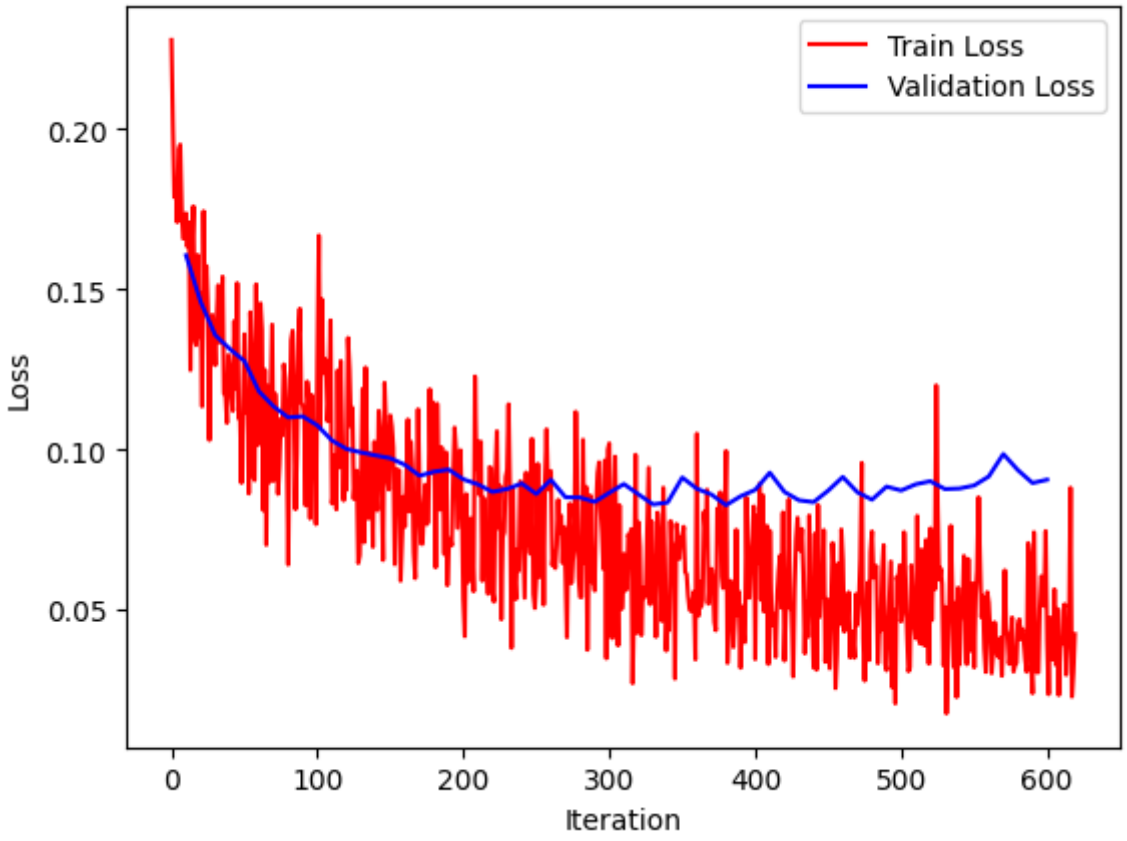


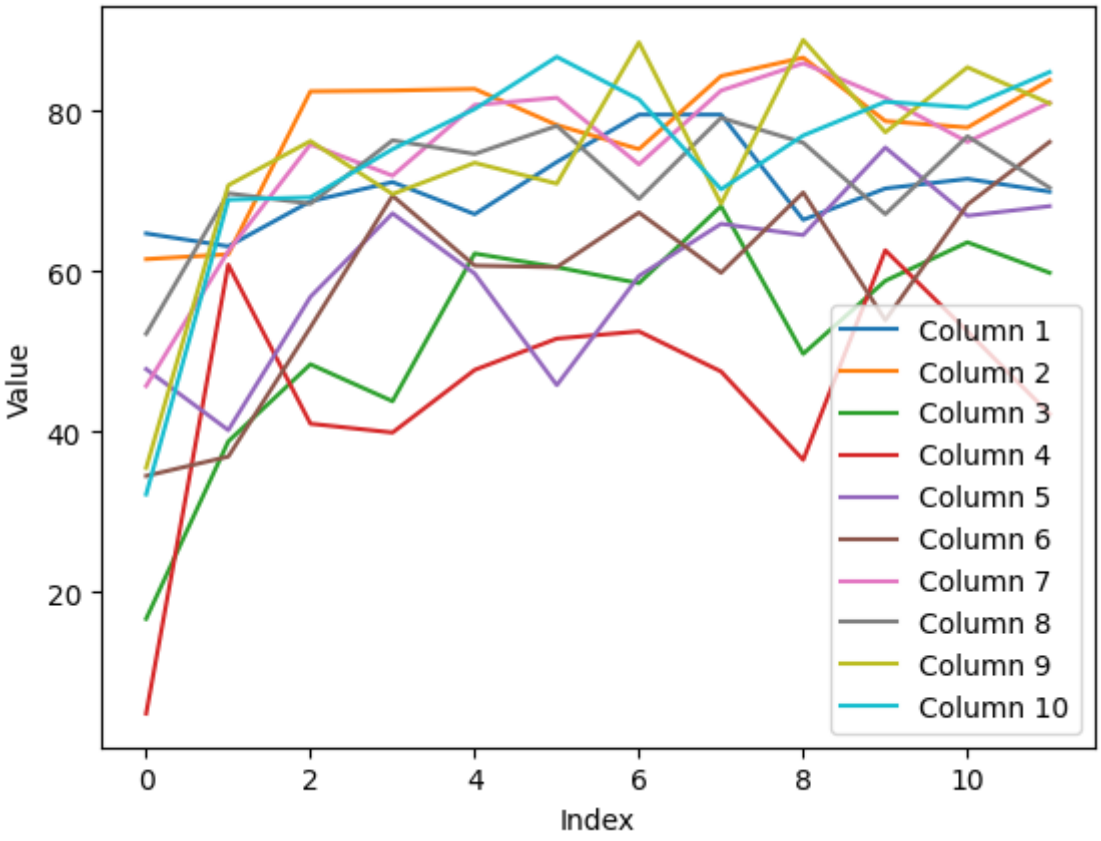
On the left is ADAM optimeser and right is SGD .

The ADAM optimizer adapts the learning rate for each parameter based on its gradient variance and mean, while the SGD optimizer uses a fixed learning rate. Therefore we can see that the adam optimiser converges faster to the minima for the target. We can see that the ADAM optimiser is able to reach minima for validation loss till iters 200 while SGD is still decreasing till iteration 400.

4.2 Varying Loss Function : KL Divergence Loss

I used one hot encoding on my labels and softmax layer on output and then computed KL Divergence loss and obtained the following results.





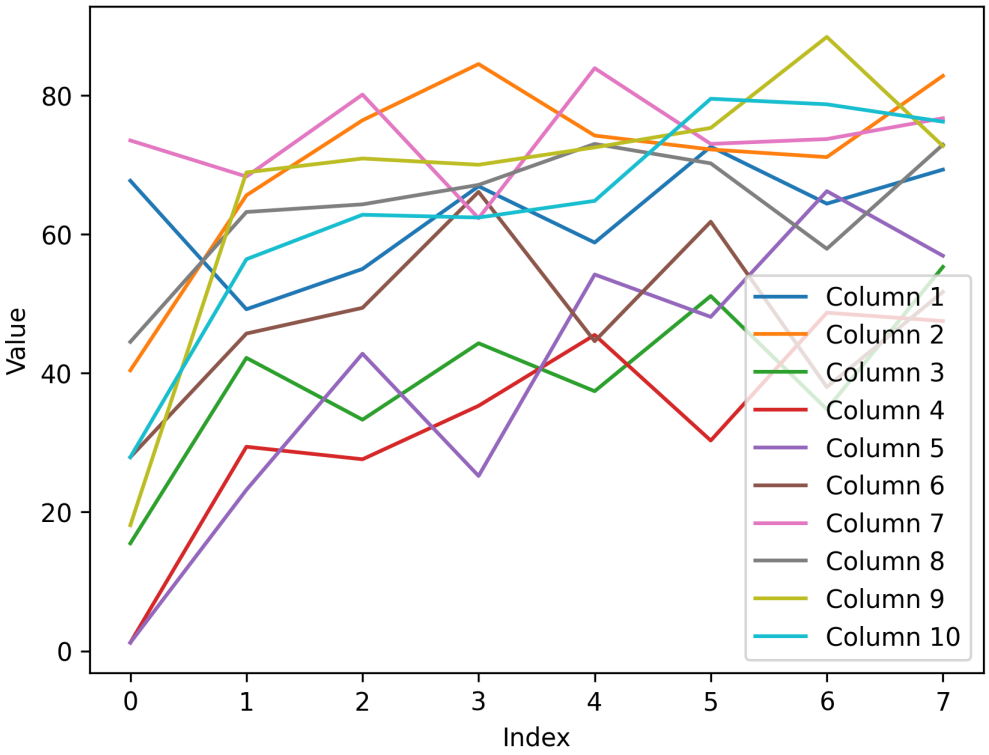
Analysis on changing loss function

We can see that KL Divergence loss reaches convergence faster than cross entropy loss, however the class wise accuracy remains lower and the overall accuracy also remains lower. This suggests that cross entropy loss is a better loss fucntion for multi-class classification. KL divergence loss is supposed to perform better for unsupervised learning tasks.

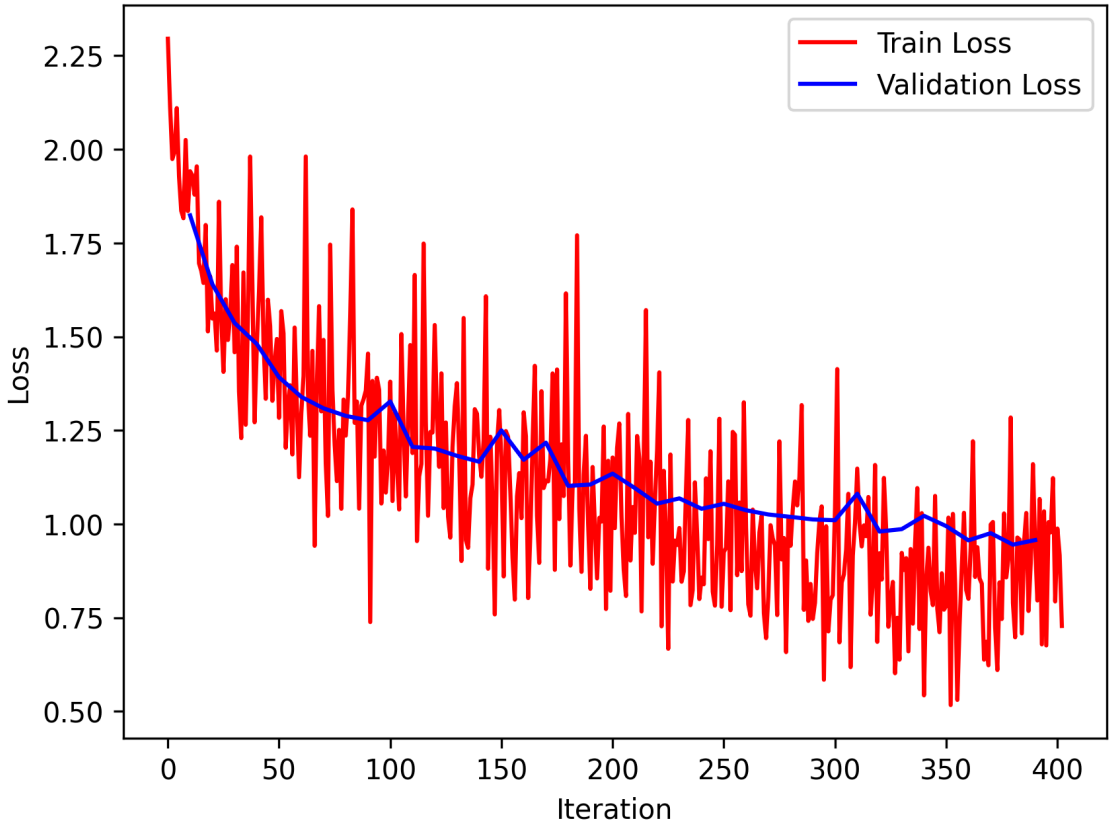
4.3 Effect of Data Augmentation :

I obtained the following curves after removing data augmentation (which was normalisation to mean 0.5 and std deviation 0.5). I have also plotted the training accuracy:

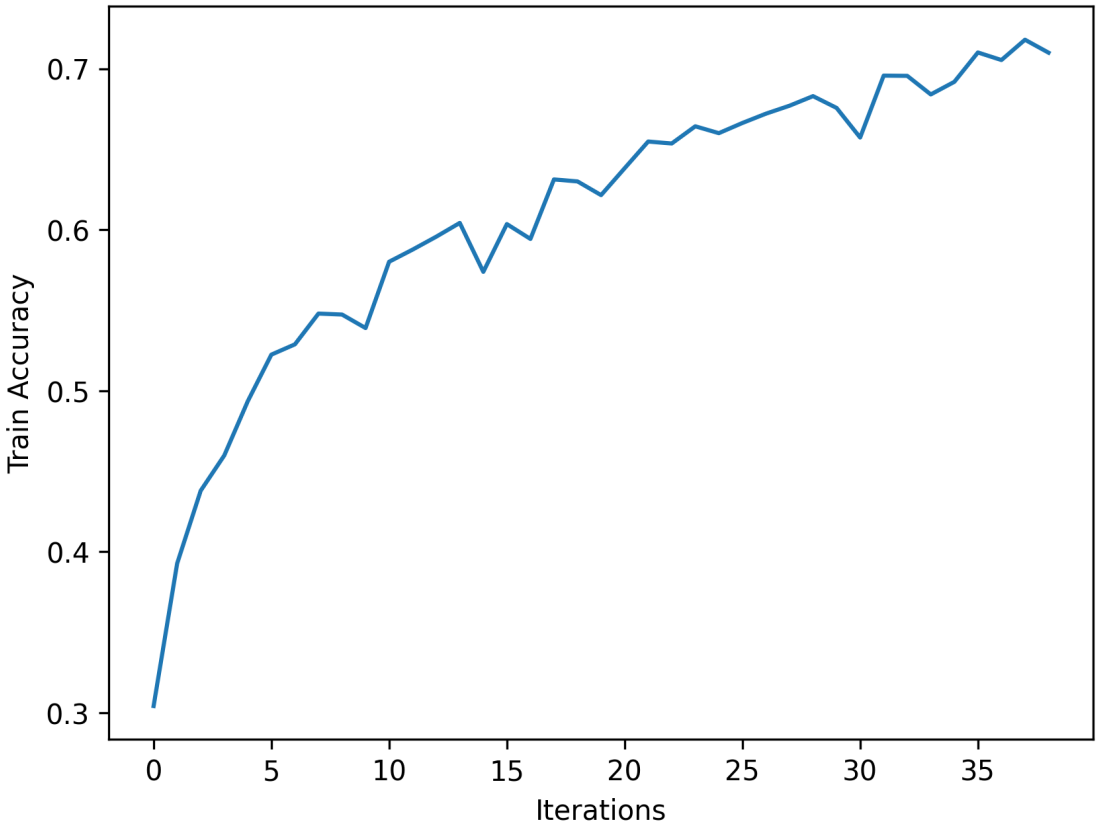
Class-wise Validation Accuracy



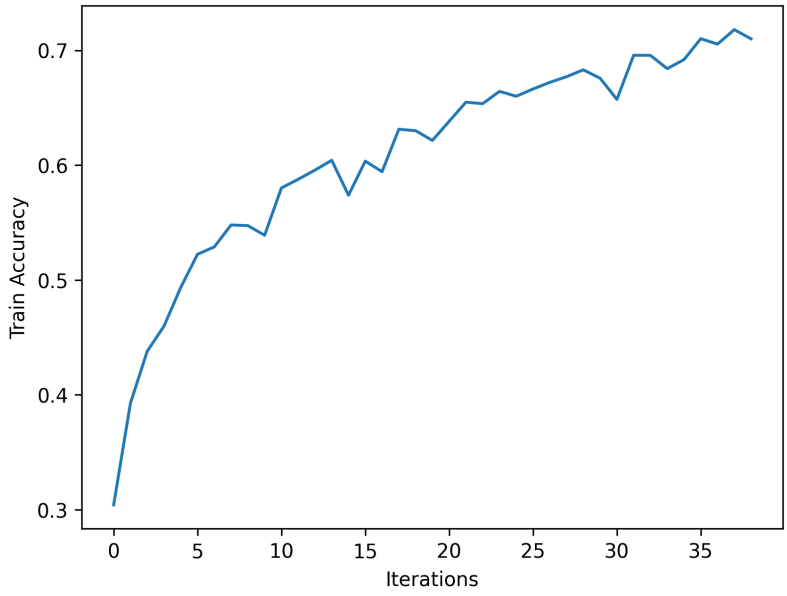
Training and Validation Loss

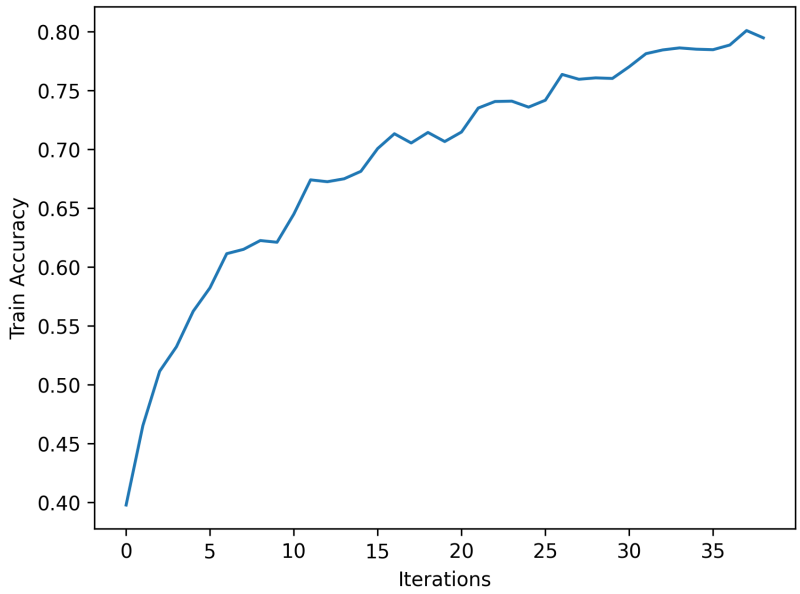


Train Accuracy



Comparison of class-wise training accuracy with Data Augmented model :





Analysis on Effect of Data Augmentation

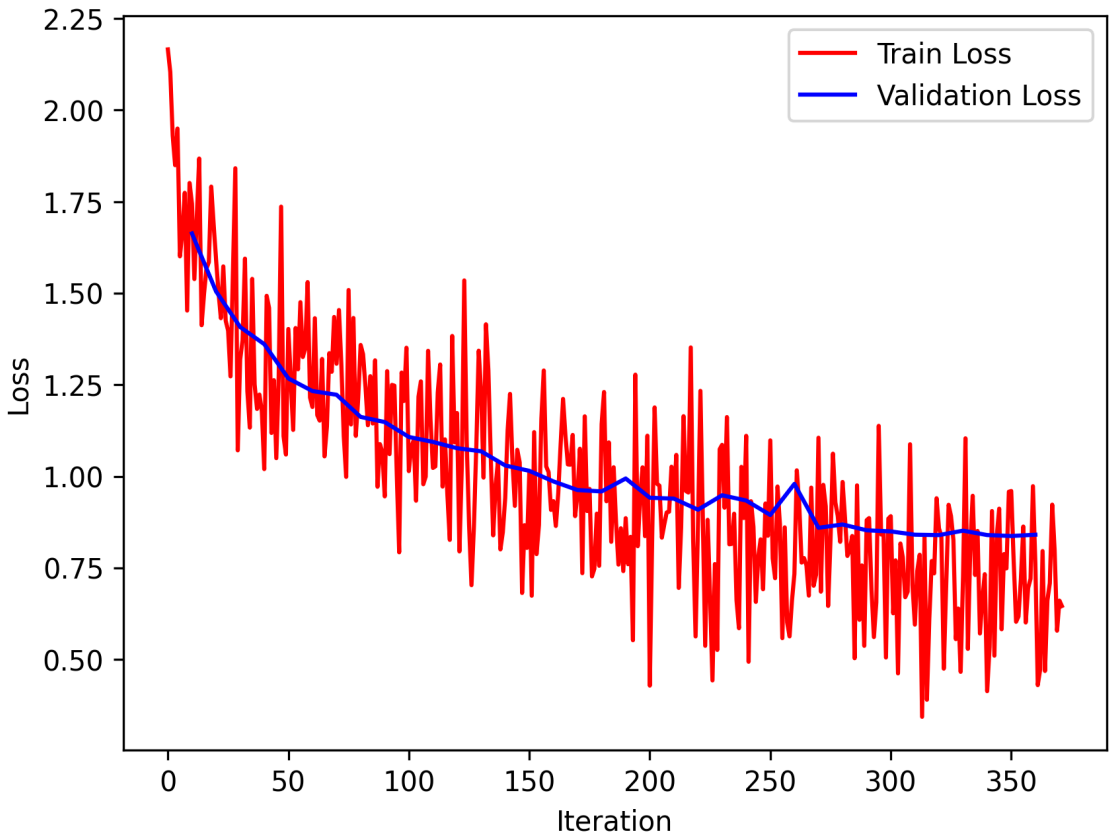
On the left is the model that does not have any data augmentation, while the model on the right has data augmentation. It can be clearly seen that in the same number of iterations the model that has data augmentation tends to have higher train accuracy, this is because augemntatino makes the training set more randomised and well distributed. Using the same set of images and the same labels we are able to learn much more things , this is because we are learning the correct features and not just spurious features. For eg. a car is labelled a car not because of its orientation and that it has four supports but because of its shape and structure.

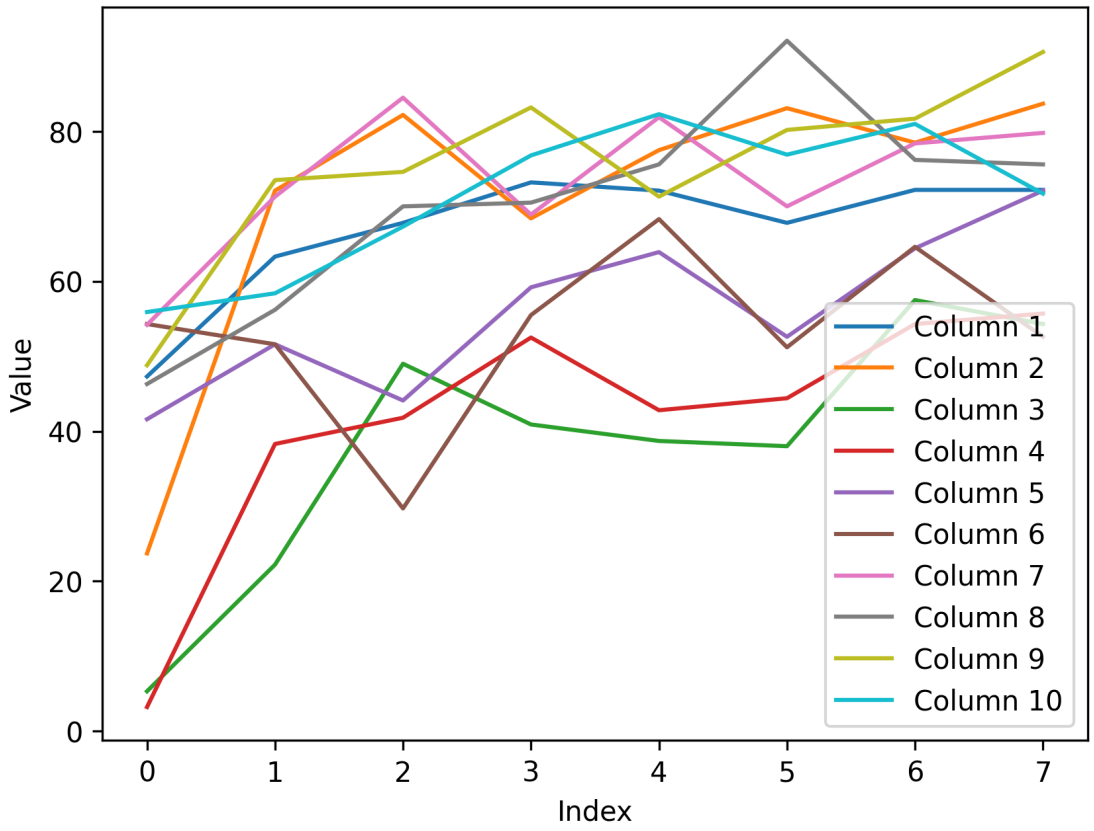
5 : Part 3 Improved CNN model

I made the following changes in the neural network

1. Added dropout layers : Dropout layer randomly sets some specified (in my case 25%) neurons in the layer to 0. This helps the model to learn the more redundant features that occur less across the dataset, otherwise the model just keeps learning those features that occur in abundance. The model starts to learn more distributed features. By dropping out we tend to generalise as the lesser present features are now the focus of the model. It makes learning more robust and helps generalization to unseen data.
2. Applied Random rotation and random crop transform to my train dataset. These transforms help to make the training data more uniform and randomised , this prevents overfitting and brings more variation into the training data. The model should tend to generalise better.

I obtained the following results :





We obtain a very high overall accuracy with each class having more than 50% correct classfication and many classes have above 80% valdiation accuracy. The overall accuracy was 74.23%

References

- To understand implementation structures and architecture of a CNN I referred to the following youtube videos :
  - https://www.youtube.com/watch?v=Lakz2MoHy6o&t=1769s&ab_channel=TheIndependentCode
  - https://www.youtube.com/watch?v=pauPCy_s0Ok&t=889s&ab_channel=TheIndependentCode
  - https://www.youtube.com/watch?v=tIeHLnjs5U8&t=34s&ab_channel=3Blue1Brown
  - https://www.youtube.com/watch?v=tIeHLnjs5U8&ab_channel=3Blue1Brown
  - Soumen Basu sir's lecture and html notebook on using pytorch