COL341 Assignment 4

Kushagra Rode (2020CS10354)

April 2023

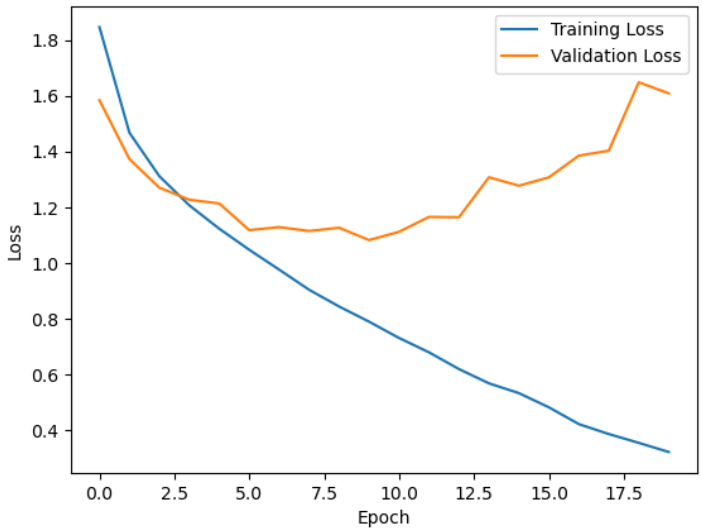
Note :

I have used Google Colab, Kaggle, and my own laptop for the various sections done below. My submission includes 3 files **nn_scratch.ipynb** which contains my own scratch implementation, **nn_pytorch.ipynb** which contains the code which I used for doing analysis and generating plots for section 4 and then **nn_improv.ipynb** which includes the improved neural network using pytorch. The path to the data in my scratch implementation is hardcoded according to my directory structure, and so should be changed accordingly while being run. It also assumes that a folder for cifar-10 dataset has already been extracted and is present in the same directory as the file.

Implement a Neural Network

I made classes for each of the layers used in the Neural network, the classes are Conv2D, MaxPooling2D, Linear, Relu and Softmax. For maxpooling, I have used stride as 2(equal to the kernel size). The optimiser used is Adam's optimiser with Categorical cross-entropy loss. The main class is the CNN class. This class is responsible for the construction of the complete neural network architecture. It has its own forward and backward methods which make use of the forward and backward methods of each of the layers. This class also has train function which uses mini-batch gradient descent with the adam's optimizer to compute the final weights and biases for the layers concerned. I tried to vectorise as much as I could and was able to bring down the running time to a significant amount of time. But still for the given data and the number of epochs it would take about 10 hours for complete training. Neverthless, the analysis is being presented on the model trained on 15000 images(rather than 50000), the number of epochs is 20, the learning rate is 0.001 and the batch size is 32. I have made use of Xavier's initialization for initializing my weights. The loss calculated is the batch mean. Basically, after each epoch I divide the cumulated loss by the number of batches to get the average

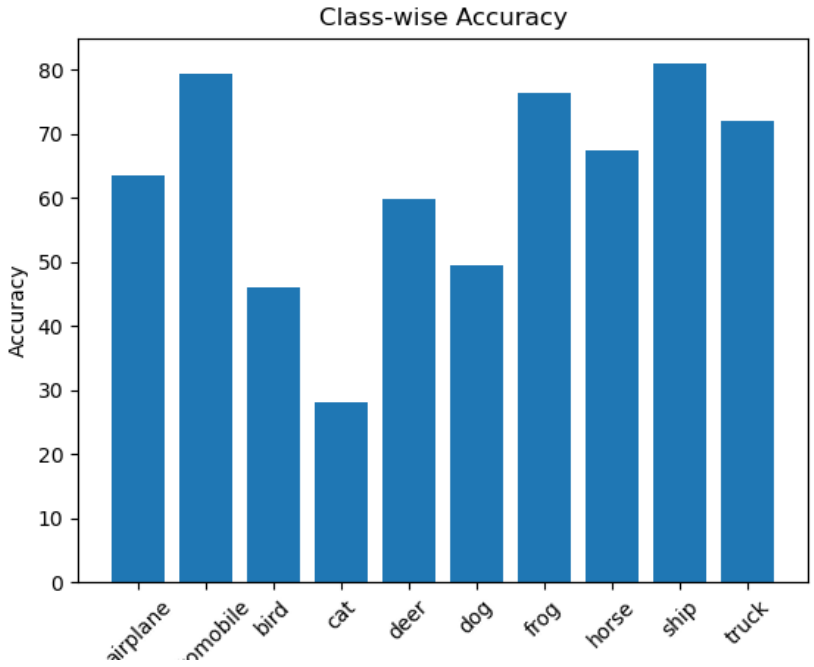
loss over the whole dataset. Following are the loss curves for both training and validation.



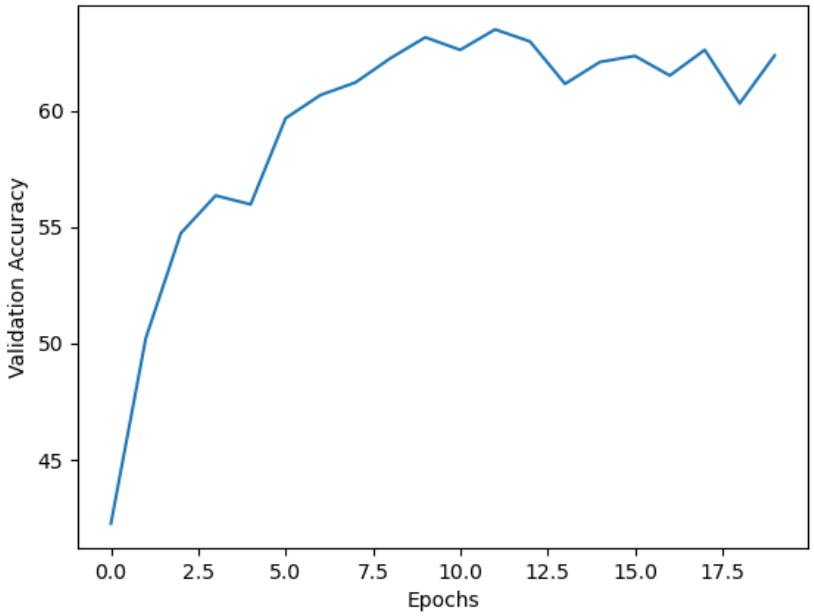
The class-wise accuracy is as follows :-

1. airplane - 63.52
2. automobile - 79.42
3. bird - 46.23
4. cat - 28.09
5. deer - 59.76
6. dog - 49.59
7. frog - 76.67
8. horse - 67.47
9. ship - 80.95
10. truck - 72.12

For better visualization, I have also plotted a bar plot for the class-wise accuracies



Following is the variation of validation accuracy with the number of epochs



The final validation accuracy comes out to be about 61%

Implement a PyTorch-based Solution

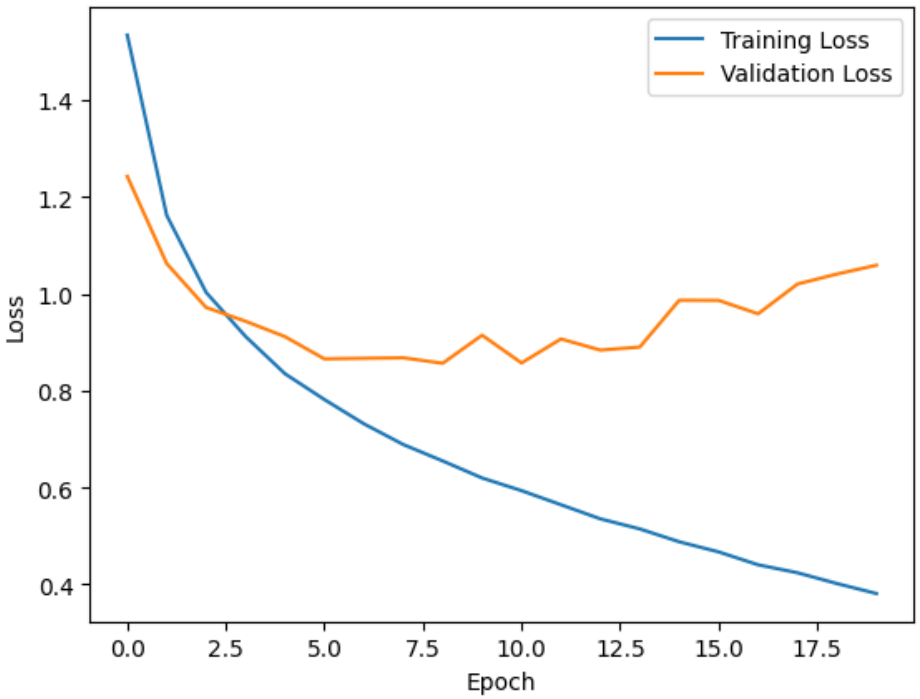
Hyper-parameter Tuning

0.0.1 Learning Rate

I have used 20 epochs with batch size as 32 for the analysis. First, I used Adam's optimizer to get the best learning rate and then I will use SGD optimizer to compare them. The data augmentation is basically normalization, I have used 'transforms.Normalize' function to normalize the pixel values of the images in the dataset.

1. Learning Rate - 0.001

   Following are the loss curves for both training and validation.



   The class-wise accuracy is as follows :-

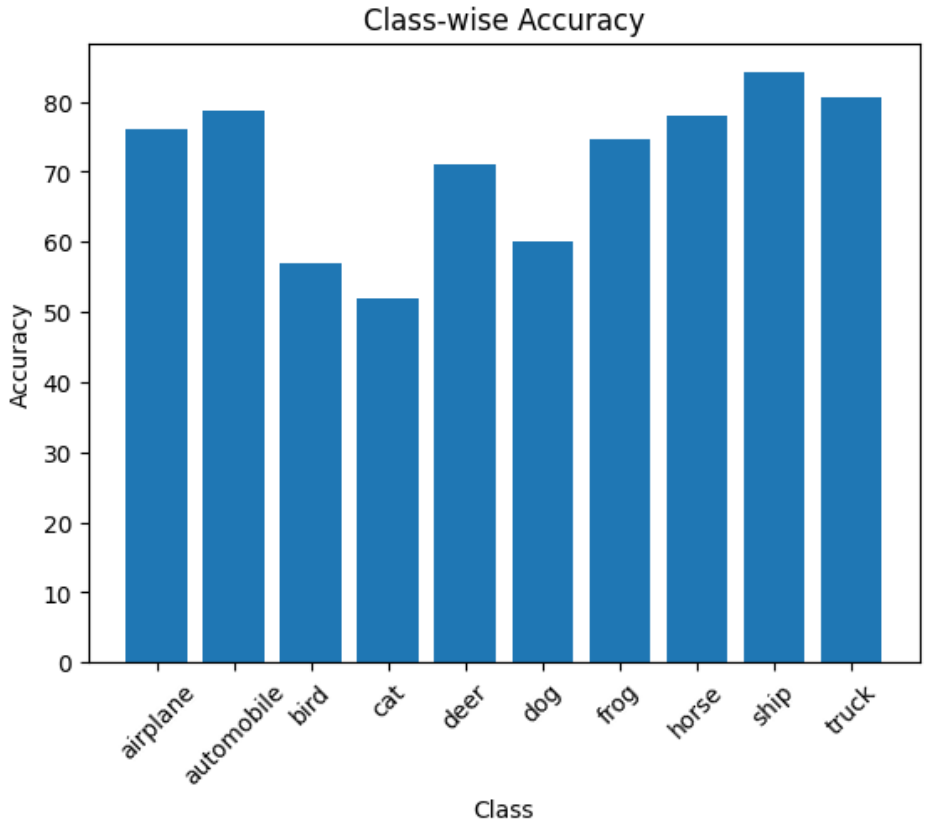
   (a) airplane - 76.15
   (b) automobile - 78.59
   (c) bird - 56.84
   (d) cat - 51.94
   (e) deer - 71.02
   (f) dog - 60.06
   (g) frog - 74.67

(h) horse - 77.88

(i) ship - 84.12

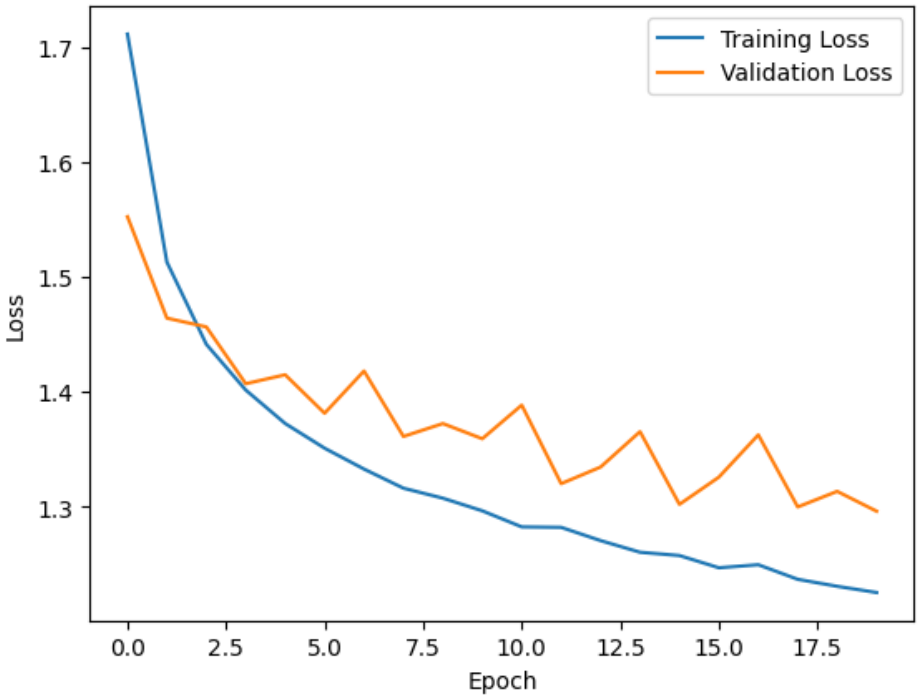
(j) truck - 80.625

For better visualisation, I have also plotted a bar plot for the class-wise accuracies



2. Learning Rate - 0.006

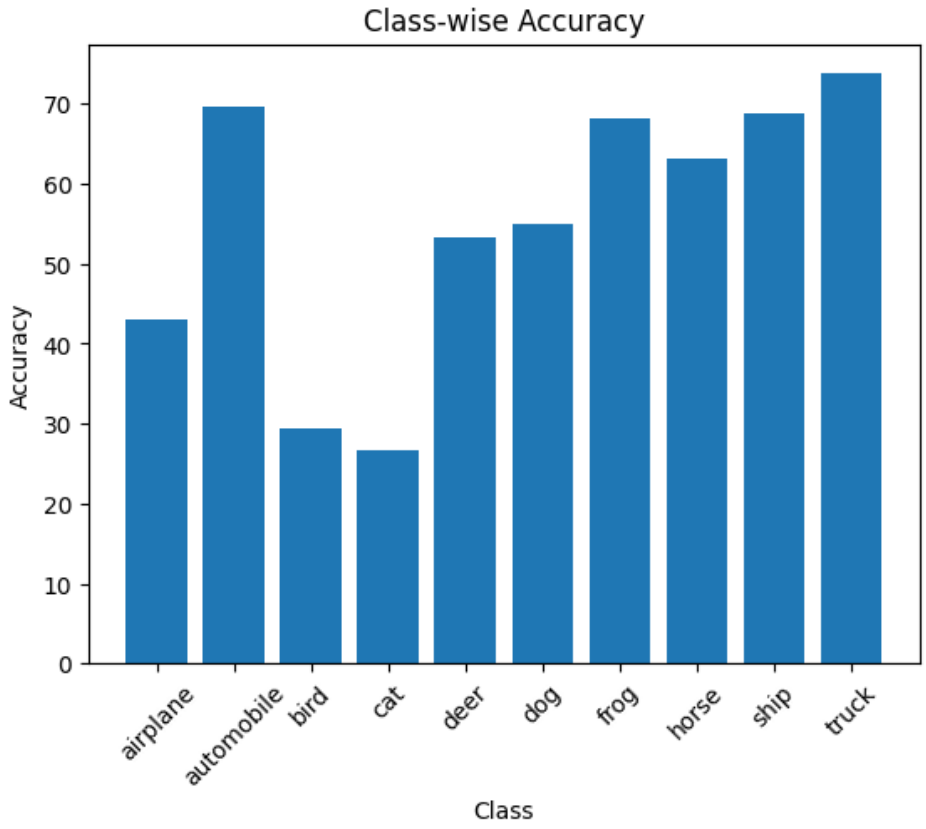
Following are the loss curves for both training and validation.



The class-wise accuracy is as follows :-

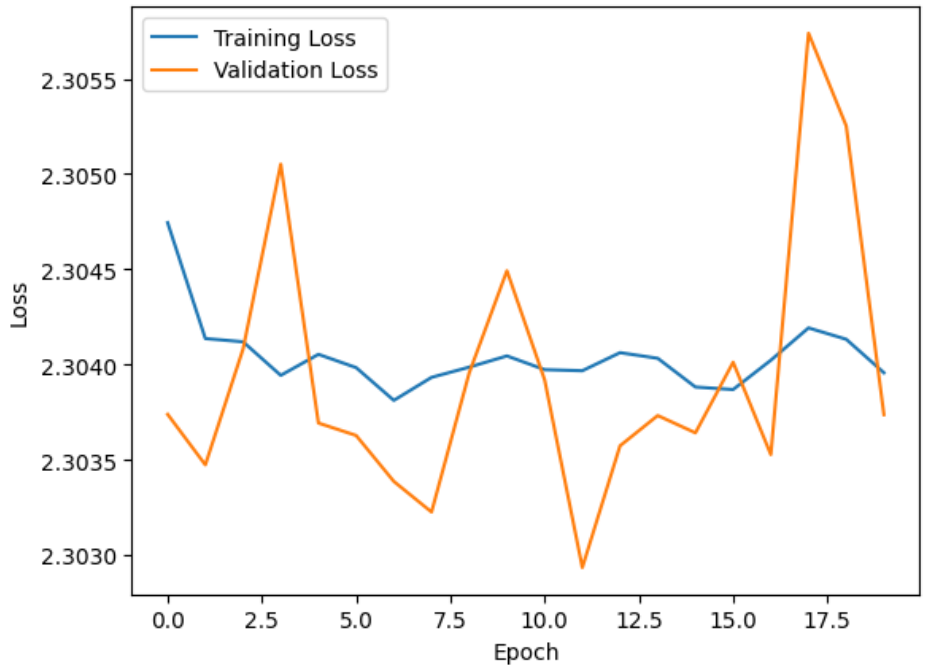
(a) airplane - 43.04
(b) automobile - 69.56
(c) bird - 29.46
(d) cat - 26.62
(e) deer - 53.27
(f) dog - 54.95
(g) frog - 68.09
(h) horse - 63.14
(i) ship - 68.88
(j) truck - 73.75

For better visualization, I have also plotted a bar plot for the class-wise accuracies



3. Learning Rate - 0.01

   Following are the loss curves for both training and validation.



   The class-wise accuracy is as follows :-

   (a) airplane - 100

(b) automobile - 0

(c) bird - 0

(d) cat - 0

(e) deer - 0

(f) dog - 0

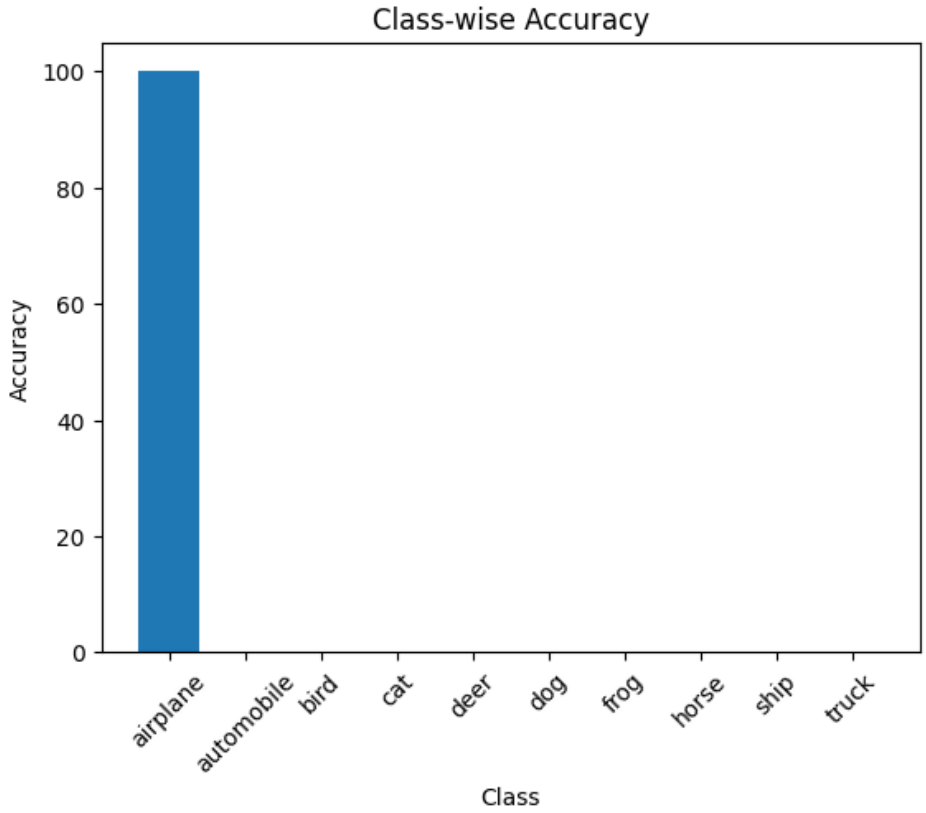
(g) frog - 0

(h) horse - 0

(i) ship - 0

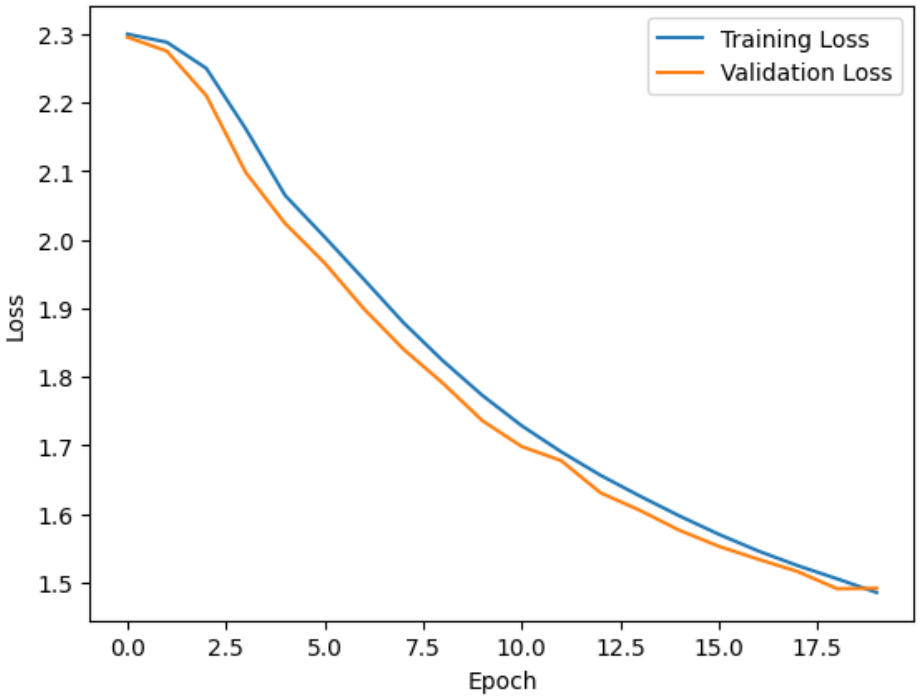
(j) truck - 0

For better visualisation, I have also plotted a bar plot for the class-wise accuracies



The learning rate of 0.001 seems to be the best for Adam's optimizer and hence I'll now train using the SGD optimizer using the same learning rate. The learning rate 0.001 provides the best class wise accuracies and also reduces the training and validation loss to a very low value

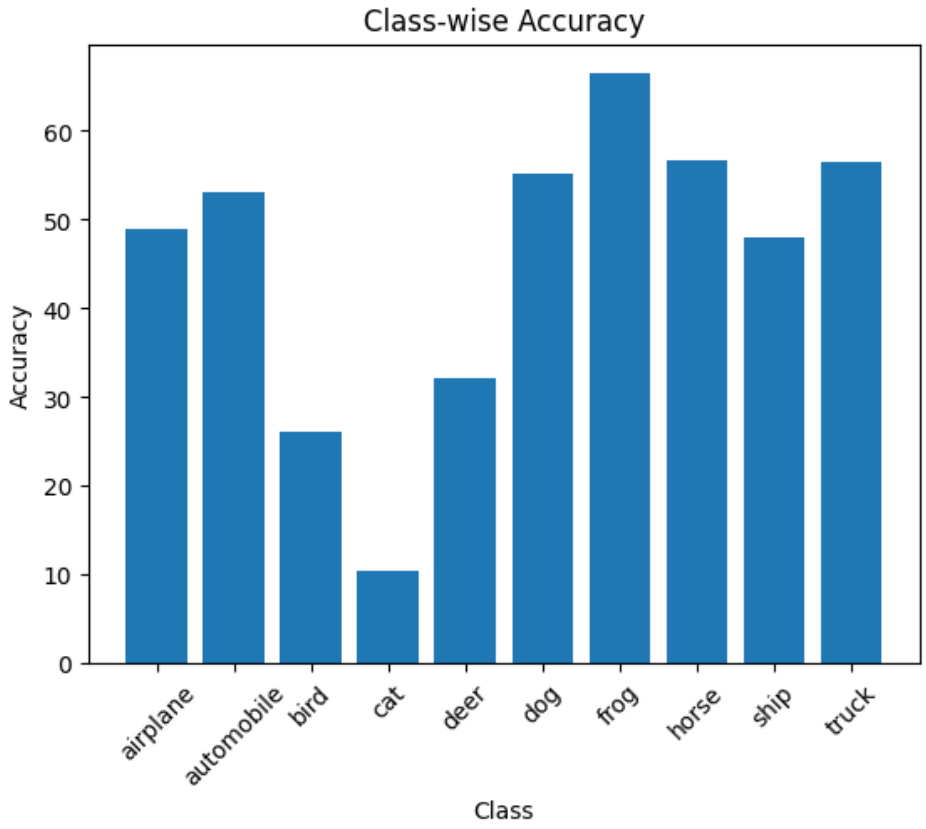
Following are the loss curves for both training and validation using SGD optimiser.



The class-wise accuracy is as follows :-

1. airplane - 49.006
2. automobile - 53.17
3. bird - 26.62
4. cat - 10.08
5. deer - 32.45
6. dog - 55.27
7. frog - 66.47
8. horse - 56.73
9. ship - 47.93
10. truck - 56.625

For better visualisation, I have also plotted a bar plot for the class-wise accuracies



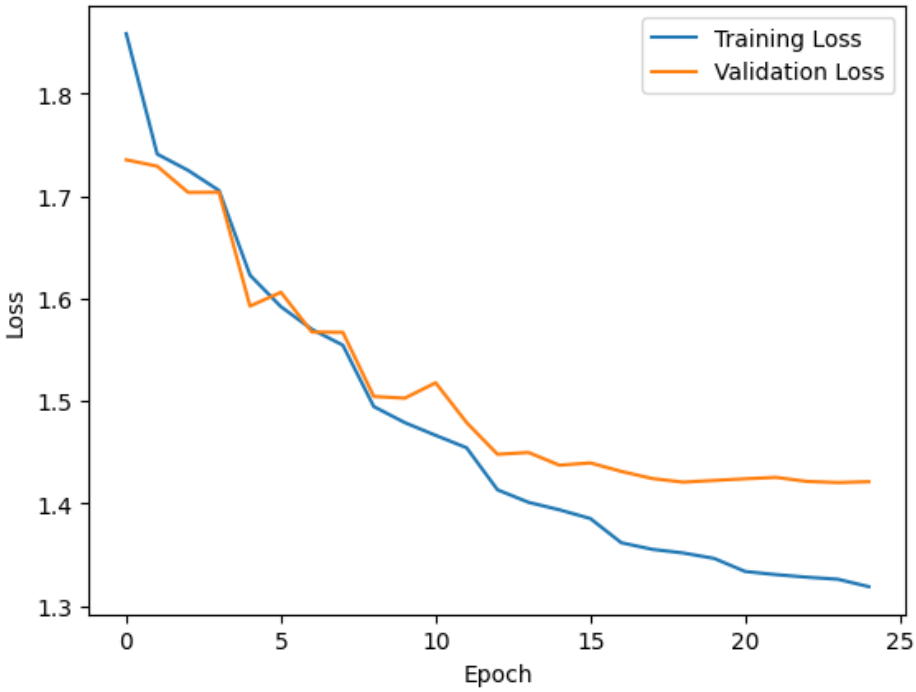
Clearly the Adam's optimiser seems to be working in a better way. Although, the sgd optimiser provides a smoother curve, for the given number of epochs Adam converges quickly than the SGD. The class wise accuracies are also very high for the Adam than SGD.

0.0.2 Variation in LR

I made use of 2 kinds of scheduler, the StepLR and the MultiStepLR. The number of epochs were set as 25 and the batch size was 32. The optimizer used was Adam.

1. StepLR (Step size - 4 and gamma = 0.5, initial learning rate = 0.01)

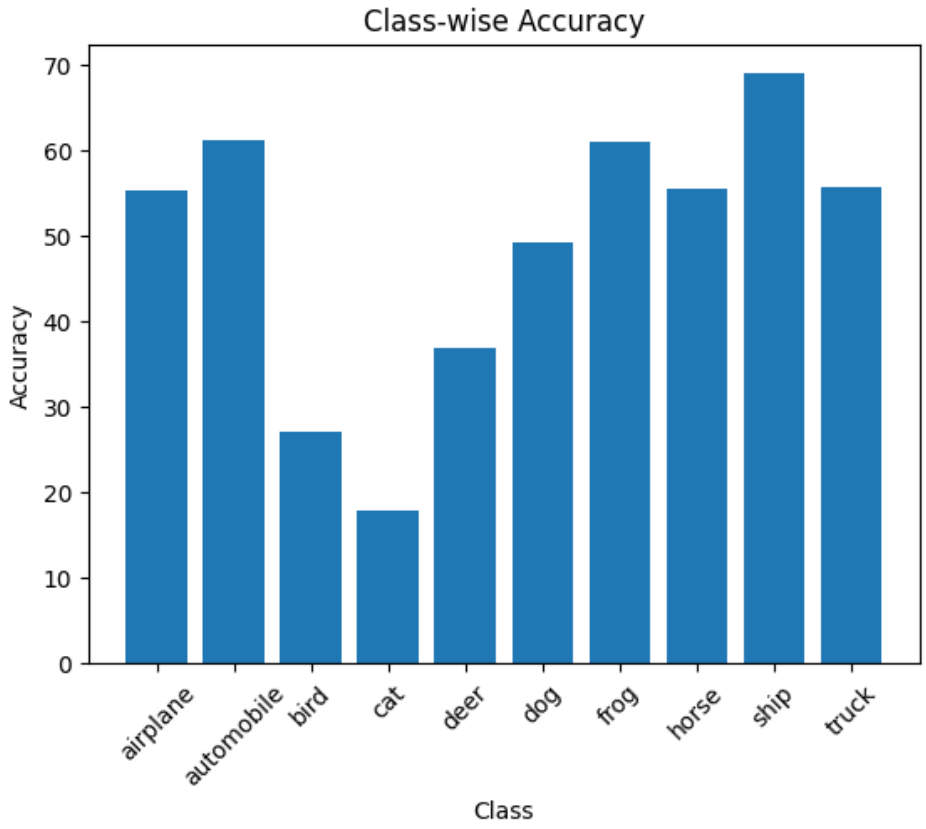
   Following are the loss curves for both training and validation.



The class-wise accuracy is as follows :-

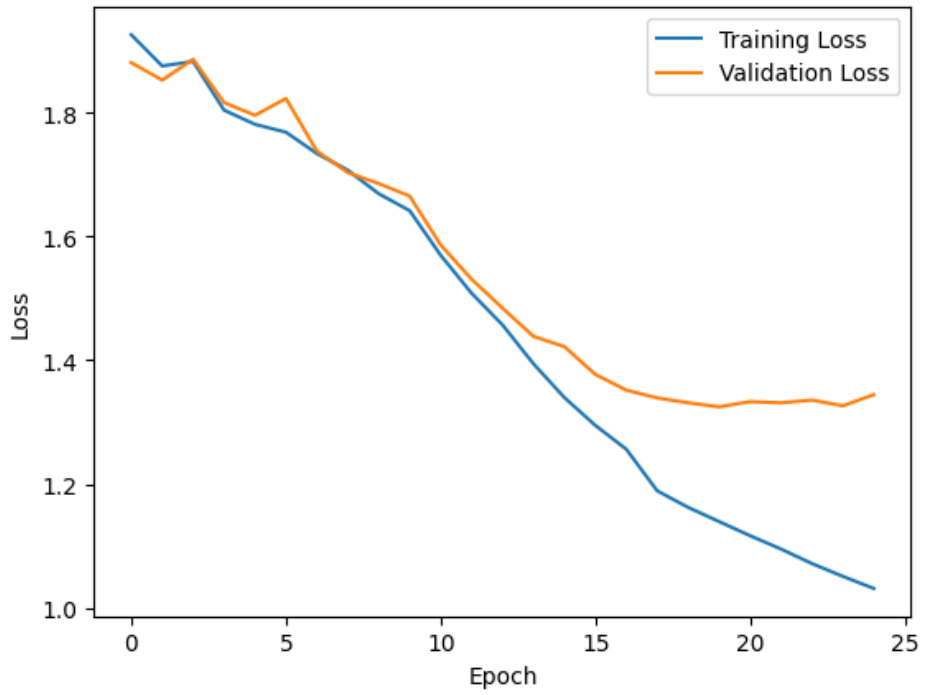
(a) airplane - 55.04
(b) automobile - 61.25
(c) bird - 27.41
(d) cat - 17.52
(e) deer - 36.57
(f) dog - 49.32
(g) frog - 60.41
(h) horse - 55.78
(i) ship - 68.57
(j) truck - 55.02

For better visualization, I have also plotted a bar plot for the class-wise accuracies



2. MultiStepLR - (milestones = $[3, 6, 10, 17]$, initial learning rate - 0.01, gamma - 0.5)

   Following are the loss curves for both training and validation.



   The class-wise accuracy is as follows :-

(a) airplane - 60.25

(b) automobile - 65.48

(c) bird - 38.89

(d) cat - 37.42

(e) deer - 38.52

(f) dog - 41.69

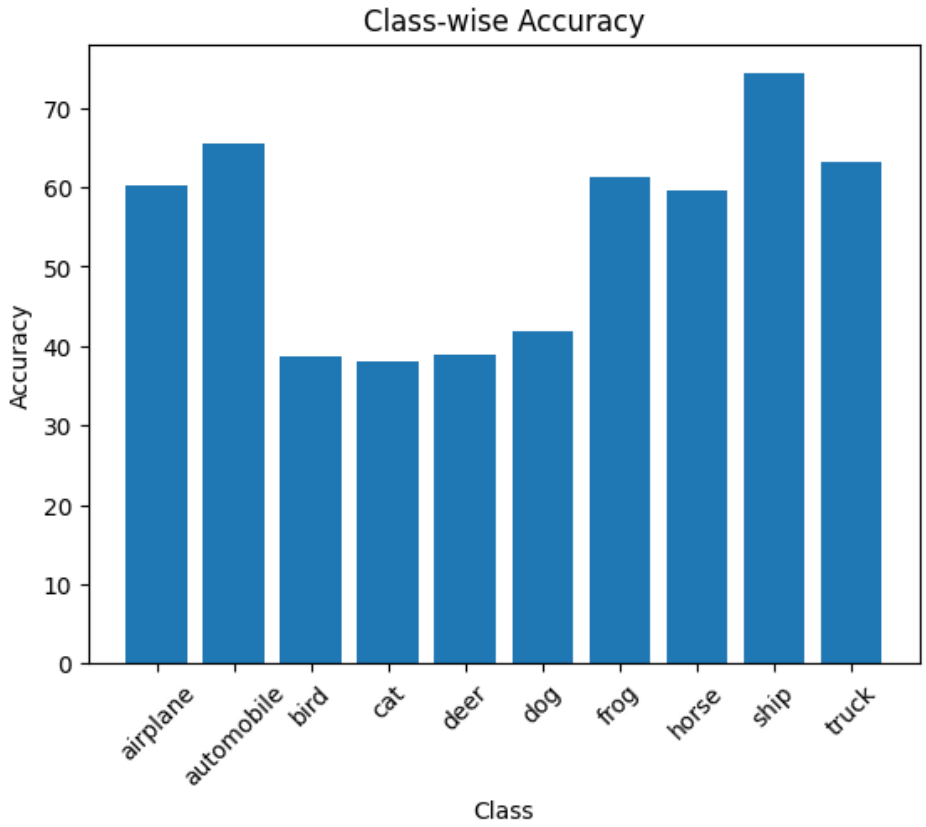
(g) frog - 61.56

(h) horse - 59.023

(i) ship - 74.28

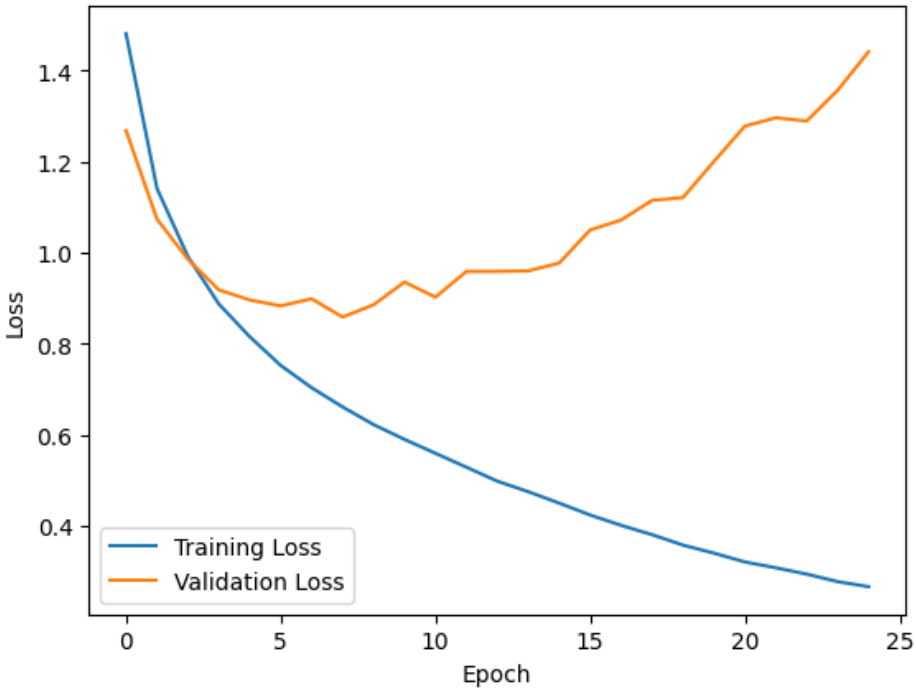
(j) truck - 63.125

For better visualization, I have also plotted a bar plot for the class-wise accuracies



3. Constant Learning Rate - 0.001

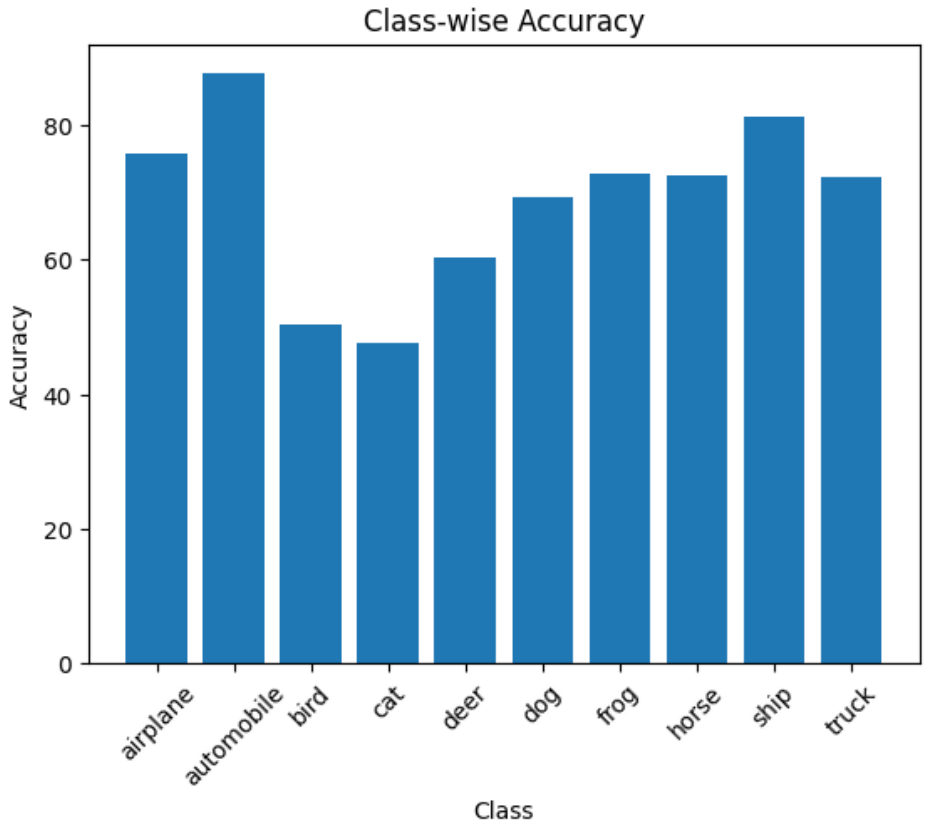
Following are the loss curves for both training and validation.



The class-wise accuracy is as follows :-

(a) airplane - 75.82
(b) automobile - 87.56
(c) bird - 50.23
(d) cat - 47.023
(e) deer - 60.235
(f) dog - 69.12
(g) frog - 72.98
(h) horse - 72.06
(i) ship - 81.50
(j) truck - 72.34

For better visualization, I have also plotted a bar plot for the class-wise accuracies

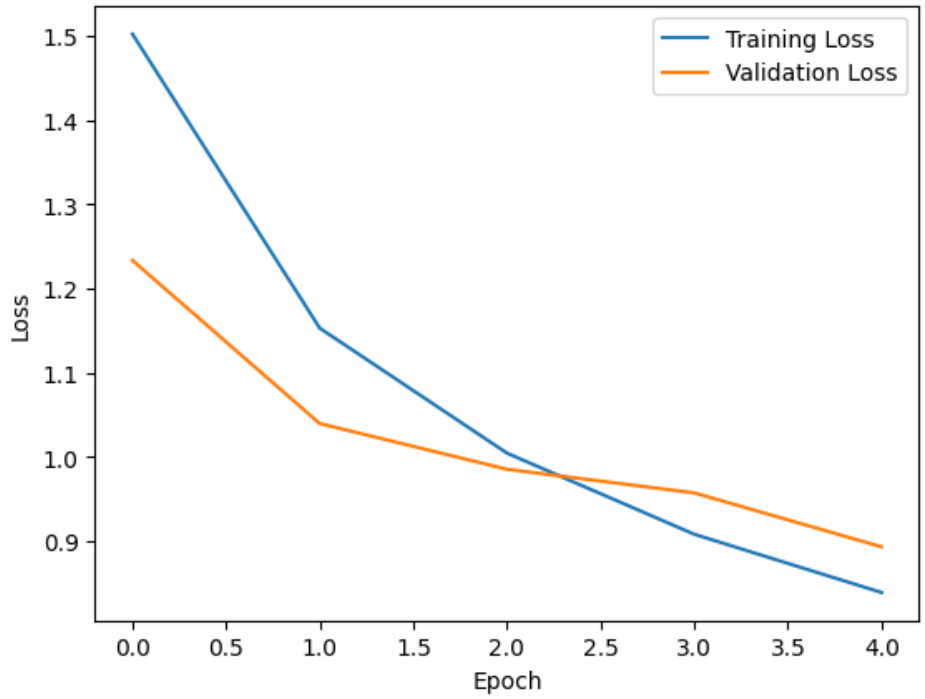


We can see that among the schedulers the multi step LR seems to be working better. The losses have a lower value than the StepLR and the validation loss stabilises quicker than the StepLR.
If we compare MultiStepLR with the constant learning rate, we can observe that the constant learning rate still works better. It converges very quickly around 10 epochs and after that the validation loss starts increasing. The value of losses also come out to be pretty low than the MultiStepLR. Convergence is slower in MultiStepLR due to gradual decrease in the learning rate which results in smoother curves but the losses don't decrease as quickly as the constant Learning rate model.

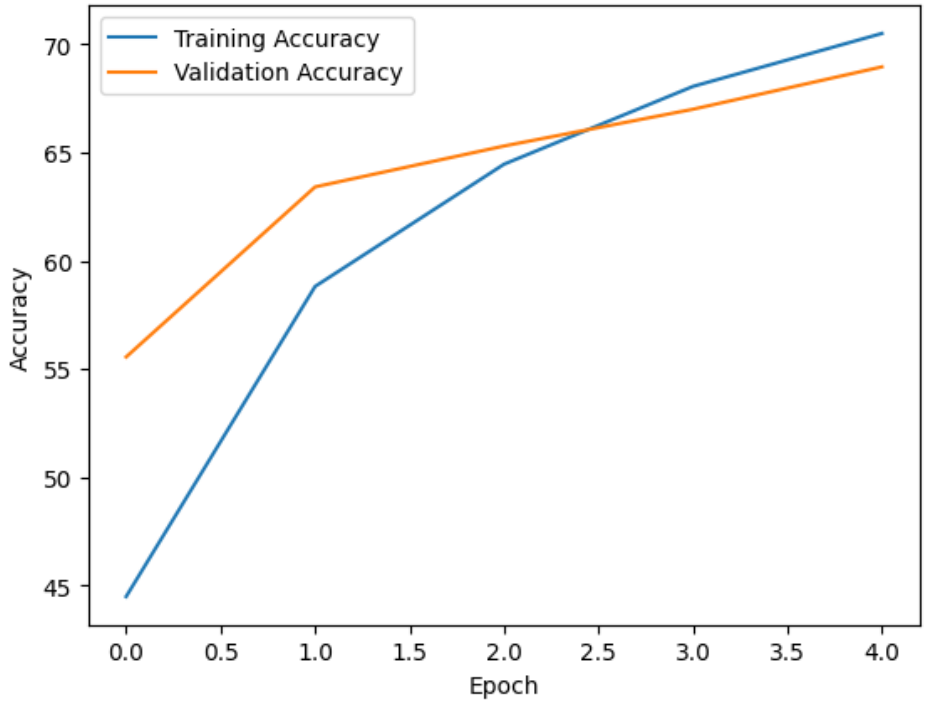
0.0.3 Number of training epochs

I used Adam's optimizer with constant learning rate as 0.001, batch size as 32.

1. 5 epochs -
   Following are the loss curves for both training and validation.

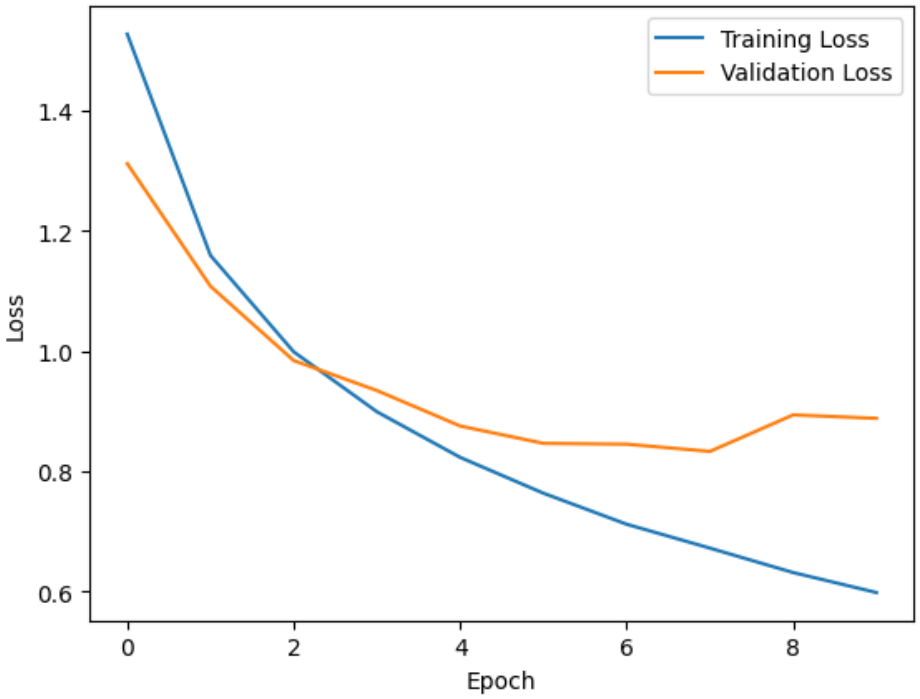


Following are the accuracy curves for both training and validation.(vs epochs)

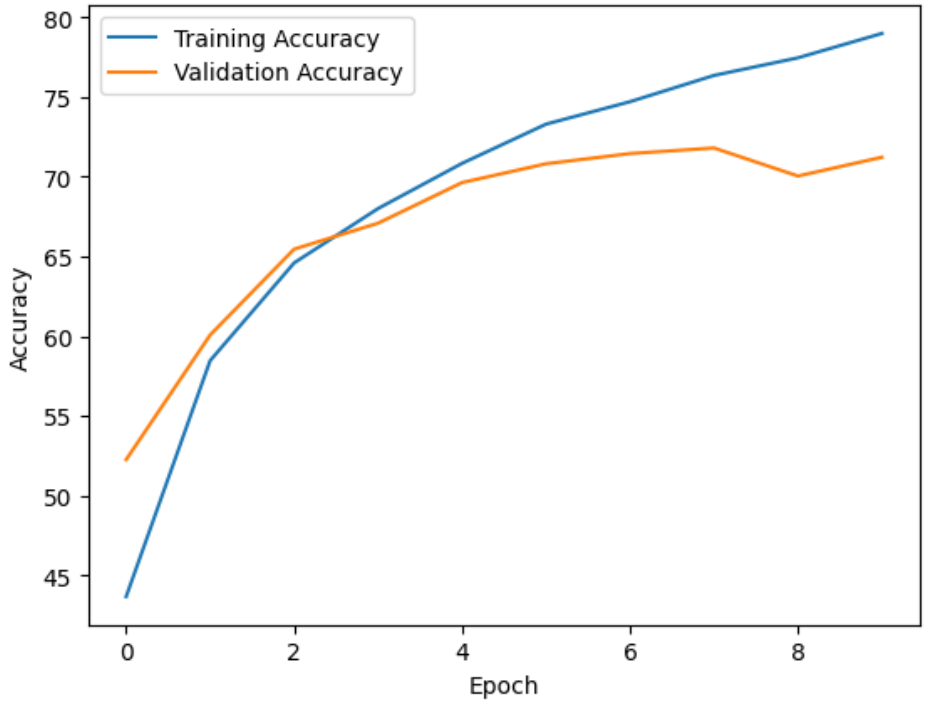


Accuracy of the final network on the train images with adam optimizer: 73%
Accuracy of the final network on the test images with adam optimizer: 68%

2. 10 epochs -
   Following are the loss curves for both training and validation.

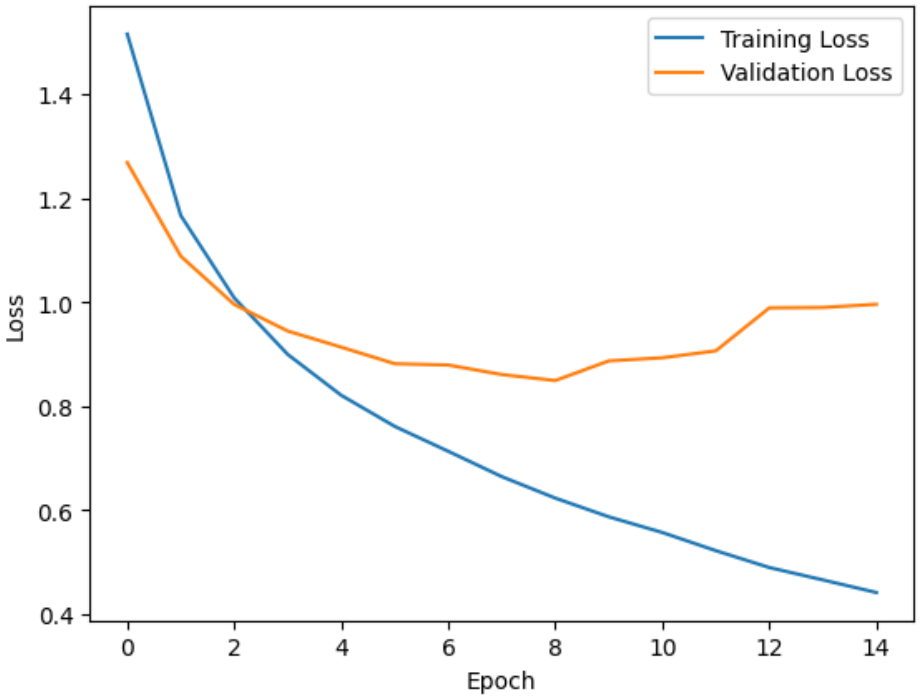


Following are the accuracy curves for both training and validation.(vs epochs)

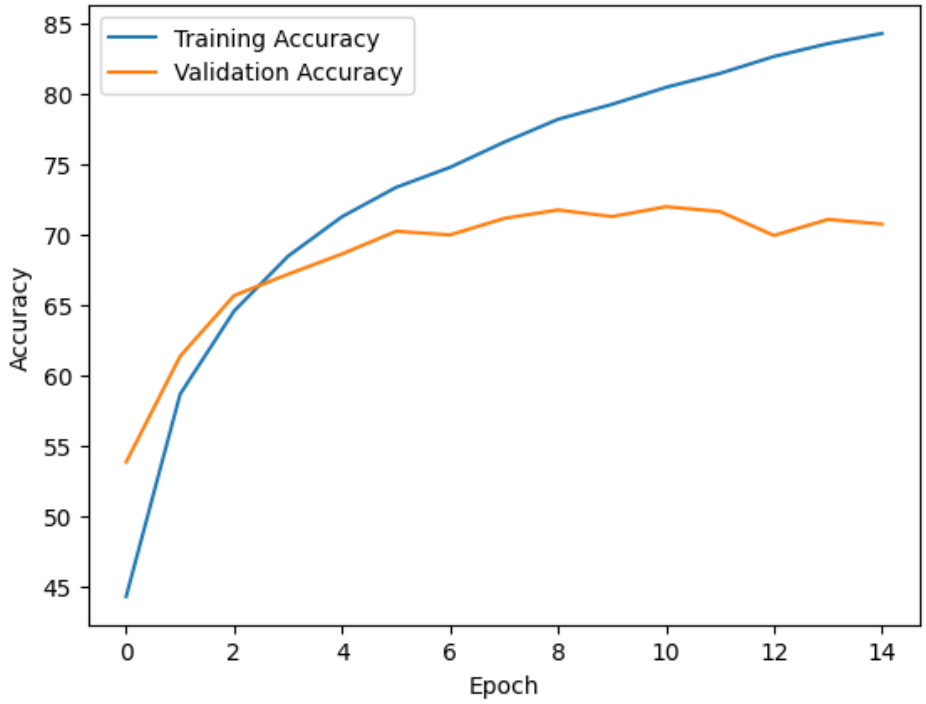


Accuracy of the network on the train images with Adam optimizer: 80 %
Accuracy of the network on the test images with Adam optimizer: 71 %

3. 15 epochs -
   Following are the loss curves for both training and validation.

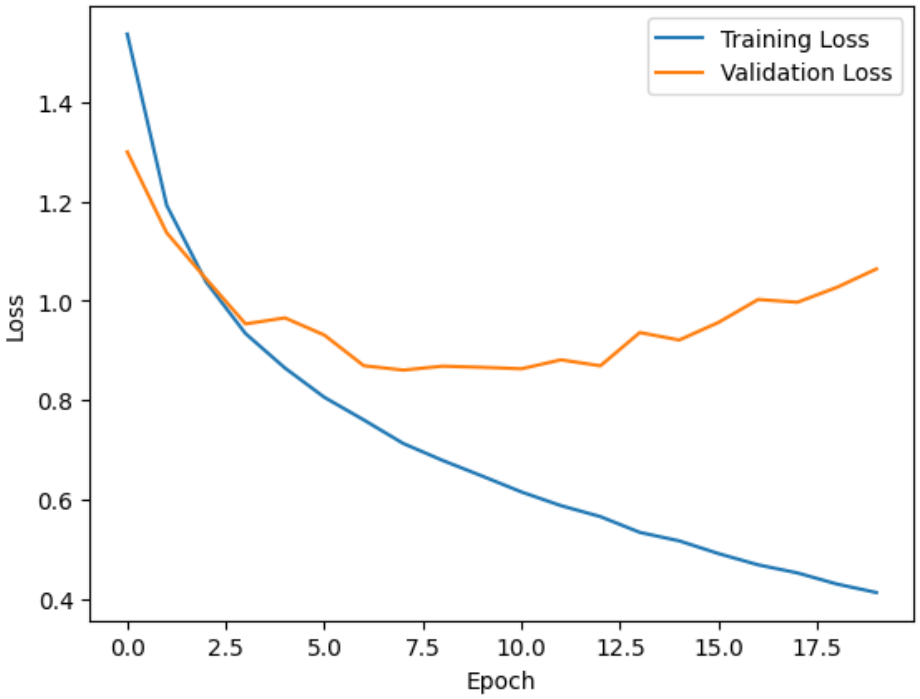


Following are the accuracy curves for both training and validation.(vs epochs)

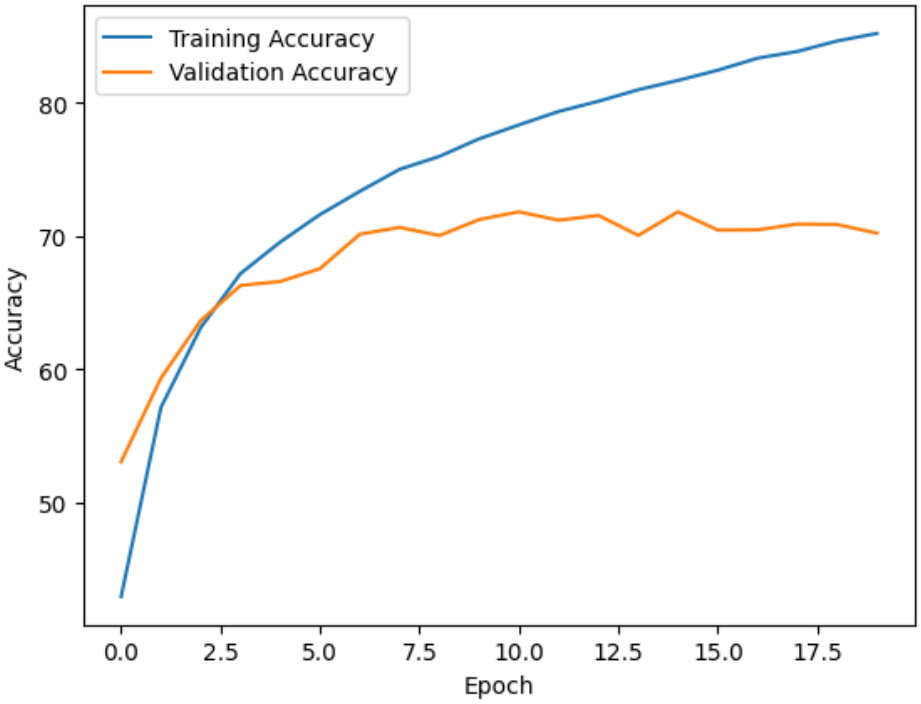


Accuracy of the network on the train images with Adam optimizer: 87 %
Accuracy of the network on the test images with Adam optimizer: 70 %

4. 20 epochs -
   Following are the loss curves for both training and validation.

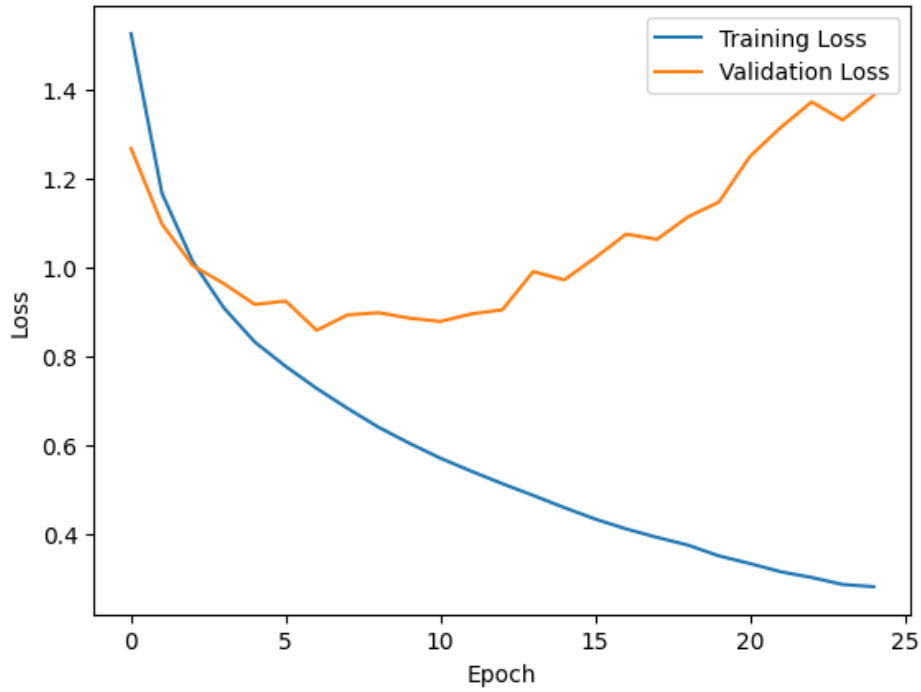


Following are the accuracy curves for both training and validation.(vs epochs)

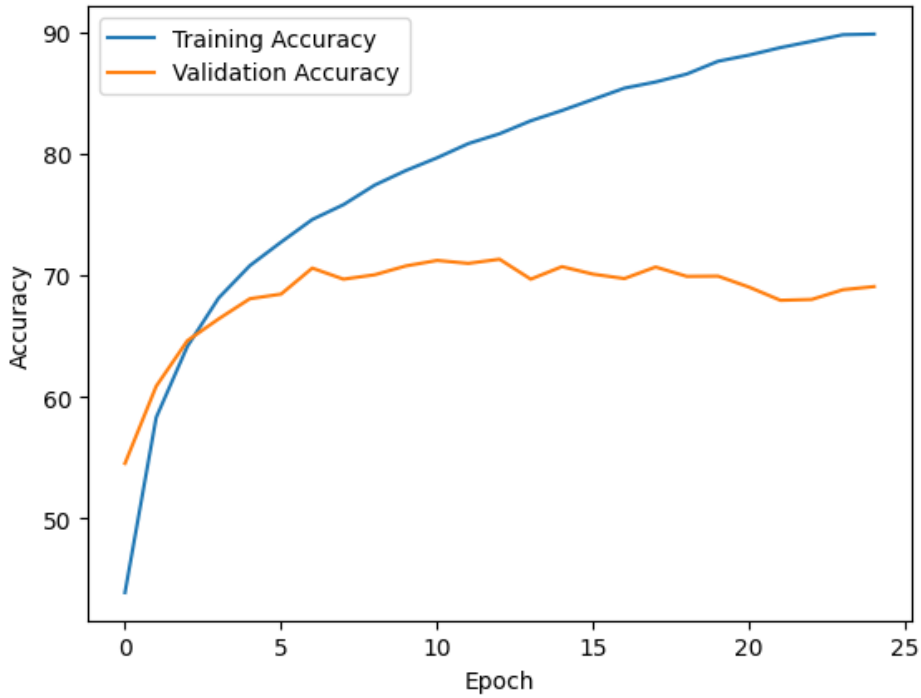


Accuracy of the network on the train images with Adam optimizer: 86 %
Accuracy of the network on the test images with Adam optimizer: 70 %

5. 25 epochs -
   Following are the loss curves for both training and validation.

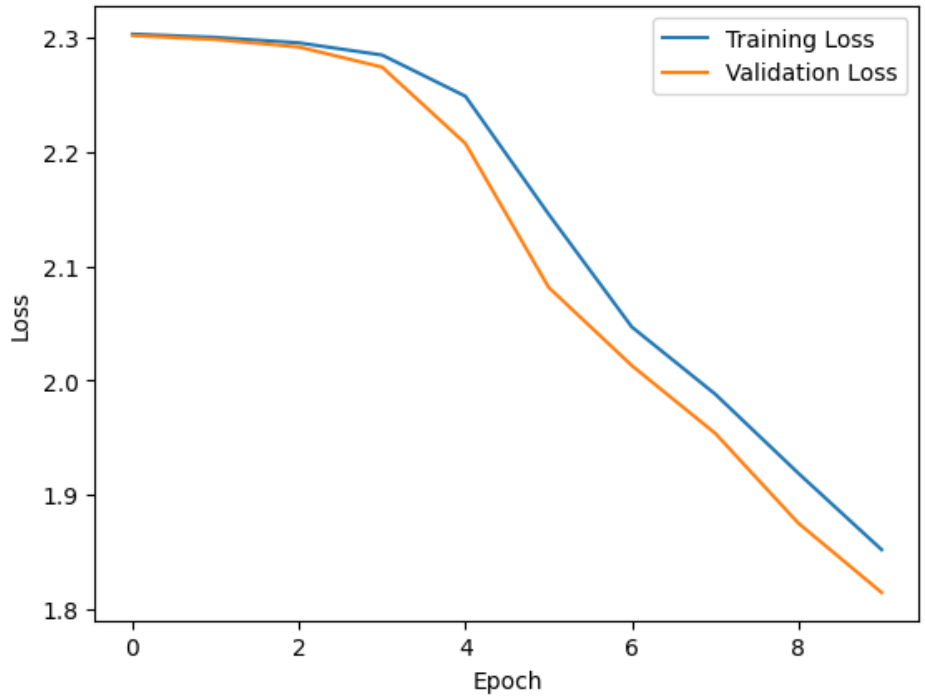


Following are the accuracy curves for both training and validation.(vs epochs)

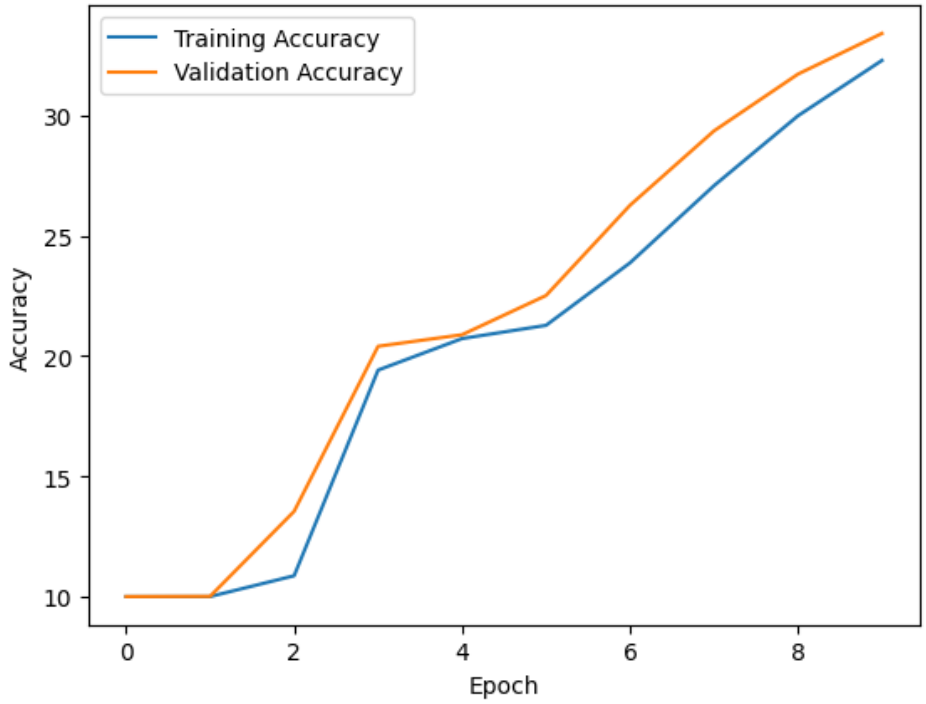


Accuracy of the network on the train images with Adam optimizer: 91 %
Accuracy of the network on the test images with Adam optimizer: 69 %

The best accuracy for adam's optimiser is obtained at 10 epochs. Let's now use the SGD optimizer. The following graphs are obtained :-



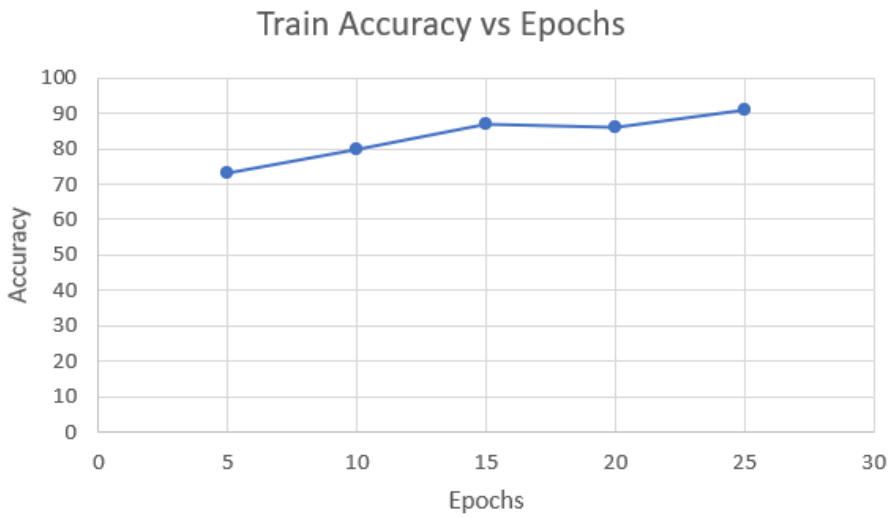
Following are the accuracy curves for both training and validation.(vs epochs)

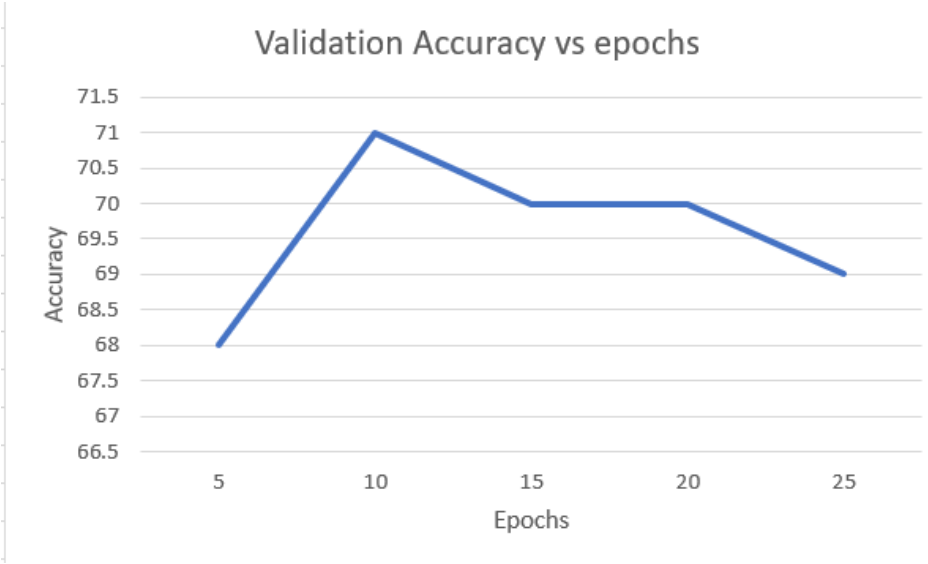


Accuracy of the network on the train images with Adam optimizer: 33%
Accuracy of the network on the test images with Adam optimizer: 33%

So we observe again that the adam's optimizer works better for this.
If we analyse the epoch variation and the final validation and training accuracies achieved, we can clearly see that increasing the epochs does not mean an overall

improvement in the generalization of the model. The training accuracy will increase as we increase the number of epochs leading to overfitting of the model and thus resulting in a decrease of the validation accuracy of the model. Hence very low number of epochs may lead to underfitting, very high number of epochs lead to overfitting. Follwowing are the trends of final training and validation accuracies with respect to the number of epochs



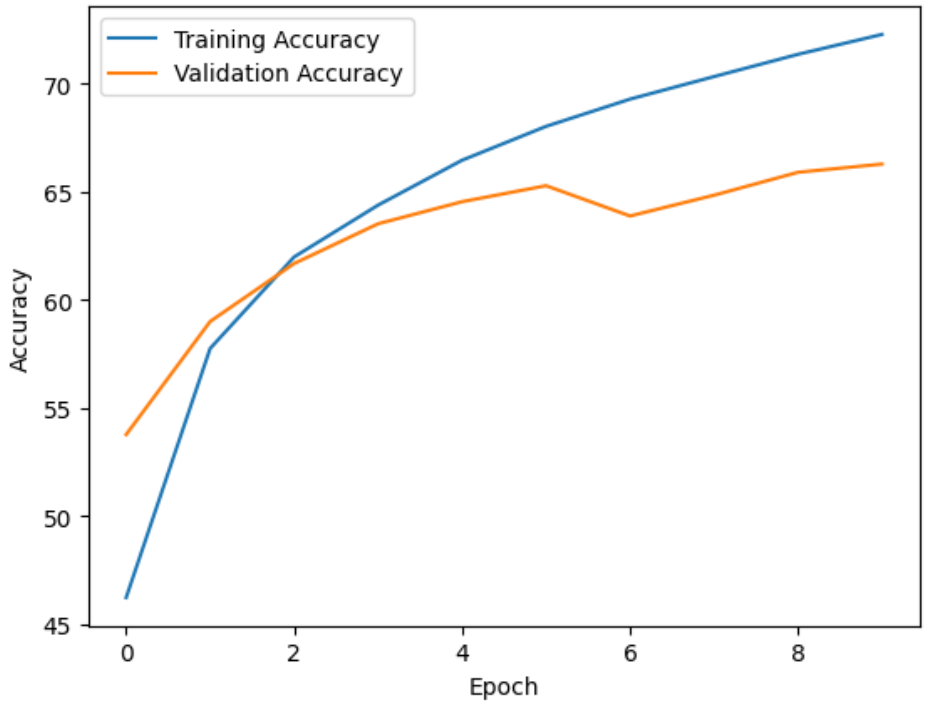


0.0.4 Batch Size

For the experiments, I have used Adam's optimizer, learning rate - 0.001 and number of epochs as 10.

1. batch size - 4

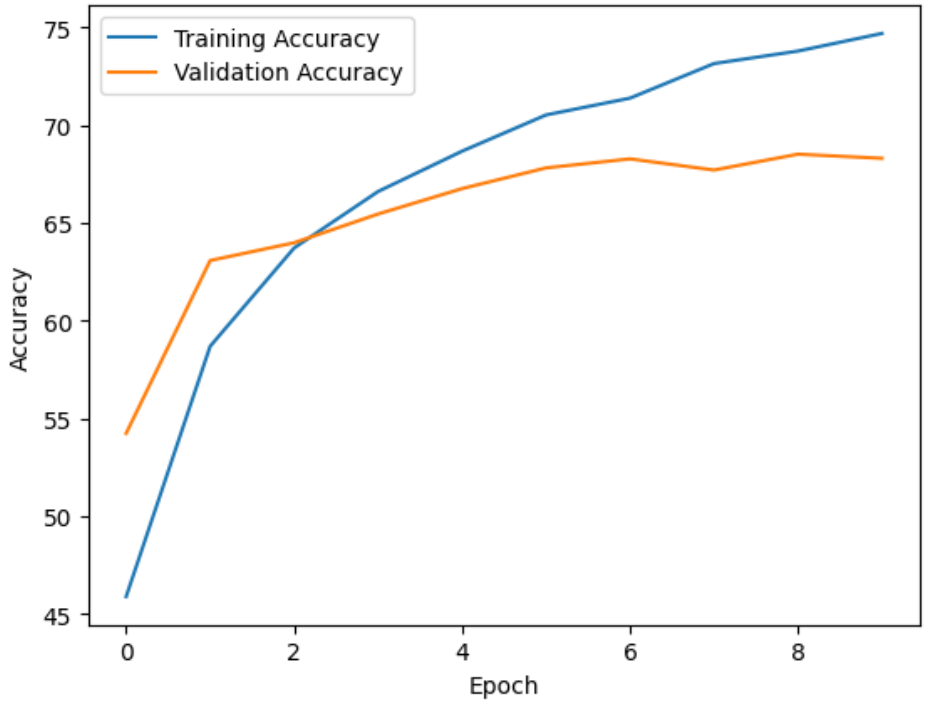
   Following are the accuracy curves for both training and validation.(vs epochs)



Accuracy of the final network on the train images with Adam optimizer: 75%
Accuracy of the final network on the test images with Adam optimizer: 66%

2. batch size - 8

Following are the accuracy curves for both training and validation.(vs epochs)

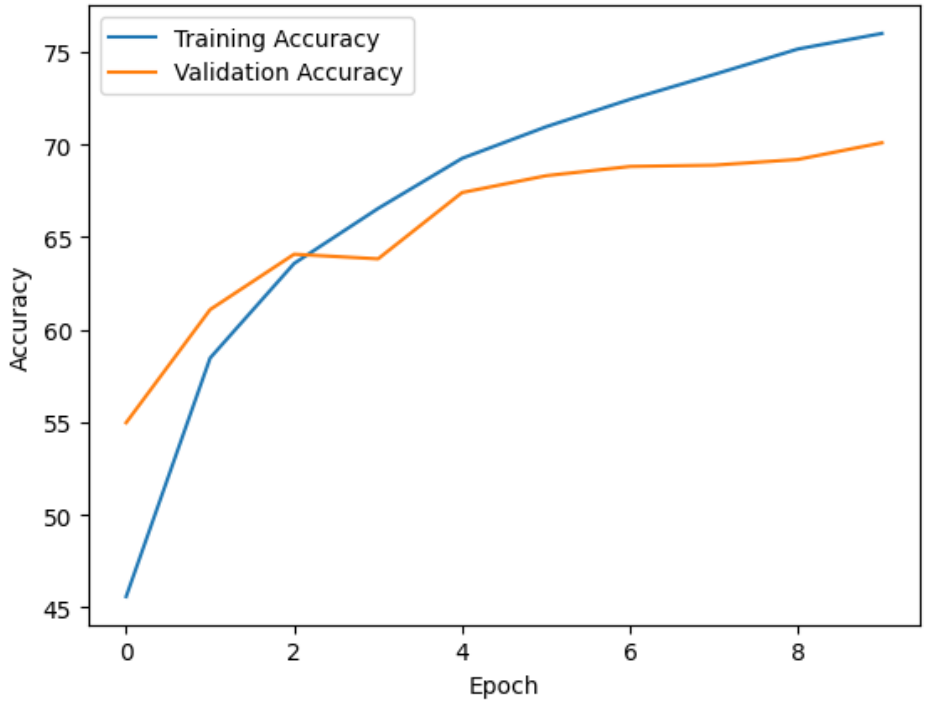


Accuracy of the final network on the train images with Adam optimizer: 76%

Accuracy of the final network on the test images with Adam optimizer: 68%

3. batch size - 16

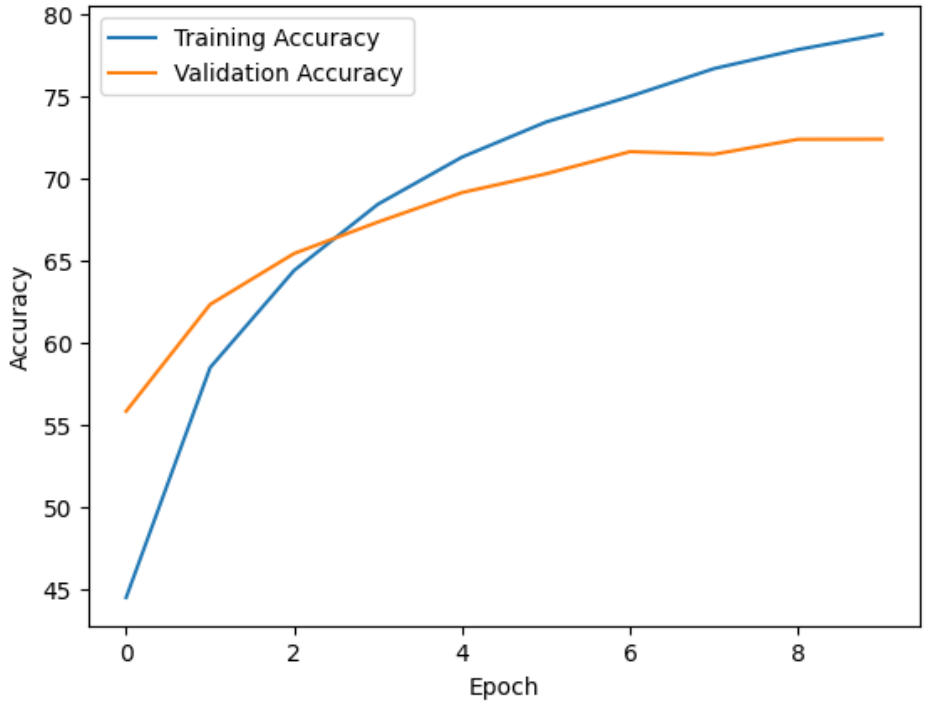
   Following are the accuracy curves for both training and validation.(vs epochs)



   Accuracy of the final network on the train images with Adam optimizer: 79%
   Accuracy of the final network on the test images with Adam optimizer: 70%

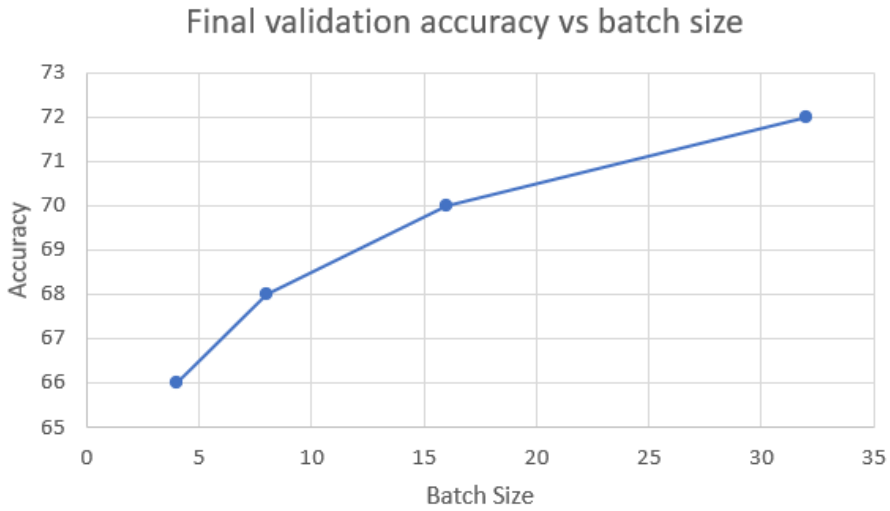
4. batch size - 32

   Following are the accuracy curves for both training and validation.(vs epochs)

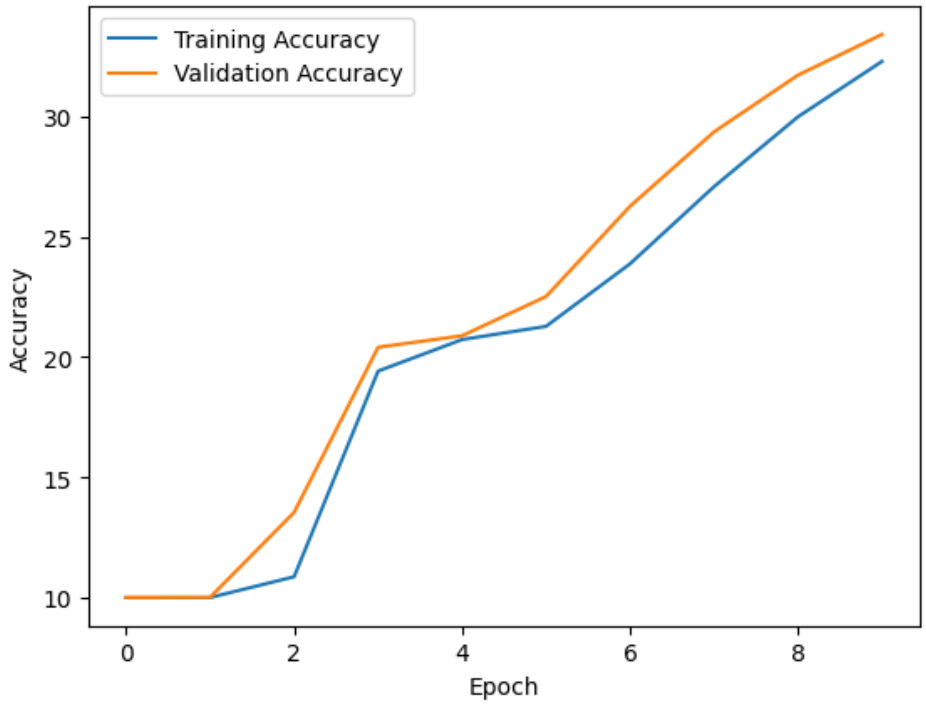


Accuracy of the final network on the train images with Adam optimizer: 82%
Accuracy of the final network on the test images with Adam optimizer: 72%

To understand the trend with batch size, I have plotted the final validation accuracies vs the batch size. Graph is as follows :-



We can observe that the validation accuracy increases with batch size for the given batch sizes, although this trend might not continue if we further increase the batch sizes. Increasing batch sizes help in reduction in the training time somewhat but may not help with the validation accuracy improvement.
If I now use our best batch size i.e 32 and use SGD to train our model, the following graph is obtained



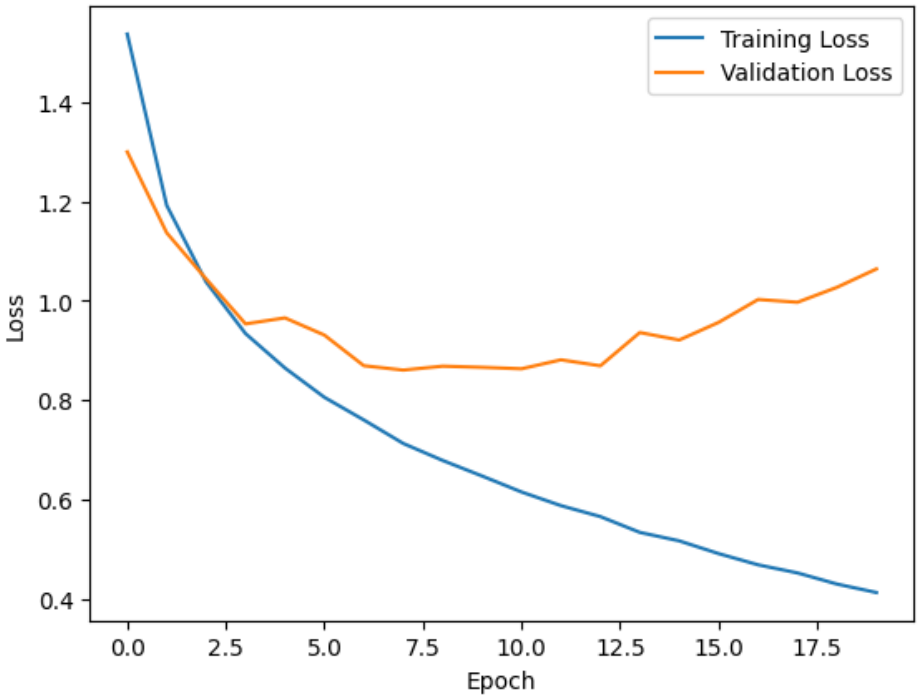
Accuracy of the final network on the train images with SGD optimizer: 33%
Accuracy of the final network on the test images with SGD optimizer: 33%

We can clearly see that the SGD optimizer does not fair well against the Adam's optimizer. The loss values are also pretty high for SGD. SGD might need more epochs in order to converge to it's optimal value.

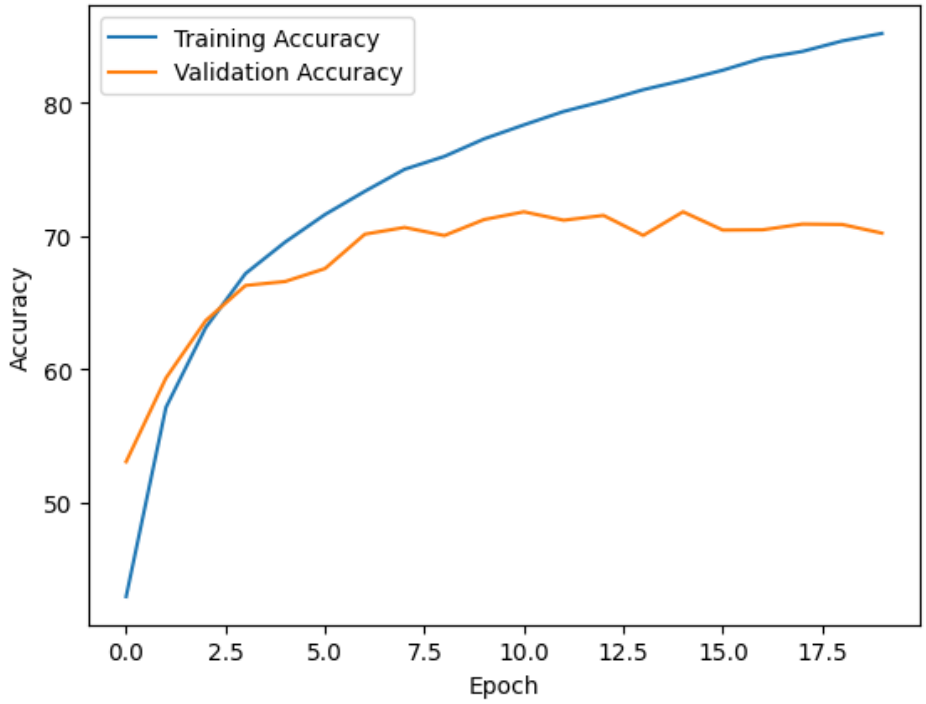
Effect of Loss Function

I have used 20 epochs with batch size of 32 for training. The learning rate is 0.001.

Following are the loss curves for both training and validation using Categorical cross-entropy loss.

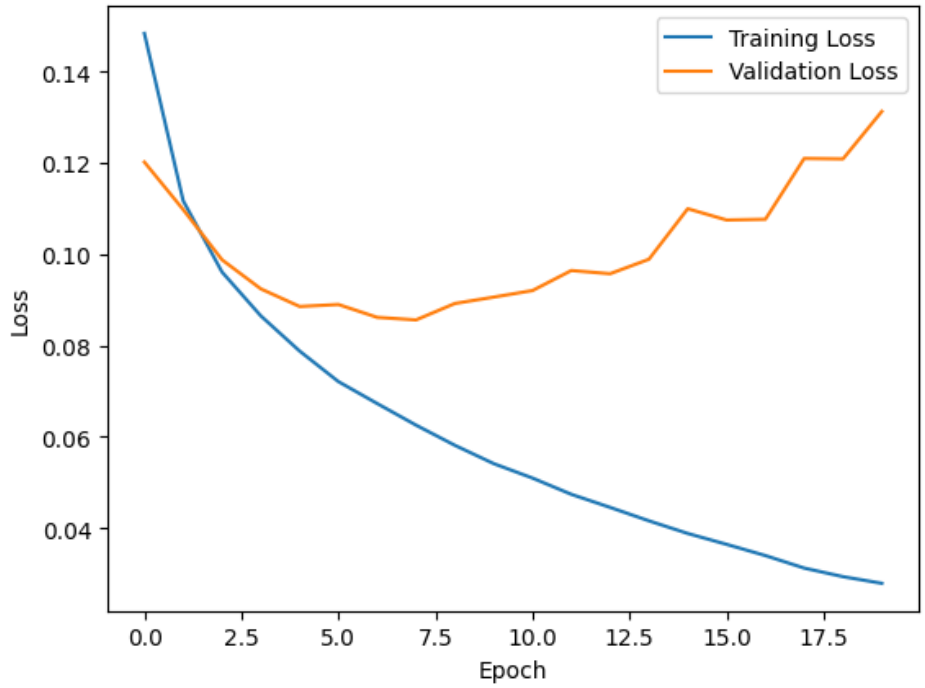


Following are the accuracy curves for both training and validation.(vs epochs)

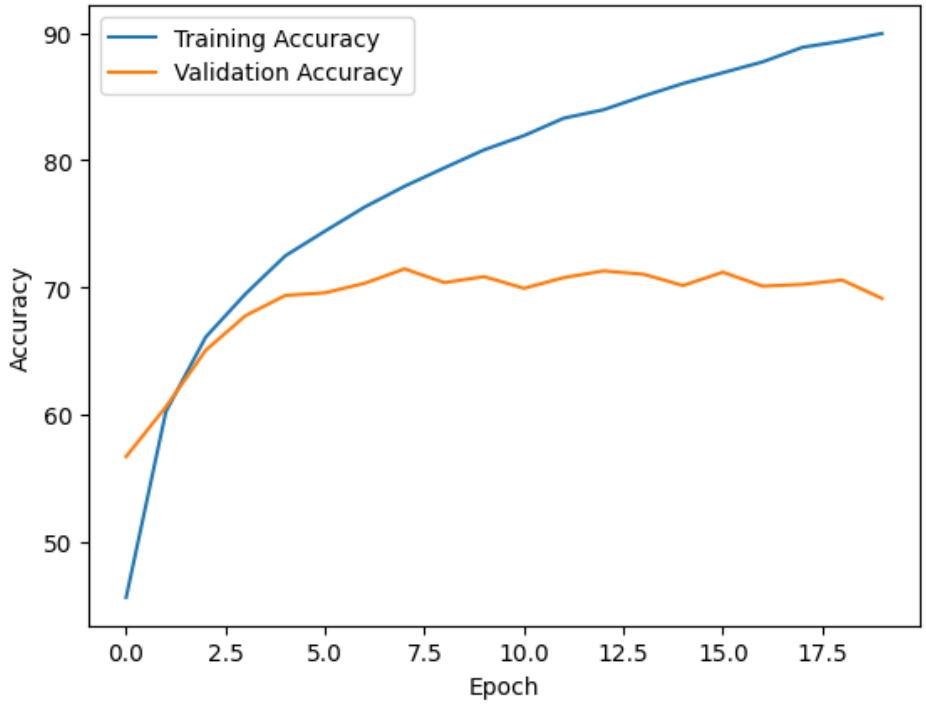


Accuracy of the network on the train images with Adam optimizer: 86 %
Accuracy of the network on the test images with Adam optimizer: 70 %

Let us change the loss function to KL divergence loss.
Following are the loss curves for both training and validation.



Following are the accuracy curves for both training and validation.(vs epochs)



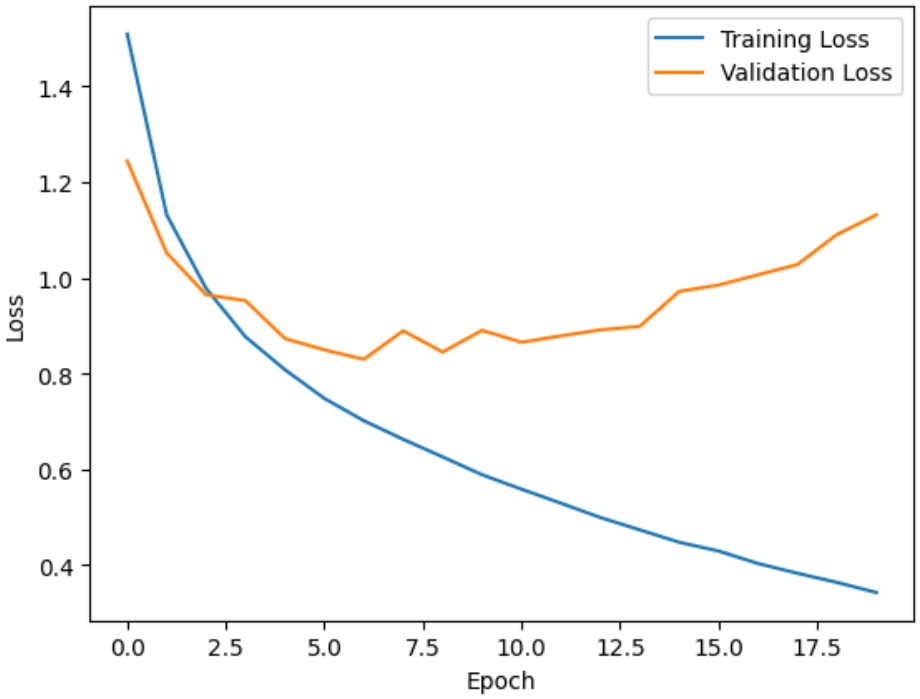
Accuracy of the network on the train images with Adam optimizer: 90 %
Accuracy of the network on the test images with Adam optimizer: 69 %
If we analyse the final accuracies, they come out to be somewhat similar. One important difference to not is that in KL Divergence loss the validation loss saturates pretty quickly(around 6-7 epochs) and then it starts increasing whereas for the cross-entropy loss it saturates around 10 epochs. KL divergence loss also has a better training accuracy. Though, we can argue that the KL divergence loss

model might overfit easily and hence is not robust as compared to the categorical cross-entropy loss.

Effect of Data Augmentation

With Augmentation

I tried different augmentations like normalisation, random cropping and random horizontal flips. The one which gave better results was the combination of normalisation with random horizontal flips. Following are the loss curves and the accuracies

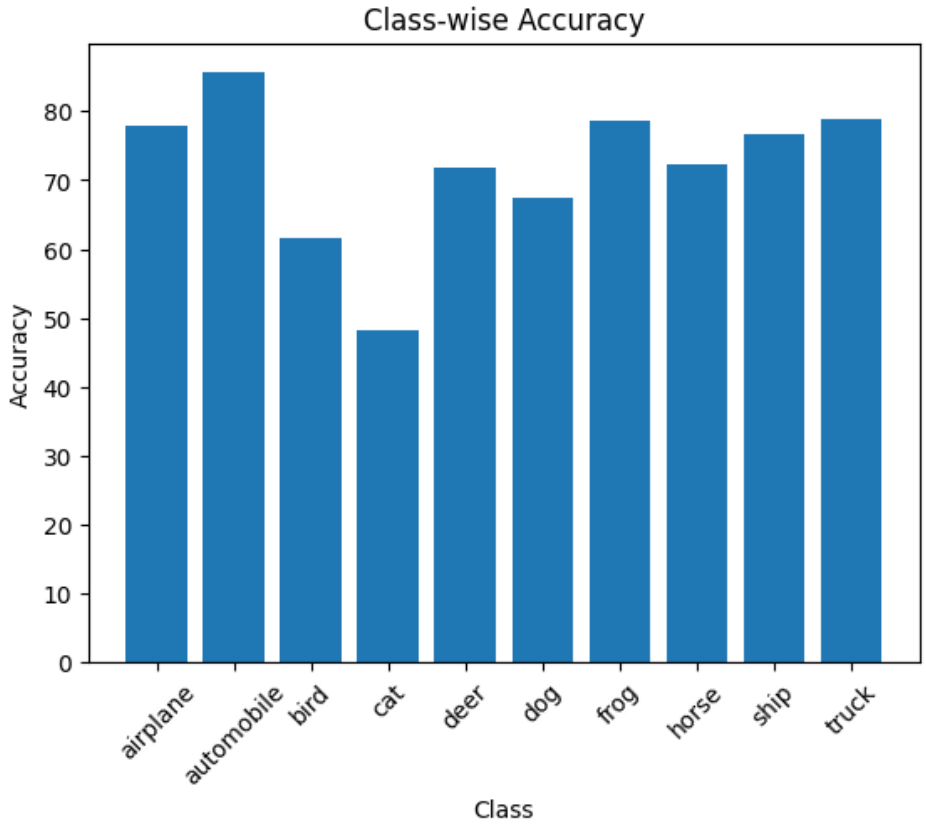


The class-wise accuracy is as follows :-

1. airplane - 77.9
2. automobile - 85.6
3. bird - 61.6
4. cat - 48.3
5. deer - 71.7
6. dog - 67.5
7. frog - 78.6
8. horse - 72.3

9. ship - 76.8
10. truck - 78.8

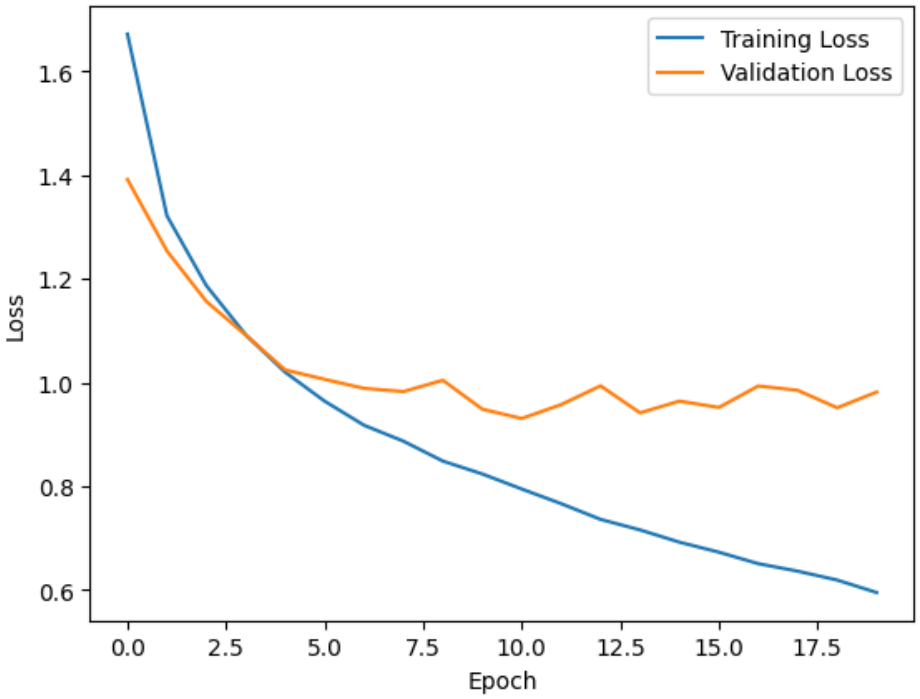
For better visualization, I have also plotted a bar plot for the class-wise accuracies



Accuracy of the network on the train images with Adam optimizer: 89 %
Accuracy of the network on the test images with Adam optimizer: 73 %

Without Augmentation

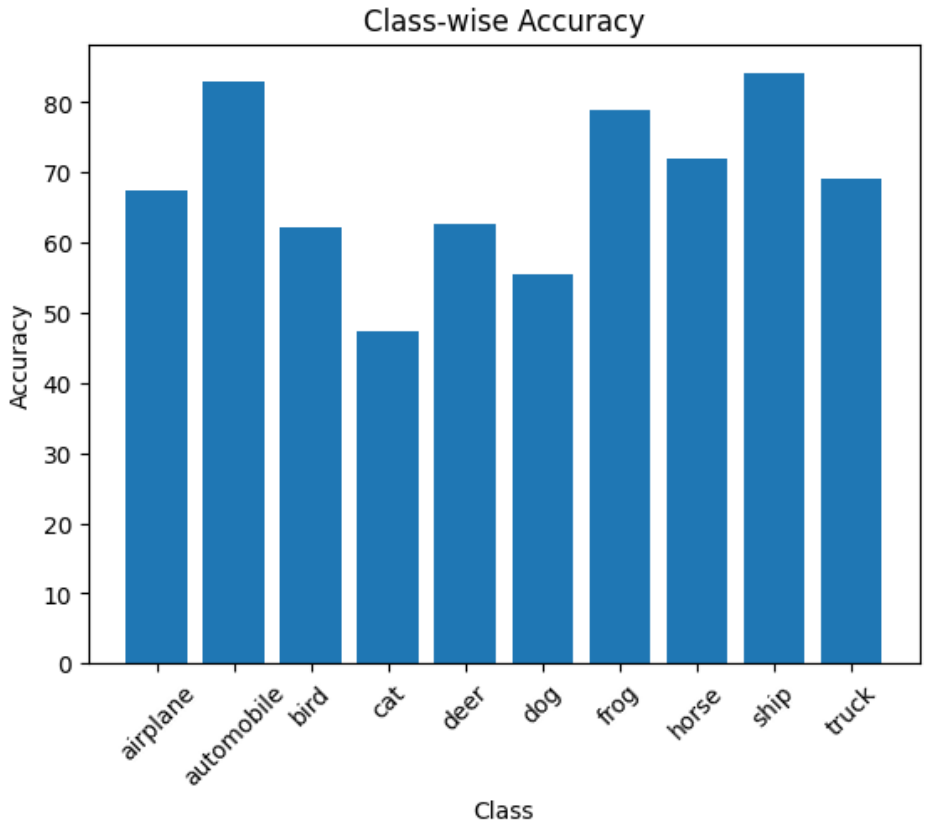
I trained my model with 20 epochs, batch size 32, learning rate 0.001 without any augmentation in the image data. I obtained the following loss curves and the accuracies.



The class-wise accuracy is as follows :-

1. airplane - 67.5
2. automobile - 82.9
3. bird - 62.3
4. cat - 47.5
5. deer - 62.7
6. dog - 55.6
7. frog - 79.0
8. horse - 72.1
9. ship - 84.1
10. truck - 69.1

For better visualization, I have also plotted a bar plot for the class-wise accuracies



Accuracy of the network on the train images with Adam optimizer: 80 %
Accuracy of the network on the test images with Adam optimizer: 68 %

The effect of data augmentation is quite evident, both the training and validation accuracies decrease by a considerable amount if no augmentation is used. Further data augmentation helped our model to converge quicker than the one with no augmentation. Also, if look at the bar graphs of the class-wise accuracy, we see a decrease in almost all classes in the case of no augmentation except ship and bird where the accuracy increases a little bit. All in all, data augmentation helped our model to generalize in a better way thus performing significantly better on the test data and the training data.

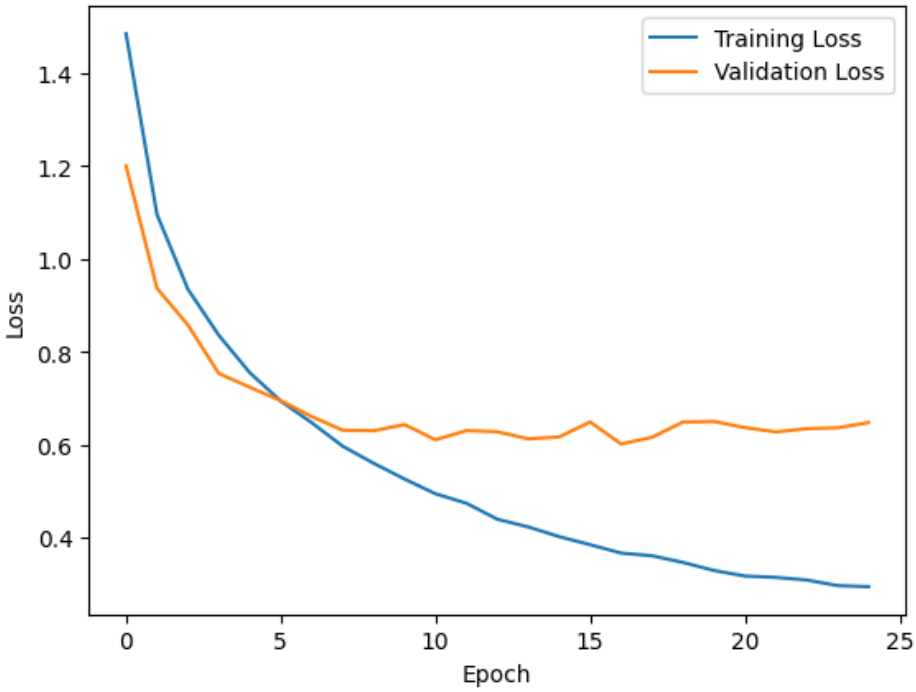
Improving the Neural Network

I am again going to use pytorch for this purpose. Let us first modify the architecture of the neural network. I will make use of an additional layer for droputs. Further I will make use of 4 convolutional layers. They are described as follows. For showing the type of convolution layer I will follow the following convention (in channels, out channels, kernel size, padding)
Conv1 = (3,32,3,1)
Conv2 = (32,32,3,1)
Conv3 = (32,64,3,1)

Conv4 = (64,64,3,1)
2 max-pooling layers
convention (kernel size, stride)
Pool1 = (2,2)
Pool2 = (2,2)
3 dropout layers
d1 = (p = 0.25)
d2 = (p = 0.25)
d3 = (p = 0.5)
2 fully-connected layers
fc1 = (64 * 8 * 8, 512)
fc2 = (512, 10)
After each convolution layer and after fc1, I have used relu activation function. After each pooling layer and the fc1 layer, I have added a dropout layer with a probability of 0.25 for dropping some inputs. The dropout layer after fc1 has a probability of 0.5 so as to aid in regularisation
Further, the learning is not a constant. It is initially set as 0.0005 and then I make use of MultiStepLR scheduler to reduce this by 0.2 at the 15th epoch. Totally, I am using 25 epochs for training the model. The loss function used is Categorical cross-entropy loss.

I have made the following design decisions because of the following reasons :- 1. The dropout layer can prevent overfitting which occurs in the original architecture by randomly dropping out a fraction of the input units during training. This helps to reduce the dependence of the model on specific input features and encourages the network to learn more robust features that generalize better to new data.
2. I have used a deeper neural network so as to better identify the features embedded in the images. Each convolutional layer extracts increasingly complex features from the input image, allowing the network to learn more abstract representations of the input image as it progresses through the layers.
3. Learning Rate is kept low as I observed that keeping high learning rates was causing the optimum to be skipped. Further, I have used a scheduler to reach the optima (probably local)
4. Data Augmentation is quite simple. It is just the normalization of the images.

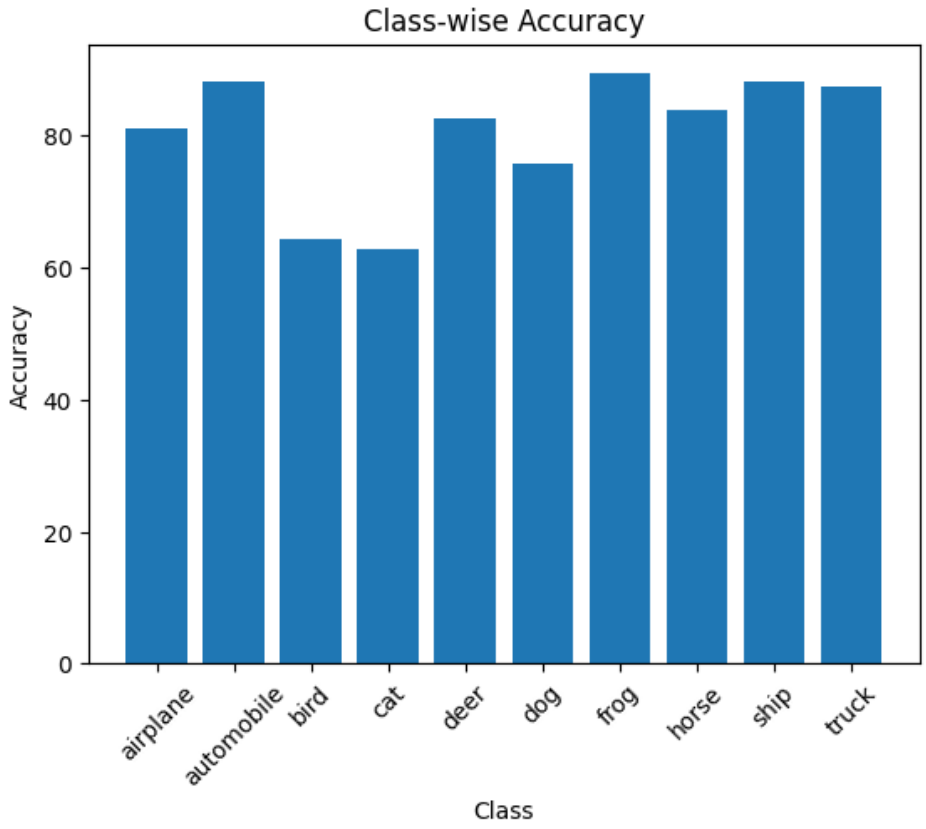
Now I'll show you the graphs for the loss and the class-wise accuracy.



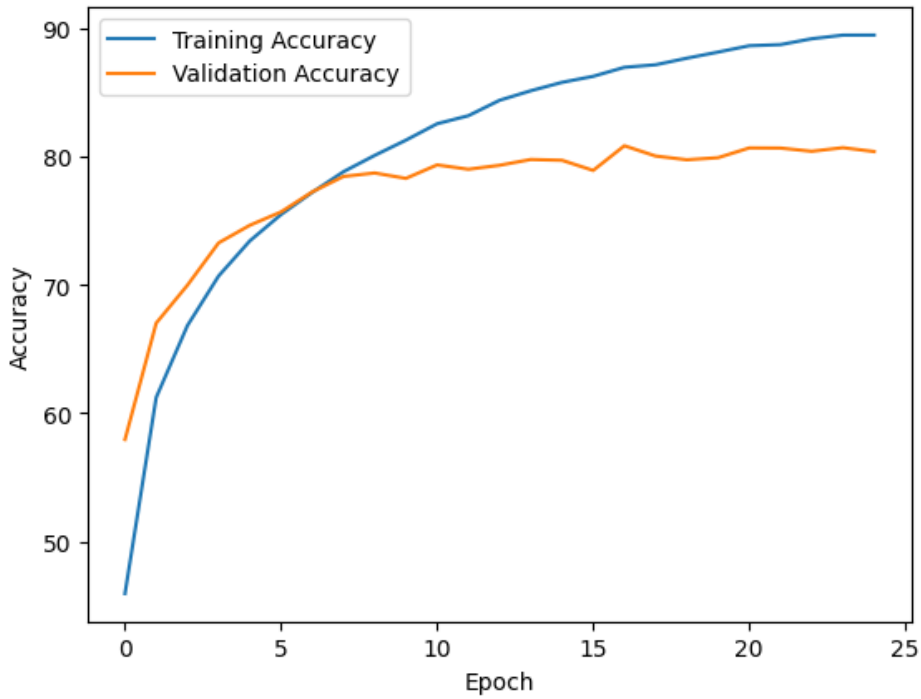
The class-wise accuracy is as follows :-

1. airplane -81.1
2. automobile - 88.2
3. bird - 64.4
4. cat - 62.8
5. deer - 82.6
6. dog - 75.7
7. frog - 89.4
8. horse - 84.0
9. ship - 88.2
10. truck - 87.4

For better visualization, I have also plotted a bar plot for the class-wise accuracies



Following is the accuracy plot :-



Accuracy of the final network on the train images with Adam optimizer: 89%
Accuracy of the final network on the test images with Adam optimizer: 80.38%
We can see that the training and validation accuracies come out to be higher compared to other models. I further tried by using data augmentation techniques

but it didn't help much with the accuracy part. Also, I tried different learning rates and changed the number of epochs, nothing was as good as this model, hence this is my improved model.

Resources

There were lot of good resources which helped me in better analysis of this assignment. Most of them were the 'Towards Data Science' articles given in the assignment pdf. For the scratch part, the following resource - `https://www.kaggle.com/code/valentynsichkar/convolutional-neural-network-from-scratch-cifar10/notebook` helped me structure my code and acted as a starting point for me. For, the Adam's optimizer, I made use of another article on the 'Towards Data Science' website. Here is the link.