COL341 - Assignment 4

Valaya - 2019MT10731

April 29, 2023

Part 1: Implement a Neural Network

First, the CIFAR-10 Dataset was loaded in the .ipynb notebook

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

Three separate classes are made for the convolutional layer, max pooling layer, and the fully connected layer. The Relu activation function is incorporated in these layers itself. Following is the description of all three layers.

- Convolutional layer

```
class ConvLayer:
      def __init__(self, input_channels, num_filters, kernel_size):
      def forward(self, X):
          return output
      def backward(self, dL_dZ):
          return dL_dX, dL_dW, dL_db
```

  Objects of the `class Convlayer` have parameters - `input_channels`(input depth), `num_filters`(output depth), and `kernel size`. The forward function evaluates the dimensions of the output volume and convolves the input with the weights followed by addition with the bias terms. The backward function evaluates the partial derivative of Loss function with respect to X, weights, and bias.

- Max-Pooling Layer

```
class MaxPoolLayer:
    def __init__(self, pool_size):
        self.pool_size = pool_size
    def forward(self, X):
        return output
    def backward(self, dL_dZ):
        return dL_dX
```

  Objects of the `class MaxPoolLayer` have parameter - `pool_size`. The forward function evaluates the dimensions of the output volume and reduces the height and width dimension of the input by half after evaluating the maximum element of each 2X2 kernel. The backward function evaluates the partial derivative of Loss function with respect to X which is used as input when the backward function of the preceding layer is called.

- Fully Connected Layer

```
class FCLayer:
    def __init__(self, input_size, output_size, activation=None):
    def forward(self, X):
        return output
    def backward(self, dL_dZ):
        return dL_dX, dL_dW, dL_db
```

Objects of the `class Convlayer` have parameters - `input_size`, `output_size`, and `activation` which is to ensure that Relu is used only after `FC1` and not `FC2`. Similar to the conv layer, The forward function of FC layer evaluates the dimensions of the output volume and convolves the input with the weights after flattening it. The backward function evaluates the partial derivative of Loss function with respect to X, weights, and bias.

While training on this Convolutional Neural Network, it was observed that on moving from one batch to another, it took 8 seconds. There are a total of 1562 batches per epoch, with 20 such epochs. So it would take approximately 8x1562x20 seconds to train the whole network.

Part 2: Implement a PyTorch-based Solution

The CIFAR-10 dataset has only 50000 training images, and the test images are left untouched (10,000). So, we will attempt to augment the training set with additional images derived from it. We will apply some transformations to the training images and use them for training. We are not just increasing the size of the training set by applying these transformations deterministically. We are applying them probabilistically.This proves to be a strong regularisation. The motivation behind these transformations is a sense of invariance in classes. For instance, on flipping an image of a car upside down, it is still an image of a car. Therefore, we will use the following transformations:

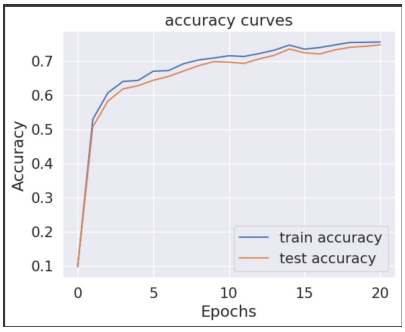
1. **Rotation:** Rotate the image by 45 degrees
2. **Horizontal Flip:** Flip the image horizontally with a probability 0.5.
3. **RandomAffine:** Random affine transformation of the image keeping center invariant.

We apply these transformations to the image probabilistically and then use the transformed images as input to the network for training.

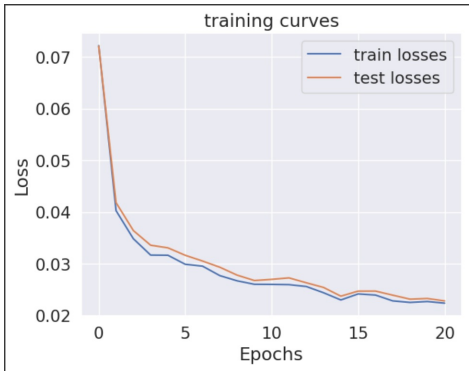
Hyper-parameter Tuning - Adam Optimiser

Learning Rate (LR)

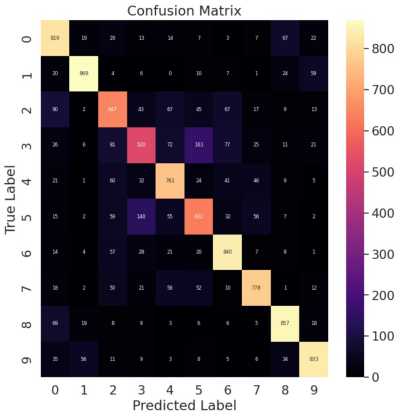
- **LR = 0.001:** Training the model using a batch size of 32 and 20 epochs, along with Adam optimiser, gives an accuracy of **75.76%**. The confusion matrix gives the classwise accuracy. As can be seen from the loss plots, the validation loss curve is converging at similar rate to the training curve indicating minimal overfitting.



(a) Accuracy Plots

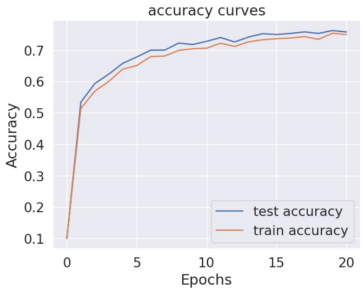


(b) Loss PLots

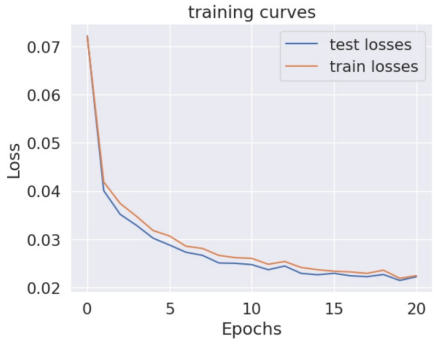


(c) Confusion Matrix

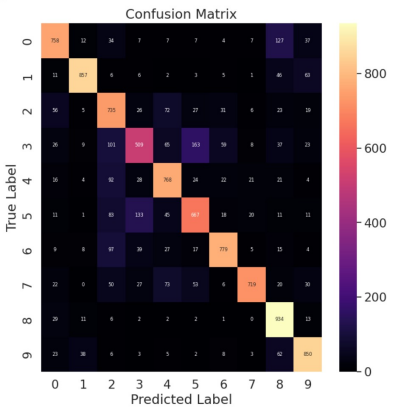
- **LR = 0.0005:** Training the model using a batch size of 32 and 20 epochs, along with Adam optimiser, gives an accuracy of **75.76%**. The confusion matrix gives the classwise accuracy. As can be seen from the loss plots, the validation loss curve is converging at similar rate to the training curve indicating minimal overfitting.



(a) **Accuracy Plots**

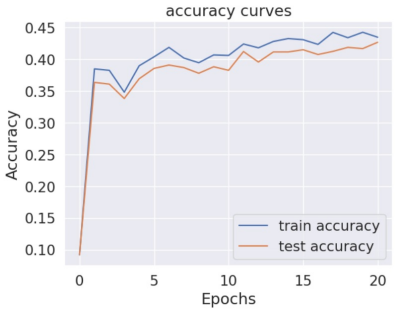


(b) **Loss PLots**

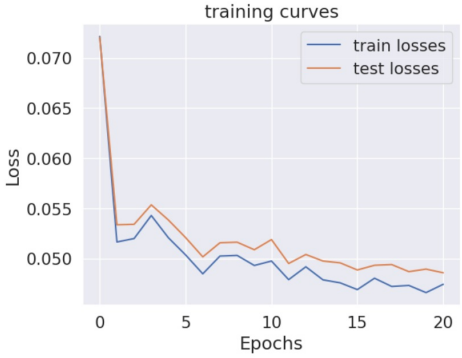


(c) **Confusion Matrix**

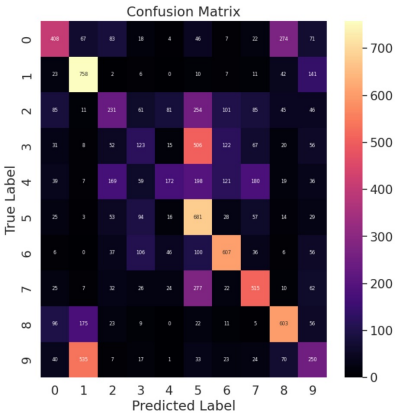
- **LR = 0.006:** Training the model using a batch size of 32 and 20 epochs, along with Adam optimiser, gives an accuracy of around **43%**. The confusion matrix gives the classwise accuracy. As can be seen from the plots, this one trains slower than the other two values of learning rate, thereby giving significantly lesser accuracy.



(a) **Accuracy Plots**



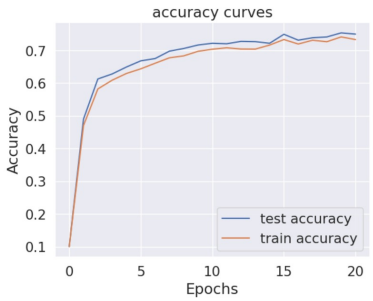
(b) **Loss PLots**



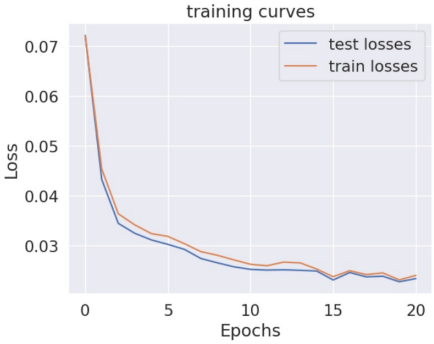
(c) **Confusion Matrix**

Variation in LR

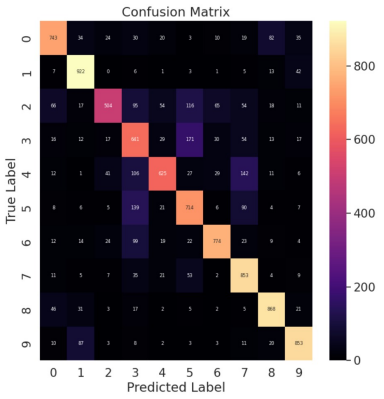
The scheduler - CosineAnnealingLR was picked as it works well on small datasets, CIFAR-10 being one. It can help prevent overfitting and improve generalization. The cosine annealing function reduces the learning rate gradually, allowing the model to converge more slowly and potentially find a better solution. We see the accuracy on using a scheduler is 74 percent. Similar to the accuracy obtained on using a fixed lr of 0.001.



(a) **Accuracy Plots**



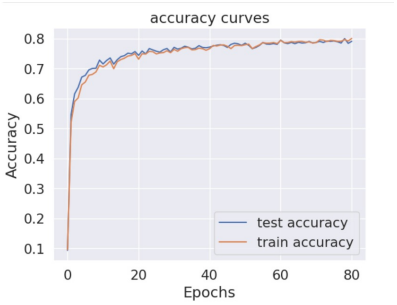
(b) **Loss PLots**



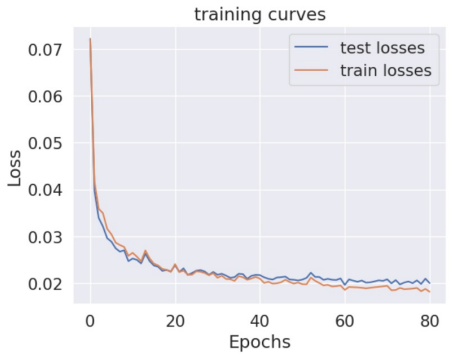
(c) **Confusion Matrix**

Number of Training Epochs

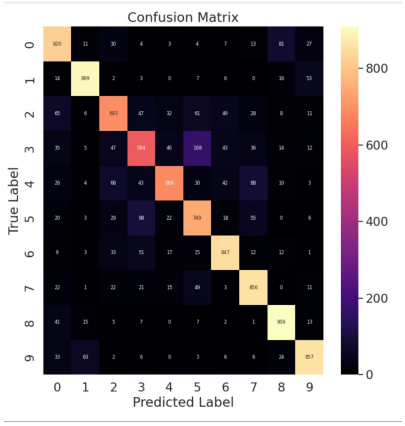
1. **Epoch = 80**



(a) Accuracy Plots

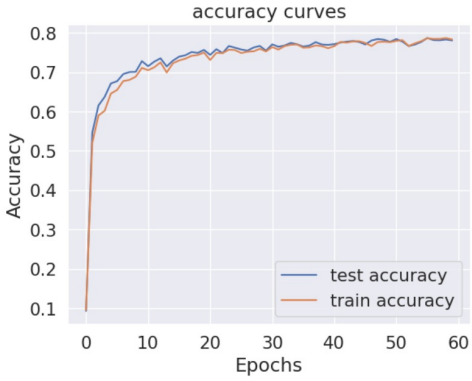


(b) Loss PLots



(c) Confusion Matrix

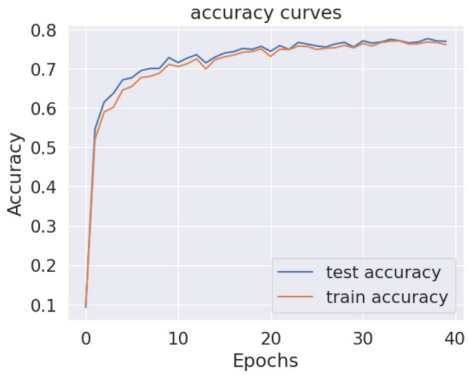
2. **Epoch = 60**



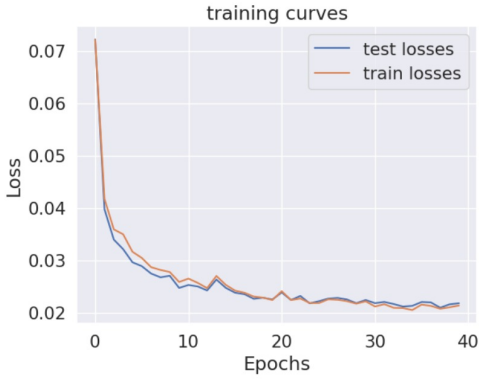
(a) Accuracy Plots

(b) Loss PLots

3. **Epoch = 40**



(a) Accuracy Plots



(b) Loss PLots

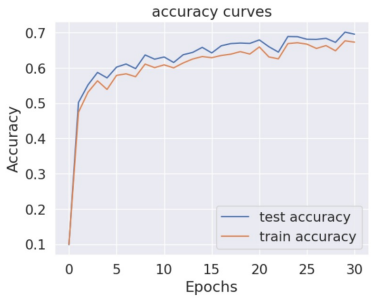
As shown from the above training and validation loss (and accuracy) curves, training the model for longer does not significantly contribute to model performance. We see only a slight improvement in accuracy on training for 80 epochs as opposed to training for only 20 epochs. However, it is important to note that even upon training for

80 epochs, the test accuracy curve is very closely attached to the training accuracy curve. This indicates that we can train for some more epochs without degradation in model performance due to overfitting.

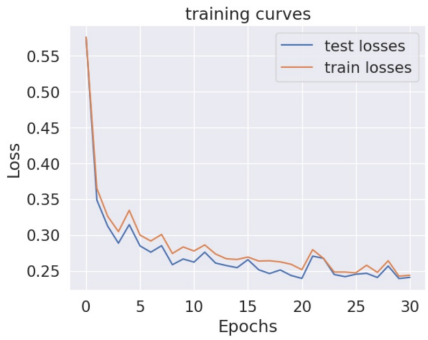
Batch Size

Here for all batch sizes, number of epochs is et to 30 and lr=0.001. Cross Entropy Loss Function is used.

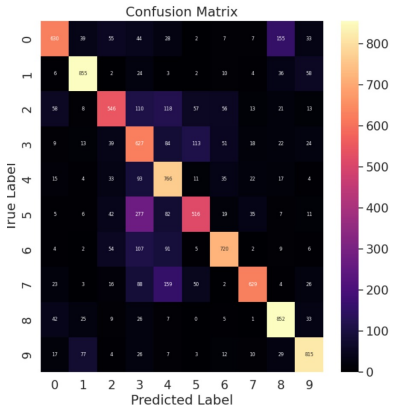
1. **Batch Size = 4** The accuracy comes out to be 69%



(a) Accuracy Plots

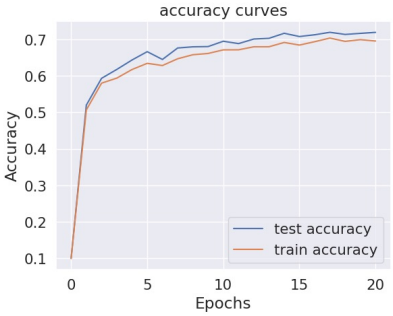


(b) Loss PLots

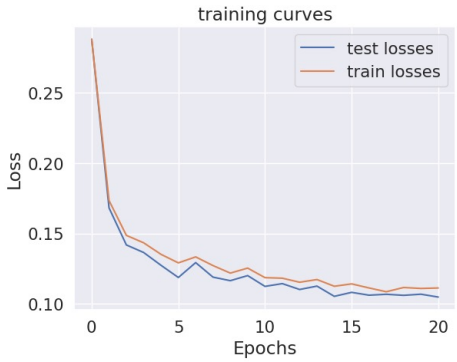


(c) Confusion Matrix

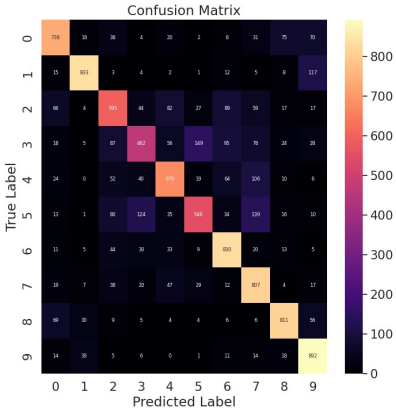
2. **Batch Size = 8** The accuracy comes out to be 71%



(a) Accuracy Plots

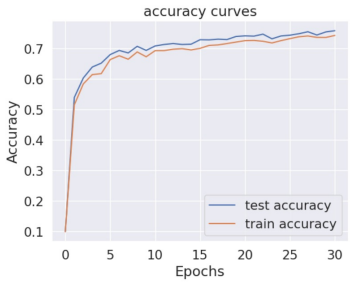


(b) Loss PLots

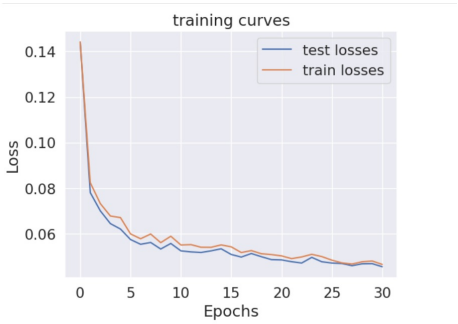


(c) Confusion Matrix

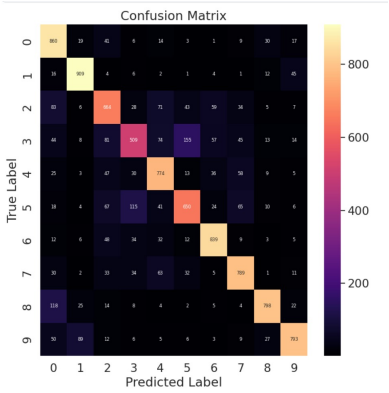
3. **Batch Size = 16** The accuracy comes out to be around 75%



(a) Accuracy Plots



(b) Loss PLots



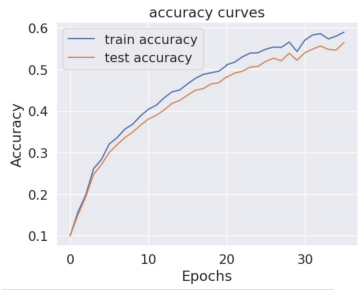
(c) Confusion Matrix

Accuracy increases with batch size. Different from SGD Optimiser.

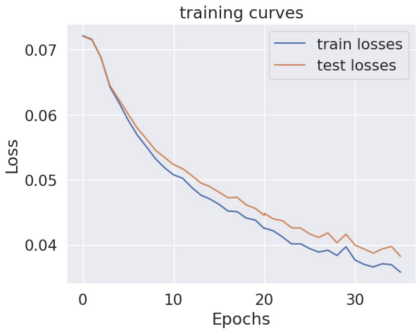
Hyper-parameter Tuning - SGD Optimiser

Learning Rate (LR)

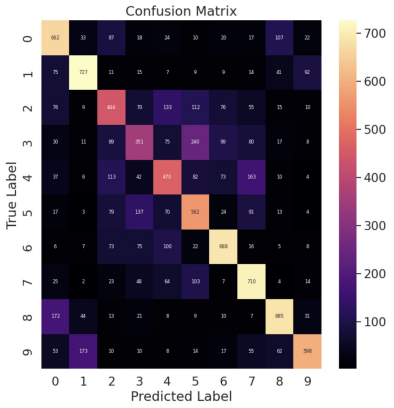
- **LR = 0.001:** Training the model using a batch size of 32 and 20 epochs, along with SGD optimiser, gives an accuracy of around **56%**. The confusion matrix gives the classwise accuracy. As can be seen from the loss plots, the test and training accuracies are very close together indicating minimal overfitting. Furthermore, we can see that training for additional epochs (36 instead of 20) led to a consistent increase in both train and test accuracies at similar rates. This indicates that further learning is possible with this hyperparameter setup.



(a) Accuracy Plots

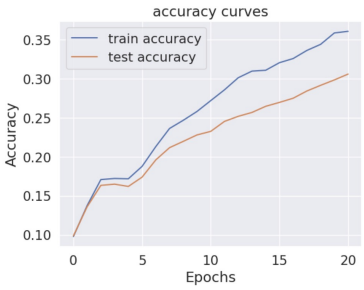


(b) Loss PLots

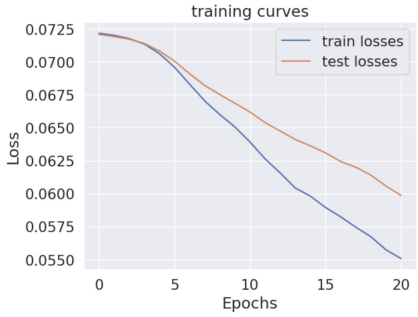


(c) Confusion Matrix

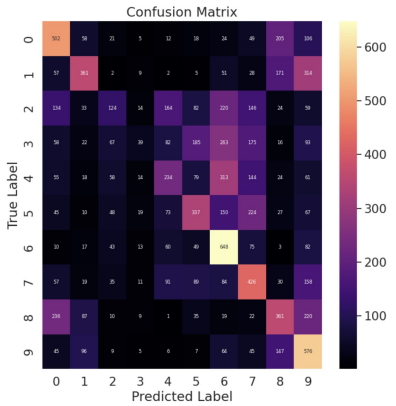
- **LR = 0.0005:** Training the model using a batch size of 32 and 20 epochs, along with SGD optimiser, gives an accuracy of around **38%**. The confusion matrix gives the classwise accuracy. Because of a lower learning rate training is happening slowly. There is more scope of learning with a different hyperparameter setup



(a) Accuracy Plots

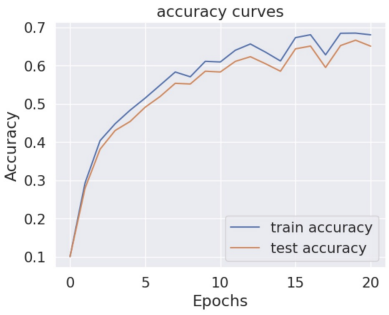


(b) Loss PLots

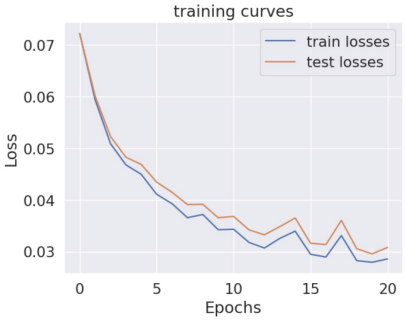


(c) Confusion Matrix

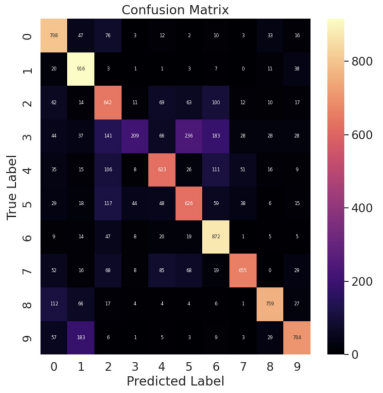
- **LR = 0.006:** Training the model using a batch size of 32 and 20 epochs, along with SGD optimiser, gives an accuracy of around **65%**. The confusion matrix gives the classwise accuracy. As we can see, the model trains faster to a lower loss value when compared to the previous learning rate choices. We note lower validation and training losses in 20 epochs with a learning rate of 6e-3 than losses in ¿30 epochs with a learning rate of 1e-3. Furthermore, we note that the validation and training curves are relatively close together – indicating that the model is training well so far without much overfitting.



(a) Accuracy Plots



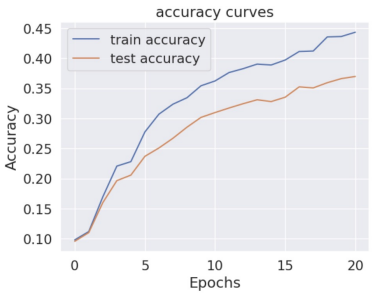
(b) Loss PLots



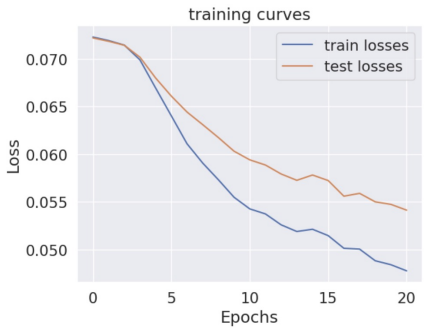
(c) Confusion Matrix

Variation in LR

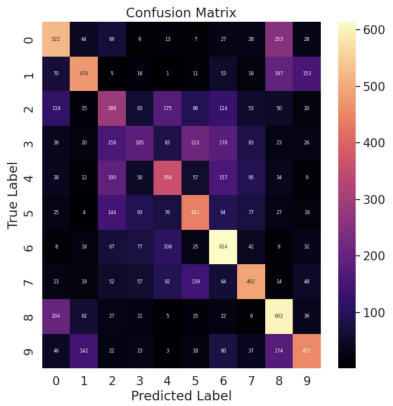
Using the Cosine Scheduler gives an accuracy of 44% which differs quite a lot from the accuracy that came when the lr was fixed at 0.001 i.e. 56%. A possible reason could be that schedulers quite often don't give optimal result on small datasets. Also, the rate of change of the learning rate could also have affected the accuracy. The graphs suggest some overfitting which could be due to the lr being stuck at the local minima.



(a) Accuracy Plots



(b) Loss PLots

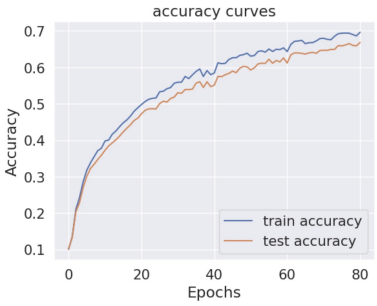


(c) Confusion Matrix

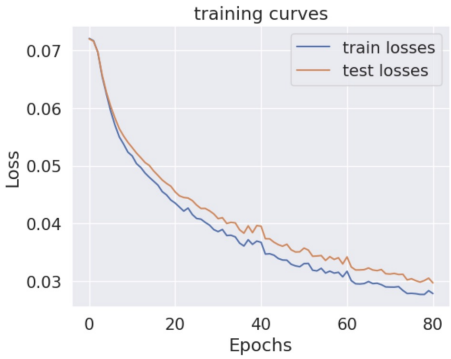
Number of Training Epochs

We get the following loss and accuracy curves upon training for 80, 60, and 40 epochs respectively. We fix our learning rate at 1e-3 and our batch size at 32 for the following experiments.

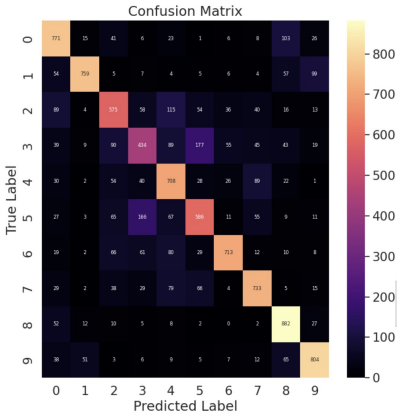
1. **Epoch = 80**



(a) Accuracy Plots

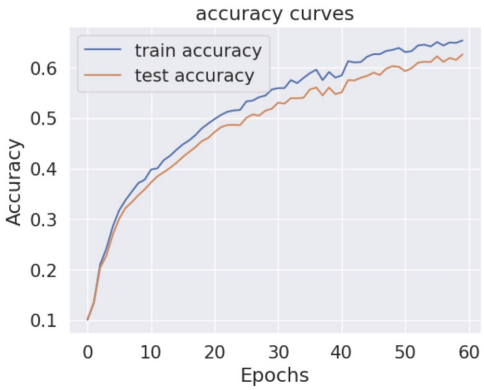


(b) Loss PLots

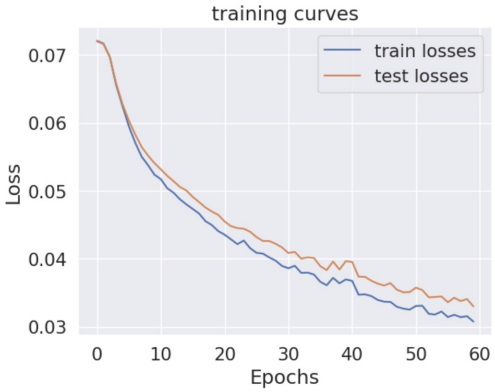


(c) Confusion Matrix

2. **Epoch = 60**

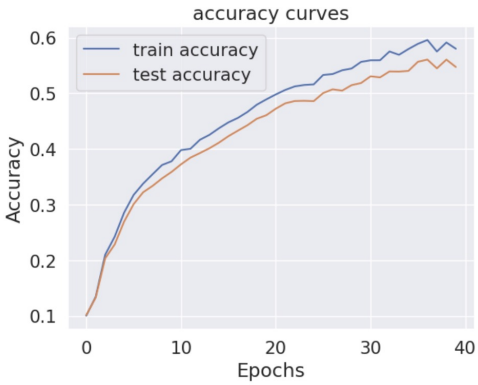


(a) Accuracy Plots

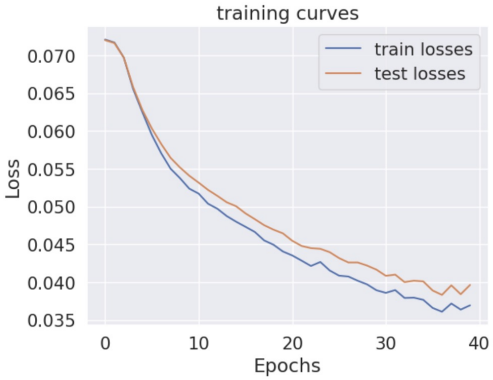


(b) Loss PLots

3. **Epoch = 40**



(a) Accuracy Plots

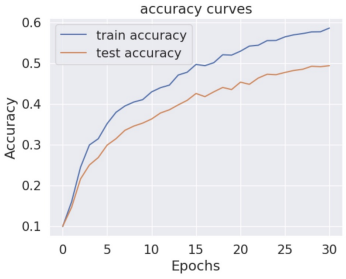


(b) Loss PLots

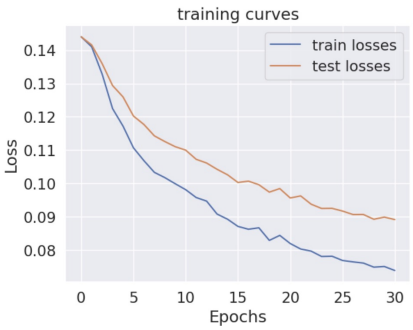
As shown from the above training and validation loss (and accuracy) curves, training the model for longer does not significantly contribute to model performance. We see only a slight improvement in accuracy (from 56% to 64%) on training for 80 epochs as opposed to training for only 20 epochs. However, it is important to note that even upon training for 80 epochs, the test accuracy curve is very closely attached to the training accuracy curve. This indicates that we can train for some more epochs without degradation in model performance due to overfitting.

Batch Size

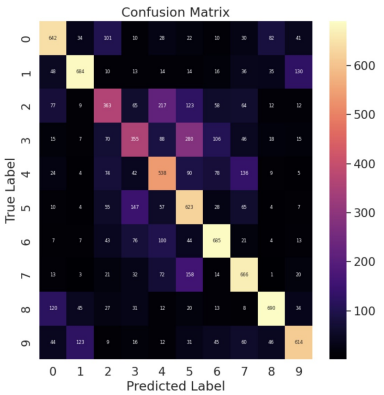
1. **Batch Size = 16** The accuracy comes out to be around 50%



(a) Accuracy Plots

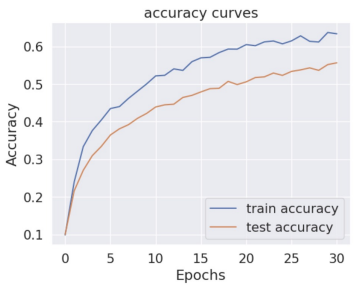


(b) Loss PLots

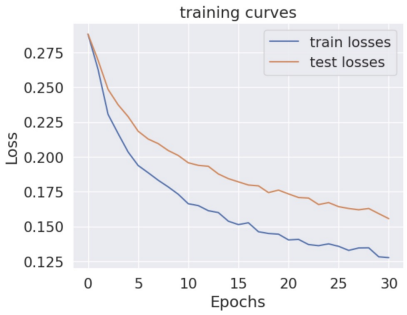


(c) Confusion Matrix

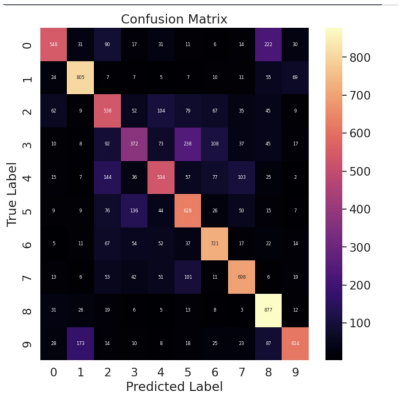
2. **Batch Size = 8** The accuracy comes out to be around 63.35%



(a) Accuracy Plots

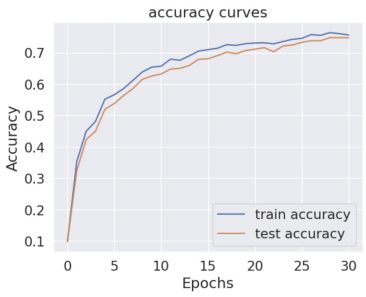


(b) Loss PLots

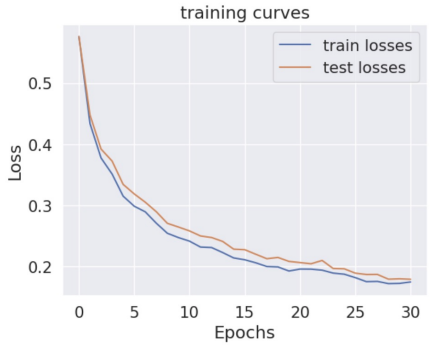


(c) Confusion Matrix

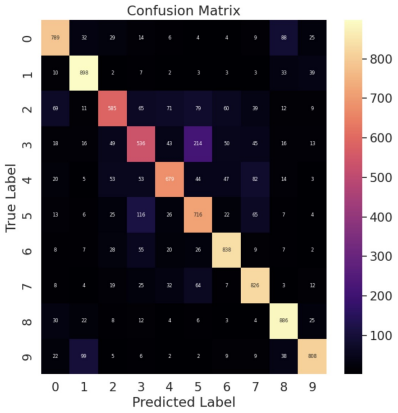
3. **Batch Size = 4** The accuracy comes out to be around 74%



(a) Accuracy Plots



(b) Loss PLots

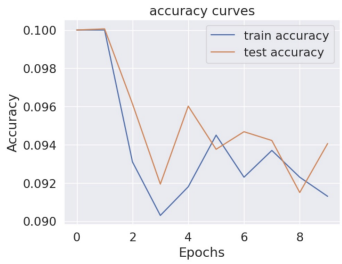


(c) Confusion Matrix

Decreasing batch size leads to increased accuracy

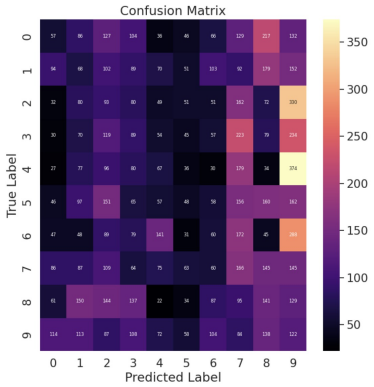
Effect of Loss Function

In this section, we shall discuss the effect of using a different loss function for training. In particular, we train our model using a KL divergence loss function with a constant learning rate of 1e-3 and batch size of 32. Due to computing resource constraints, we ran the simulation for only 10 epochs. The results for the simulations are as follows:



(a) Accuracy Plots

(b) Loss PLots

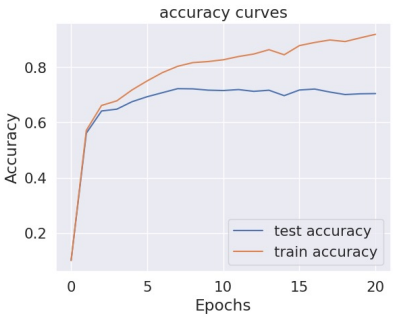


(c) Confusion Matrix

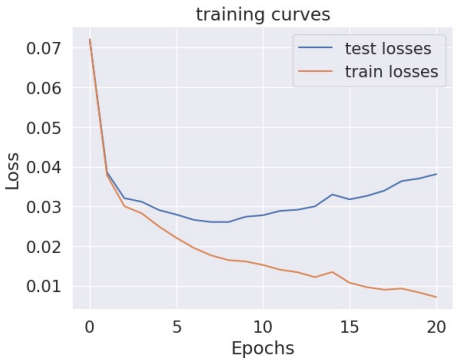
From the above figures, it is clear that KL divergence as a loss function is not suitable for classification tasks like CIFAR10. We note a steady decline in training losses (as expected since that is the metric being optimized for), but the trend in accuracies in validation loss is chaotic and haphazard. This indicates that while the model is optimizing for KL divergence loss, it is not getting any good at classifying the images as required.

Effect of Data Augmentation

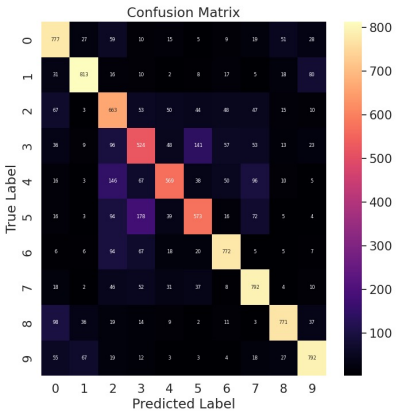
On removing data augmentation, the accuracy has come out to be **70.46%** as compared to an accuracy of 76% which highlights the benefits of Data Augmentation. Note that here I have used Adam Optimiser and Cross Entropy Loss Function over a batch size of 32, with 20 epochs and 0.001 as learning rate. Following are the plots and confusion matrix. It can be seen that Training accuracy is going beyond 80%. Also, there is decrease and then increase of validation loss. In case of data augmentation, even though the training accuracy is not that high, it can be seen that we get a much better overall validation accuracy. All these factors are indicative of overfitting which makes sense given that we don't have a large dataset which results in overfitting without the use of data augmentation.



(a) **Accuracy Plots**



(b) **Loss PLots**



(c) **Confusion Matrix**