Lab Course: Distributed Data Analytics
Exercise Sheet 7
Group 2 - Monday

Submitted by: Sruthy Annie Santhosh, 312213

Topic:Network Analysis: Image Classification - Part 2

- My System

  Hardware and Software SetUp:

Processor	M1 chip
Operating System	MacOS Monterey
Number of Cores	8
RAM	8 GB
Python version	3.8.8

The code editor used in this sheet : Colab and tensorboard for graphs
I have used Google Colab here as after the new update of M1 chip in MacOS, The kernel keeps dying while loading the Summarywriter on a jupyter notebook. The same was observed in other M1 chips also
The output images attached are screenshots.

- **Exercise 0: Baseline Model Implementation**

As given the exercise sheet, I have implemented a CNN model. I have chosen the output channels for convolutional layers randomly. The kernel size is taken as 3 for convolutional layers and 2 for maxpool layers. The stride is always taken as 1 and padding 0. For the fully connected layers the neurons are specified satisfying the formula of the neural network models. Finally, since the dataset used in CIFAR10, the input channel is 3 and output is 10 classes. I am using cross entropy loss and hence the softmax layer is not used.

```python
#Defining the convolutional neural network as given in question
class CNN(nn.Module):
    def __init__(self, num_classes):
        super(CNN, self).__init__()

        #Initialising each layer with the network params
        self.layer1 = nn.Sequential(
            nn.Conv2d(3, 6, kernel_size=3, stride=1, padding=0),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size = 2, stride = 1))
        self.layer2 = nn.Sequential(
            nn.Conv2d(6, 16, kernel_size=3, stride=1, padding=0),
            nn.ReLU())
        self.layer3 = nn.Sequential(
            nn.Conv2d(16,20,kernel_size=3, stride=1, padding=0),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size = 2, stride = 1))

        self.fc = nn.Linear(24*24*20, 200)
        self.relu = nn.ReLU()
        self.fc1 = nn.Linear(200, 100)
        self.relu1 = nn.ReLU()
        self.fc2 = nn.Linear(100, num_classes)
        self.relu3 = nn.ReLU()

    def forward(self, x):
        #Calling each layer
        out = self.layer1(x)
        out = self.layer2(out)
        out = self.layer3(out)
        out = out.reshape(out.size(0), -1)
        out = self.fc(out)
        out = self.relu(out)
        out = self.fc1(out)
        out = self.relu1(out)
        out = self.fc2(out)
        out = self.relu3(out)
        return out
```

The CIFAR10 dataset is downloaded with no data augmentation and normalisation. Only half the data is loaded into the train loader for training as mentioned in the exercise sheet.

```python
#Download the train and test datasets without any augmentation or normalization

train_dataset_cifar10 = torchvision.datasets.CIFAR10(root='./data', train=True,
                                    download=True,
                                    transform=transforms.ToTensor()
                                   )
test_dataset_cifar10 = torchvision.datasets.CIFAR10(root='./data',train=False,
                                    download=True,
                                    transform=transforms.ToTensor()
                                     )
```

```python
#Getting only half the train data
trainset_1 = torch.utils.data.Subset(train_dataset_cifar10, list(range(0, 30000)))

#Loading the given CIFAR10 dataset on the loader
train_loader = torch.utils.data.DataLoader(dataset = trainset_1,
                                           batch_size = batch_size,
                                           shuffle = True
                                          )


test_loader = torch.utils.data.DataLoader(dataset = test_dataset_cifar10,
                                           batch_size = batch_size,
                                           shuffle = True
                                          )
```

I have used Adam optimizer and learning rate of 0.001 throughout unless otherwise mentioned. The batch size is taken as 50 and number of epochs 10. Tensorboard is used for visualising the losses and accuracies for training and testing. Tensorboard is loaded and logs are created using SummaryWriter. Losses and accuracies are taken of minibatches in each epoch as asked in the question.

```
#Initializing the model
model = CNN(num_classes).to(device)

#Setting the loss function
cost = nn.CrossEntropyLoss()

#Setting the optimiser
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

#Getting the total number of images in train loader
total_step = len(train_loader)
```

The training is done as follows:

- For each epoch, training is carried out
- Iterating through each batch of images

```
images = images.to(device)
labels = labels.to(device)

#Forward pass
outputs = model(images)
#Calculating the loss
loss = cost(outputs, labels)

# Backward and optimize
optimizer.zero_grad()
loss.backward()
optimizer.step()

#To find average of losses
running_loss += loss.item()
#Finding the predicted values and calculating accuracy
_, predicted = torch.max(outputs.data, 1)
total += labels.size(0)
correct += (predicted == labels).sum().item()
```

- The outputs are found and loss is propagated backwards
- The loss is appended and average of loss is found after each 100th batch of each epoch and printed
- The accuracy is found by comparing the predicted values and true values.
- The losses and accuracy are written to log files defined by summary.

```
#writing loss and accuracy to tensorboard for each step of epoch

if (i+1) % 100 == 0:
      print ('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f} for learning rate: {} using ADAM'
                .format(epoch+1, num_epochs, i+1, total_step, running_loss/100,learning_rate))
      print('Accuracy of the network on the train images: {:.4f} % for epoch: {} using ADAM'.format(100 * correct / total,epoch+1))
      with train_summary_writer_cifar10.as_default():

        tf.summary.scalar('Cifar10 training loss for each step in epochs using ADAM', running_loss / 100, epoch * total_step + i)
        tf.summary.scalar('Cifar10 train Accuracy for each step in epochs using ADAM ', 100 * correct / total, epoch * total_step + i)
```

Visualising in tensorboard is done using:

```
#Visualising in tensorboard
%tensorboard --logdir logs/tensorboard
```

Testing is carried out for all images in the test loader. For testing gradient is not computed and losses are also not propagated back. The losses for each image batch is appended and mean is taken to get the test loss for the given step pf each epoch. The accuracy of test set is also found out. These values are written to log files and visualised using tensorboard.

```
with torch.no_grad():
    correct = 0
    total = 0
    running_loss = 0.0

    #For each epoch
    for epoch in range(num_epochs):
#For all images in test loader
      for i, (images, labels) in enumerate(test_loader):
          images = images.to(device)
          labels = labels.to(device)

          #Forward pass
          outputs = model(images)
          loss = cost(outputs, labels)


          #Finding average loss and acuuracy
          running_loss += loss.item()
          _, predicted = torch.max(outputs.data, 1)
          total += labels.size(0)
          correct += (predicted == labels).sum().item()


          #Writing to tensorboard the test loss and accuracy
          if (i+1) % 100 == 0:
              print ('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f} for learning r
                     .format(epoch+1, num_epochs, i+1, total_step, runnin
              print('Accuracy of the network on the test images: {:.4f} % for
              with test_summary_writer_cifar10.as_default():

                  tf.summary.scalar('Cifar10 test loss for each step in epochs
                  tf.summary.scalar('Cifar10 test Accuracy for each step in ep
```

The same testloader and testing function is used for all exercises. Unless otherwise specified the model, training functions used are same as above

The outputs and graphs obtained from tensorboard for baseline are plotted below:

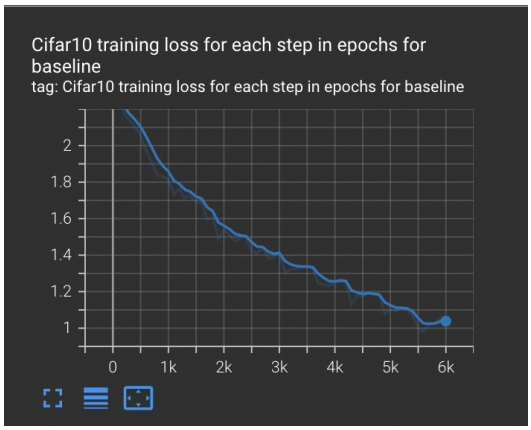
The Training loss and accuracy for baseline model

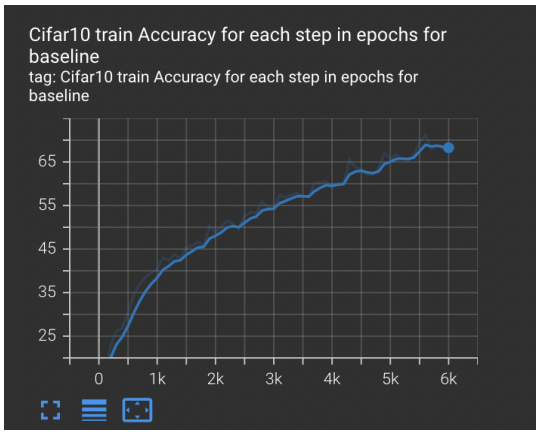
Output

```
Epoch [1/10], Step [100/600], Loss: 2.2699 for learning rate: 0.001 for baseline
Accuracy of the network on the train images: 15.1200 % for epoch: 1 for baseline
Epoch [1/10], Step [200/600], Loss: 2.1819 for learning rate: 0.001 for baseline
Accuracy of the network on the train images: 22.8200 % for epoch: 1 for baseline
Epoch [1/10], Step [300/600], Loss: 2.1346 for learning rate: 0.001 for baseline
Accuracy of the network on the train images: 26.0600 % for epoch: 1 for baseline
Epoch [1/10], Step [400/600], Loss: 2.1049 for learning rate: 0.001 for baseline
Accuracy of the network on the train images: 26.8800 % for epoch: 1 for baseline
Epoch [1/10], Step [500/600], Loss: 2.0541 for learning rate: 0.001 for baseline
Accuracy of the network on the train images: 30.5800 % for epoch: 1 for baseline
Epoch [1/10], Step [600/600], Loss: 1.9756 for learning rate: 0.001 for baseline
Accuracy of the network on the train images: 34.5000 % for epoch: 1 for baseline
Epoch [2/10], Step [100/600], Loss: 1.9079 for learning rate: 0.001 for baseline
Accuracy of the network on the train images: 36.9200 % for epoch: 2 for baseline
Epoch [2/10], Step [200/600], Loss: 1.8404 for learning rate: 0.001 for baseline
Accuracy of the network on the train images: 38.5000 % for epoch: 2 for baseline
Epoch [2/10], Step [300/600], Loss: 1.8304 for learning rate: 0.001 for baseline
Accuracy of the network on the train images: 39.5800 % for epoch: 2 for baseline
Epoch [2/10], Step [400/600], Loss: 1.8141 for learning rate: 0.001 for baseline
Accuracy of the network on the train images: 40.4400 % for epoch: 2 for baseline
Epoch [2/10], Step [500/600], Loss: 1.7330 for learning rate: 0.001 for baseline
Accuracy of the network on the train images: 42.9600 % for epoch: 2 for baseline
Epoch [2/10], Step [600/600], Loss: 1.7620 for learning rate: 0.001 for baseline
Accuracy of the network on the train images: 42.3800 % for epoch: 2 for baseline
Epoch [3/10], Step [100/600], Loss: 1.7129 for learning rate: 0.001 for baseline
Accuracy of the network on the train images: 43.8200 % for epoch: 3 for baseline
Epoch [3/10], Step [200/600], Loss: 1.7277 for learning rate: 0.001 for baseline
Accuracy of the network on the train images: 42.8200 % for epoch: 3 for baseline
Epoch [3/10], Step [300/600], Loss: 1.6837 for learning rate: 0.001 for baseline
Accuracy of the network on the train images: 45.4600 % for epoch: 3 for baseline
Epoch [3/10], Step [400/600], Loss: 1.6917 for learning rate: 0.001 for baseline
Accuracy of the network on the train images: 45.7800 % for epoch: 3 for baseline
Epoch [3/10], Step [500/600], Loss: 1.5929 for learning rate: 0.001 for baseline
Accuracy of the network on the train images: 46.6400 % for epoch: 3 for baseline
Epoch [3/10], Step [600/600], Loss: 1.6079 for learning rate: 0.001 for baseline
Accuracy of the network on the train images: 45.8200 % for epoch: 3 for baseline
Epoch [4/10], Step [100/600], Loss: 1.4873 for learning rate: 0.001 for baseline
✓ 1m 3s completed
```

```
Epoch [7/10], Step [600/600], Loss: 1.2549 for learning rate: 0.001 for baseline
Accuracy of the network on the train images: 60.2200 % for epoch: 7 for baseline
Epoch [8/10], Step [100/600], Loss: 1.1296 for learning rate: 0.001 for baseline
Accuracy of the network on the train images: 65.3800 % for epoch: 8 for baseline
Epoch [8/10], Step [200/600], Loss: 1.1789 for learning rate: 0.001 for baseline
Accuracy of the network on the train images: 63.7800 % for epoch: 8 for baseline
Epoch [8/10], Step [300/600], Loss: 1.1737 for learning rate: 0.001 for baseline
Accuracy of the network on the train images: 63.2800 % for epoch: 8 for baseline
Epoch [8/10], Step [400/600], Loss: 1.1977 for learning rate: 0.001 for baseline
Accuracy of the network on the train images: 62.1000 % for epoch: 8 for baseline
Epoch [8/10], Step [500/600], Loss: 1.1849 for learning rate: 0.001 for baseline
Accuracy of the network on the train images: 62.1000 % for epoch: 8 for baseline
Epoch [8/10], Step [600/600], Loss: 1.1770 for learning rate: 0.001 for baseline
Accuracy of the network on the train images: 63.4400 % for epoch: 8 for baseline
Epoch [9/10], Step [100/600], Loss: 1.0788 for learning rate: 0.001 for baseline
Accuracy of the network on the train images: 67.0800 % for epoch: 9 for baseline
Epoch [9/10], Step [200/600], Loss: 1.1005 for learning rate: 0.001 for baseline
Accuracy of the network on the train images: 65.7800 % for epoch: 9 for baseline
Epoch [9/10], Step [300/600], Loss: 1.0930 for learning rate: 0.001 for baseline
Accuracy of the network on the train images: 66.6800 % for epoch: 9 for baseline
Epoch [9/10], Step [400/600], Loss: 1.1103 for learning rate: 0.001 for baseline
Accuracy of the network on the train images: 65.8400 % for epoch: 9 for baseline
Epoch [9/10], Step [500/600], Loss: 1.1045 for learning rate: 0.001 for baseline
Accuracy of the network on the train images: 65.5400 % for epoch: 9 for baseline
Epoch [9/10], Step [600/600], Loss: 1.0685 for learning rate: 0.001 for baseline
Accuracy of the network on the train images: 66.4600 % for epoch: 9 for baseline
Epoch [10/10], Step [100/600], Loss: 0.9993 for learning rate: 0.001 for baseline
Accuracy of the network on the train images: 69.5000 % for epoch: 10 for baseline
Epoch [10/10], Step [200/600], Loss: 0.9824 for learning rate: 0.001 for baseline
Accuracy of the network on the train images: 71.2000 % for epoch: 10 for baseline
Epoch [10/10], Step [300/600], Loss: 1.0217 for learning rate: 0.001 for baseline
Accuracy of the network on the train images: 68.0600 % for epoch: 10 for baseline
Epoch [10/10], Step [400/600], Loss: 1.0272 for learning rate: 0.001 for baseline
Accuracy of the network on the train images: 68.9200 % for epoch: 10 for baseline
Epoch [10/10], Step [500/600], Loss: 1.0544 for learning rate: 0.001 for baseline
Accuracy of the network on the train images: 68.1600 % for epoch: 10 for baseline
Epoch [10/10], Step [600/600], Loss: 1.0403 for learning rate: 0.001 for baseline
Accuracy of the network on the train images: 67.9200 % for epoch: 10 for baseline
```

Graph in Tensorboard



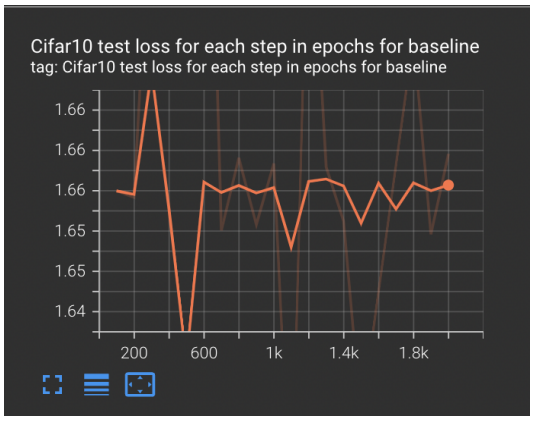


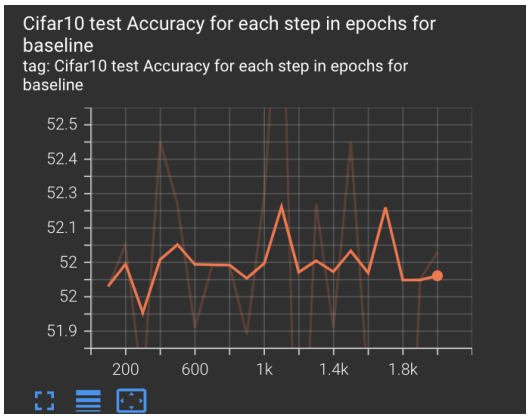
The Testing loss and accuracy for baseline model

Output

```
Epoch [1/10], Step [100/200], Loss: 1.6560 for learning rate: 0.001 for baseline
Accuracy of the network on the test images: 51.9800 % for epoch: 1 for baseline
Epoch [1/10], Step [200/200], Loss: 1.6553 for learning rate: 0.001 for baseline
Accuracy of the network on the test images: 52.1000 % for epoch: 1 for baseline
Epoch [2/10], Step [100/200], Loss: 1.6873 for learning rate: 0.001 for baseline
Accuracy of the network on the test images: 51.6800 % for epoch: 2 for baseline
Epoch [2/10], Step [200/200], Loss: 1.6240 for learning rate: 0.001 for baseline
Accuracy of the network on the test images: 52.4000 % for epoch: 2 for baseline
Epoch [3/10], Step [100/200], Loss: 1.5976 for learning rate: 0.001 for baseline
Accuracy of the network on the test images: 52.2200 % for epoch: 3 for baseline
Epoch [3/10], Step [200/200], Loss: 1.7138 for learning rate: 0.001 for baseline
Accuracy of the network on the test images: 51.8600 % for epoch: 3 for baseline
Epoch [4/10], Step [100/200], Loss: 1.6521 for learning rate: 0.001 for baseline
Accuracy of the network on the test images: 52.0400 % for epoch: 4 for baseline
Epoch [4/10], Step [200/200], Loss: 1.6593 for learning rate: 0.001 for baseline
Accuracy of the network on the test images: 52.0400 % for epoch: 4 for baseline
Epoch [5/10], Step [100/200], Loss: 1.6526 for learning rate: 0.001 for baseline
Accuracy of the network on the test images: 51.8400 % for epoch: 5 for baseline
Epoch [5/10], Step [200/200], Loss: 1.6587 for learning rate: 0.001 for baseline
Accuracy of the network on the test images: 52.2400 % for epoch: 5 for baseline
Epoch [6/10], Step [100/200], Loss: 1.6229 for learning rate: 0.001 for baseline
Accuracy of the network on the test images: 52.9800 % for epoch: 6 for baseline
Epoch [6/10], Step [200/200], Loss: 1.6884 for learning rate: 0.001 for baseline
Accuracy of the network on the test images: 51.1000 % for epoch: 6 for baseline
Epoch [7/10], Step [100/200], Loss: 1.6584 for learning rate: 0.001 for baseline
Accuracy of the network on the test images: 52.2200 % for epoch: 7 for baseline
Epoch [7/10], Step [200/200], Loss: 1.6530 for learning rate: 0.001 for baseline
Accuracy of the network on the test images: 51.8600 % for epoch: 7 for baseline
Epoch [8/10], Step [100/200], Loss: 1.6335 for learning rate: 0.001 for baseline
Accuracy of the network on the test images: 52.4000 % for epoch: 8 for baseline
Epoch [8/10], Step [200/200], Loss: 1.6779 for learning rate: 0.001 for baseline
Accuracy of the network on the test images: 51.6800 % for epoch: 8 for baseline
Epoch [9/10], Step [100/200], Loss: 1.6403 for learning rate: 0.001 for baseline
Accuracy of the network on the test images: 53.2400 % for epoch: 9 for baseline
Epoch [9/10], Step [200/200], Loss: 1.6711 for learning rate: 0.001 for baseline
Accuracy of the network on the test images: 50.8400 % for epoch: 9 for baseline
Epoch [10/10], Step [100/200], Loss: 1.6517 for learning rate: 0.001 for baseline
Accuracy of the network on the test images: 52.0000 % for epoch: 10 for baseline
Epoch [10/10], Step [200/200], Loss: 1.6596 for learning rate: 0.001 for baseline
Accuracy of the network on the test images: 52.0800 % for epoch: 10 for baseline
```

Graph in Tensorboard





Observation:

We observe that the training losses decrease and training accuracy increase after each epoch. The final training accuracy is approx 67%.

The test loss increases and decreases throughout, being around 1.66. The testing accuracy also varies throughout. Final testing accuracy is around 51%. This is less than the training accuracy due to overfitting. This result is as expected. Now we will compare these results after different techniques.

- ## Exercise 1: Normalization Effect (CNN)

In this exercise, we try improving the data using data augmentation and normalization and then training the model.

1. Data Augmentation

   This is the process of artificially 'increasing' our dataset by adding translation, scaling and flipping to the images to fabricate examples for training.
   Here I am doing horizontal flip with probability 0.5 and randomcrop with padding=4 to the already selected training dataset. This data is loaded and used for training. The model used is same as baseline. Then testing is also done. The losses and accuracies for training and testing is plotted.

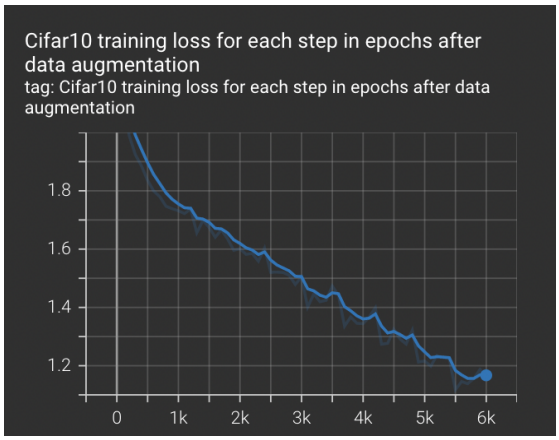
```
# performing Data Augmentation

train_transform_aug = Compose([                #doing flipping and padding
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomCrop(32, padding=4),
    transforms.ToTensor()])
```

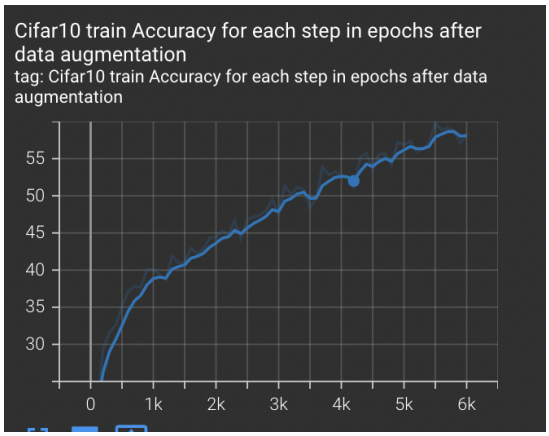
Training loss and accuracy after data augmentation

Output

```
Accuracy of the network on the train images: 45.3000 % for epoch: 4 after data augmentation
Epoch [4/10], Step [400/600], Loss: 1.5851 for learning rate: 0.001 after data augmentation
Accuracy of the network on the train images: 44.7800 % for epoch: 4 after data augmentation
Epoch [4/10], Step [500/600], Loss: 1.5585 for learning rate: 0.001 after data augmentation
Accuracy of the network on the train images: 46.6800 % for epoch: 4 after data augmentation
Epoch [4/10], Step [600/600], Loss: 1.6033 for learning rate: 0.001 after data augmentation
Accuracy of the network on the train images: 44.2200 % for epoch: 4 after data augmentation
Epoch [5/10], Step [100/600], Loss: 1.5207 for learning rate: 0.001 after data augmentation
Accuracy of the network on the train images: 46.8200 % for epoch: 5 after data augmentation
Epoch [5/10], Step [200/600], Loss: 1.5211 for learning rate: 0.001 after data augmentation
Accuracy of the network on the train images: 47.1600 % for epoch: 5 after data augmentation
Epoch [5/10], Step [300/600], Loss: 1.5199 for learning rate: 0.001 after data augmentation
Accuracy of the network on the train images: 47.3800 % for epoch: 5 after data augmentation
Epoch [5/10], Step [400/600], Loss: 1.5103 for learning rate: 0.001 after data augmentation
Accuracy of the network on the train images: 48.0200 % for epoch: 5 after data augmentation
Epoch [5/10], Step [500/600], Loss: 1.4789 for learning rate: 0.001 after data augmentation
Accuracy of the network on the train images: 49.4800 % for epoch: 5 after data augmentation
```

Graph in Tensorboard



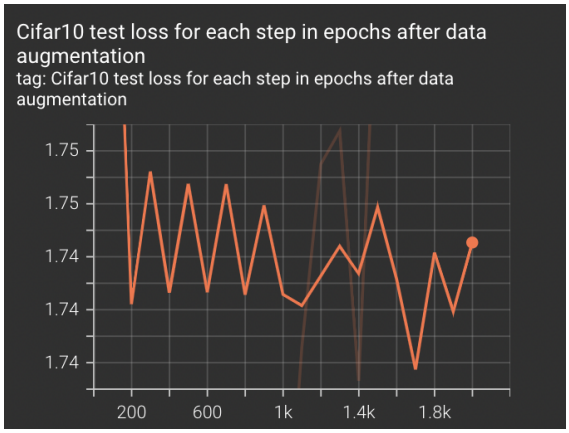


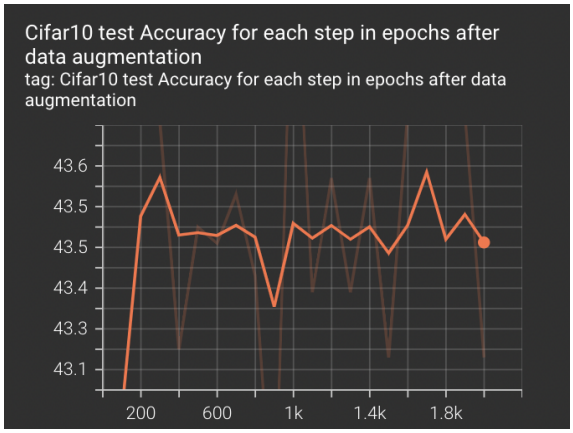
Testing loss and accuracy after data augmentation

Output

```
Epoch [7/10], Step [100/200], Loss: 1.7478 for learning rate: 0.001 after data augmentation
Accuracy of the network on the test images: 43.3400 % for epoch: 7 after data augmentation
Epoch [7/10], Step [200/200], Loss: 1.7383 for learning rate: 0.001 after data augmentation
Accuracy of the network on the test images: 43.6200 % for epoch: 7 after data augmentation
Epoch [8/10], Step [100/200], Loss: 1.7548 for learning rate: 0.001 after data augmentation
Accuracy of the network on the test images: 43.1800 % for epoch: 8 after data augmentation
Epoch [8/10], Step [200/200], Loss: 1.7313 for learning rate: 0.001 after data augmentation
Accuracy of the network on the test images: 43.7800 % for epoch: 8 after data augmentation
Epoch [9/10], Step [100/200], Loss: 1.7250 for learning rate: 0.001 after data augmentation
Accuracy of the network on the test images: 44.1600 % for epoch: 9 after data augmentation
Epoch [9/10], Step [200/200], Loss: 1.7611 for learning rate: 0.001 after data augmentation
Accuracy of the network on the test images: 42.8000 % for epoch: 9 after data augmentation
Epoch [10/10], Step [100/200], Loss: 1.7318 for learning rate: 0.001 after data augmentatio
Accuracy of the network on the test images: 43.7800 % for epoch: 10 after data augmentation
Epoch [10/10], Step [200/200], Loss: 1.7542 for learning rate: 0.001 after data augmentatio
Accuracy of the network on the test images: 43.1800 % for epoch: 10 after data augmentation
```

Graph in Tensorboard





We observe that similar to the baseline model, the training loss decreases and training accuracy increases with each epochs. But the final accuracy got is 57% which is less than that observed during baseline.

For testing, we observe a more visible pattern here. The testing loss seems to have a decreasing trend and testing accuracy an increasing trend. The final testing accuracy observed is 44%. But this accuracy is still less than that observed in baseline model(51%). But overall a better trend is observed and the testing accuracy is more closer to the training accuracy here - less overfitted.

---

2. Data Normalization

   Normalizing the input data helps remove the dataset artifacts that can cause poor model performance. For normalizing the data we need to find the mean and standard deviation of training set. Then normalization is done and model is trained. Rest of processes done are same.

```
print(train_dataset_cifar10.data.shape)
print(train_dataset_cifar10.data.mean(axis=(0,1,2))/255)
print(train_dataset_cifar10.data.std(axis=(0,1,2))/255)
```

```
(50000, 32, 32, 3)
[0.49139968 0.48215841 0.44653091]
[0.24703223 0.24348513 0.26158784]
```

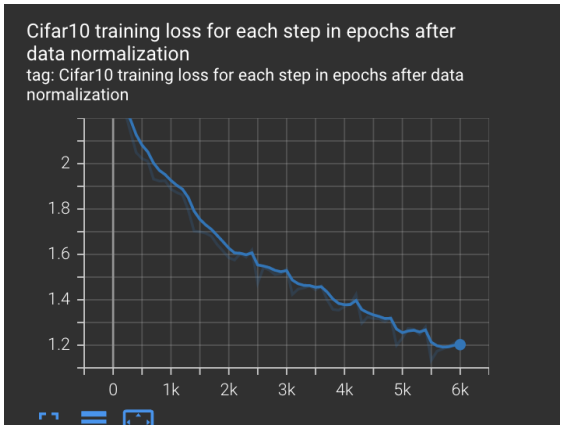
```
# Performing Data Normalization

normalize1 = Compose([
                  transforms.ToTensor(),
                  transforms.Normalize(mean=[0.49139968, 0.48215841, 0.44653091], std=[0.24703223, 0.24348513, 0.26158784])

                    ])
```

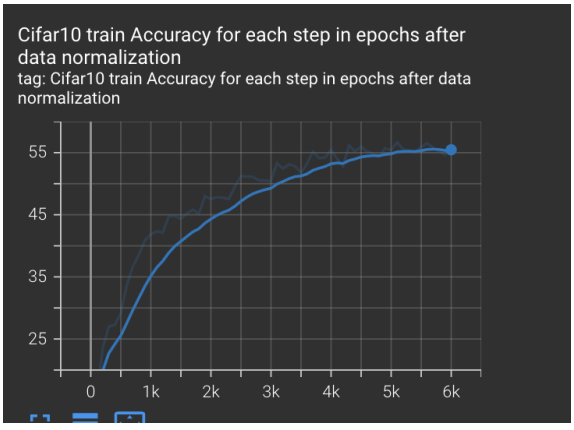
Training loss and accuracy after data normalisation

Output

```
Epoch [9/10], Step [100/600], Loss: 1.1993 for learning rate: 0.001 after data normalization
Accuracy of the network on the train images: 58.0400 % for epoch: 9 after data normalization
Epoch [9/10], Step [200/600], Loss: 1.2311 for learning rate: 0.001 after data normalization
Accuracy of the network on the train images: 56.8800 % for epoch: 9 after data normalization
Epoch [9/10], Step [300/600], Loss: 1.2736 for learning rate: 0.001 after data normalization
Accuracy of the network on the train images: 55.3600 % for epoch: 9 after data normalization
Epoch [9/10], Step [400/600], Loss: 1.2703 for learning rate: 0.001 after data normalization
Accuracy of the network on the train images: 55.8200 % for epoch: 9 after data normalization
Epoch [9/10], Step [500/600], Loss: 1.2474 for learning rate: 0.001 after data normalization
Accuracy of the network on the train images: 56.3800 % for epoch: 9 after data normalization
Epoch [9/10], Step [600/600], Loss: 1.2801 for learning rate: 0.001 after data normalization
Accuracy of the network on the train images: 54.7400 % for epoch: 9 after data normalization
Epoch [10/10], Step [100/600], Loss: 1.1326 for learning rate: 0.001 after data normalization
Accuracy of the network on the train images: 59.3000 % for epoch: 10 after data normalization
Epoch [10/10], Step [200/600], Loss: 1.1737 for learning rate: 0.001 after data normalization
Accuracy of the network on the train images: 57.9800 % for epoch: 10 after data normalization
Epoch [10/10], Step [300/600], Loss: 1.1835 for learning rate: 0.001 after data normalization
Accuracy of the network on the train images: 58.0200 % for epoch: 10 after data normalization
Epoch [10/10], Step [400/600], Loss: 1.1946 for learning rate: 0.001 after data normalization
Accuracy of the network on the train images: 57.6800 % for epoch: 10 after data normalization
Epoch [10/10], Step [500/600], Loss: 1.2096 for learning rate: 0.001 after data normalization
Accuracy of the network on the train images: 57.2600 % for epoch: 10 after data normalization
Epoch [10/10], Step [600/600], Loss: 1.2066 for learning rate: 0.001 after data normalization
Accuracy of the network on the train images: 57.7200 % for epoch: 10 after data normalization
```

Graph in Tensorboard



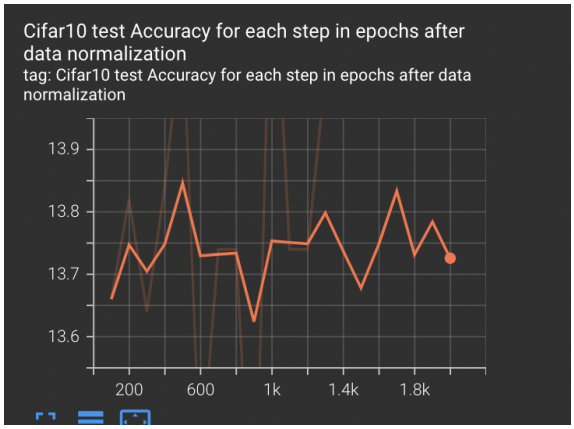


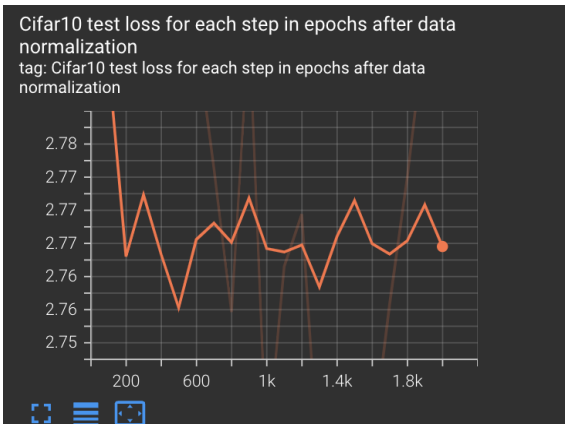
Testing loss and accuracy after data normalisation

Output

```
Accuracy of the network on the test images: 13.7400 % for epoch: 6 after data normalization
Epoch [7/10], Step [100/200], Loss: 2.7381 for learning rate: 0.001 after data normalizatio
Accuracy of the network on the test images: 14.0200 % for epoch: 7 after data normalization
Epoch [7/10], Step [200/200], Loss: 2.7947 for learning rate: 0.001 after data normalizatio
Accuracy of the network on the test images: 13.4600 % for epoch: 7 after data normalization
Epoch [8/10], Step [100/200], Loss: 2.7913 for learning rate: 0.001 after data normalizatio
Accuracy of the network on the test images: 13.4000 % for epoch: 8 after data normalization
Epoch [8/10], Step [200/200], Loss: 2.7415 for learning rate: 0.001 after data normalizatio
Accuracy of the network on the test images: 14.0800 % for epoch: 8 after data normalization
Epoch [9/10], Step [100/200], Loss: 2.7587 for learning rate: 0.001 after data normalizatio
Accuracy of the network on the test images: 14.2400 % for epoch: 9 after data normalization
Epoch [9/10], Step [200/200], Loss: 2.7741 for learning rate: 0.001 after data normalizatio
Accuracy of the network on the test images: 13.2400 % for epoch: 9 after data normalization
Epoch [10/10], Step [100/200], Loss: 2.7918 for learning rate: 0.001 after data normalizati
Accuracy of the network on the test images: 14.0400 % for epoch: 10 after data normalizatio
Epoch [10/10], Step [200/200], Loss: 2.7410 for learning rate: 0.001 after data normalizati
Accuracy of the network on the test images: 13.4400 % for epoch: 10 after data normalizatio
```

Graph in Tensorboard





We observe after data normalization, the training loss and accuracy trend is same. Loss is decreasing and accuracy increasing, but it is more smooth here than in incase of augmentation and baseline. But the accuracy obtained is less than that of baseline, but similar to that of augmentation (55%).

For testing, the losses show an overall trend of decreasing with epoch, though there are spikes in between. The accuracy also shows a slightly increasing trend with spikes, but the final accuracy obtained is around 13.8% which is less than the baseline and augmentation model. This value is very much less than the training accuracy also. Maybe for this model and dataset, normalization have a negative impact.

3. Data Augmentation and Normalization

   Here we apply both data augmentation and normalization to the training dataset and then train the model.

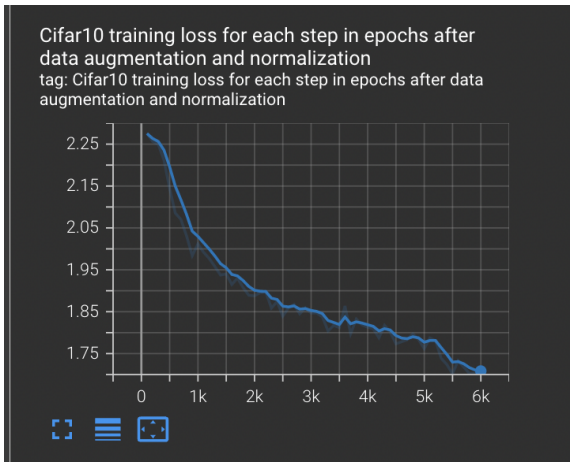
```
#performing data augmentation and normalization

train_transform_aug_norm = Compose([
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomCrop(32, padding=4),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.49139968, 0.48215841, 0.44653091], std=[0.24703223, 0.24348513, 0.26158784])
    ])
```

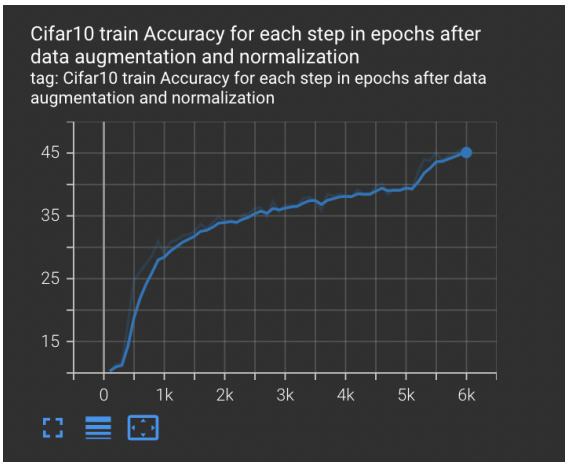
Training loss and accuracy after data augmentation and normalization

Output

```
Epoch [9/10], Step [300/600], Loss: 1.7872 for learning rate: 0.001 after data augmentation and normalization
Accuracy of the network on the train images: 39.1600 % for epoch: 9 after data augmentation and normalization
Epoch [9/10], Step [400/600], Loss: 1.7811 for learning rate: 0.001 after data augmentation and normalization
Accuracy of the network on the train images: 41.9600 % for epoch: 9 after data augmentation and normalization
Epoch [9/10], Step [500/600], Loss: 1.7387 for learning rate: 0.001 after data augmentation and normalization
Accuracy of the network on the train images: 43.9600 % for epoch: 9 after data augmentation and normalization
Epoch [9/10], Step [600/600], Loss: 1.7241 for learning rate: 0.001 after data augmentation and normalization
Accuracy of the network on the train images: 43.8000 % for epoch: 9 after data augmentation and normalization
Epoch [10/10], Step [100/600], Loss: 1.7019 for learning rate: 0.001 after data augmentation and normalization
Accuracy of the network on the train images: 45.1000 % for epoch: 10 after data augmentation and normalization
Epoch [10/10], Step [200/600], Loss: 1.7328 for learning rate: 0.001 after data augmentation and normalization
Accuracy of the network on the train images: 43.8000 % for epoch: 10 after data augmentation and normalization
Epoch [10/10], Step [300/600], Loss: 1.7177 for learning rate: 0.001 after data augmentation and normalization
Accuracy of the network on the train images: 44.5800 % for epoch: 10 after data augmentation and normalization
Epoch [10/10], Step [400/600], Loss: 1.7023 for learning rate: 0.001 after data augmentation and normalization
Accuracy of the network on the train images: 44.9600 % for epoch: 10 after data augmentation and normalization
Epoch [10/10], Step [500/600], Loss: 1.7027 for learning rate: 0.001 after data augmentation and normalization
Accuracy of the network on the train images: 45.3800 % for epoch: 10 after data augmentation and normalization
Epoch [10/10], Step [600/600], Loss: 1.7048 for learning rate: 0.001 after data augmentation and normalization
Accuracy of the network on the train images: 45.5000 % for epoch: 10 after data augmentation and normalization
```

Graph in Tensorboard



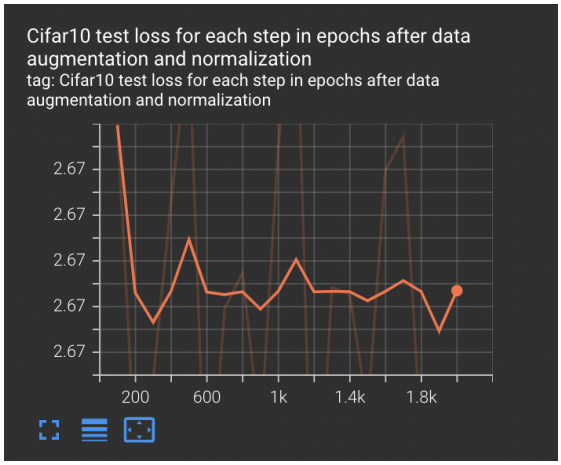


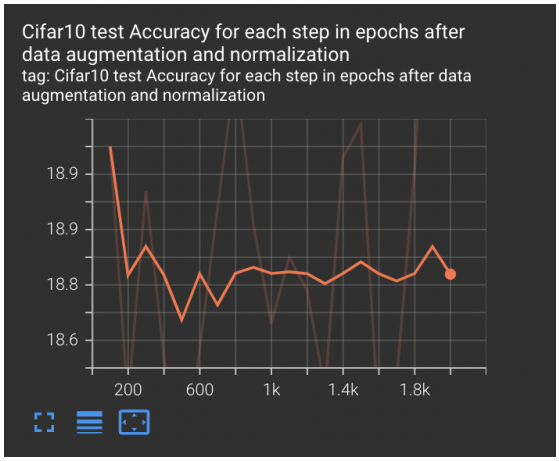
Testing loss and accuracy after data augmentation and normalization

Output

```
Epoch [6/10], Step [200/200], Loss: 2.6560 for learning rate: 0.001 after data augmentation and normalization
Accuracy of the network on the test images: 18.7400 % for epoch: 6 after data augmentation and normalization
Epoch [7/10], Step [100/200], Loss: 2.6698 for learning rate: 0.001 after data augmentation and normalization
Accuracy of the network on the test images: 18.5600 % for epoch: 7 after data augmentation and normalization
Epoch [7/10], Step [200/200], Loss: 2.6695 for learning rate: 0.001 after data augmentation and normalization
Accuracy of the network on the test images: 18.9800 % for epoch: 7 after data augmentation and normalization
Epoch [8/10], Step [100/200], Loss: 2.6644 for learning rate: 0.001 after data augmentation and normalization
Accuracy of the network on the test images: 19.0400 % for epoch: 8 after data augmentation and normalization
Epoch [8/10], Step [200/200], Loss: 2.6749 for learning rate: 0.001 after data augmentation and normalization
Accuracy of the network on the test images: 18.5000 % for epoch: 8 after data augmentation and normalization
Epoch [9/10], Step [100/200], Loss: 2.6764 for learning rate: 0.001 after data augmentation and normalization
Accuracy of the network on the test images: 18.5800 % for epoch: 9 after data augmentation and normalization
Epoch [9/10], Step [200/200], Loss: 2.6629 for learning rate: 0.001 after data augmentation and normalization
Accuracy of the network on the test images: 18.9600 % for epoch: 9 after data augmentation and normalization
Epoch [10/10], Step [100/200], Loss: 2.6423 for learning rate: 0.001 after data augmentation and normalization
Accuracy of the network on the test images: 19.5400 % for epoch: 10 after data augmentation and normalization
Epoch [10/10], Step [200/200], Loss: 2.6971 for learning rate: 0.001 after data augmentation and normalization
Accuracy of the network on the test images: 18.0000 % for epoch: 10 after data augmentation and normalization
```

Graph in Tensorboard





We observe that during training, the losses are decreasing and accuracy increasing as expected. The maximum accuracy achieved is 45% which is lesser than what was obtained in all the above cases.

For testing, the see that the loss decreases steepily initially and then spikes are there but still there is a slight trend of decrease. The accuracy spikes up and down with being approx 18.9 % which is less than what was observed in baseline, but better than what was obtained during data normalization alone. Here we see that the testing accuracy is still less than training accuracy.

Overall I observe that data augmentation gives the best model in this case, which is least overfitted and having best testing accuracy. Hence I will be using this data for further exercises.

- **Exercise 2: Network Regularization (CNN)**

In this exercise, we use regularization techniques. They are useful in learning a generalizable solution and help in avoiding overfitting. For the Deep Neural Network regularization is generally achieved by using a dropout technique, L1,L2 regularization among many other techniques.

1. Dropout technique

   Adding dropout layers to a network simply means we stochastically turn off a set number of neurons in our Fully connected layer to prevent the model from overfitting on training data. I have given the probability of neurons to be dropped out as 0.25. One dropout layer is added after each fully connected layers. A new model is defined here. Rest all parameters like optimizer and learning rate used is same as the baseline model. Also the training and testing processes remain the same.

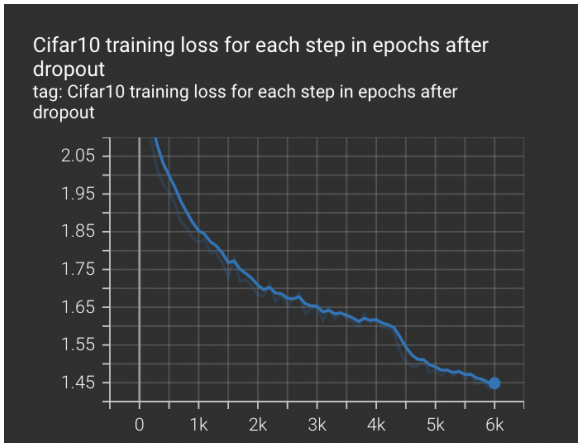
```
# Adding Dropout layer to the model

class CNN_dropout(nn.Module):
    def __init__(self, num_classes):
        super(CNN_dropout, self).__init__()
        #Initialising each layer with the network params
        self.layer1 = nn.Sequential(
            nn.Conv2d(3, 6, kernel_size=3, stride=1, padding=0),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size = 2, stride = 1))
        self.layer2 = nn.Sequential(
            nn.Conv2d(6, 16, kernel_size=3, stride=1, padding=0),
            nn.ReLU())
        self.layer3 = nn.Sequential(
            nn.Conv2d(16,20,kernel_size=3, stride=1, padding=0),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size = 2, stride = 1))

        self.fc = nn.Linear(24*24*20, 200)
        self.relu = nn.ReLU()
        self.fc1 = nn.Linear(200, 100)
        self.relu1 = nn.ReLU()
        self.fc2 = nn.Linear(100, num_classes)
        self.relu3 = nn.ReLU()

        self.dropout = nn.Dropout(0.25) #dropout

    def forward(self, x):
        #Calling each layer
        out = self.layer1(x)
        out = self.layer2(out)
        out = self.layer3(out)
        out = out.reshape(out.size(0), -1)
        out = self.fc(out)
        out = self.relu(out)
        out = self.dropout(out) #dropout layer added with fc
        out = self.fc1(out)
        out = self.relu1(out)
        out = self.dropout(out)
        out = self.fc2(out)
        out = self.relu3(out)
        return out
```

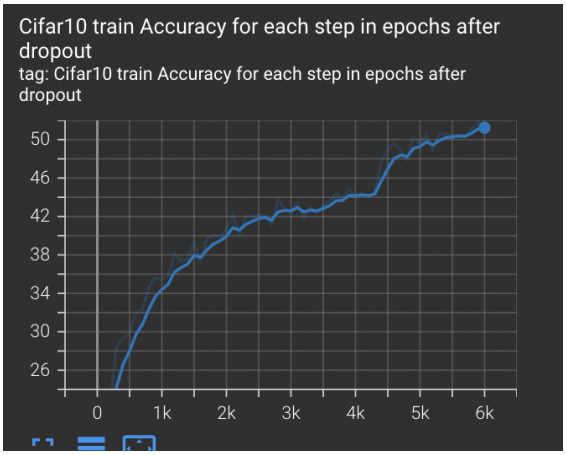
Training loss and accuracy after dropout

Output

```
Epoch [9/10], Step [400/600], Loss: 1.4839 for learning rate: 0.001 after dropout
Accuracy of the network on the train images: 48.9800 % for epoch: 9 after dropout
Epoch [9/10], Step [500/600], Loss: 1.4677 for learning rate: 0.001 after dropout
Accuracy of the network on the train images: 50.6000 % for epoch: 9 after dropout
Epoch [9/10], Step [600/600], Loss: 1.4833 for learning rate: 0.001 after dropout
Accuracy of the network on the train images: 50.6400 % for epoch: 9 after dropout
Epoch [10/10], Step [100/600], Loss: 1.4608 for learning rate: 0.001 after dropout
Accuracy of the network on the train images: 50.3800 % for epoch: 10 after dropout
Epoch [10/10], Step [200/600], Loss: 1.4719 for learning rate: 0.001 after dropout
Accuracy of the network on the train images: 50.5000 % for epoch: 10 after dropout
Epoch [10/10], Step [300/600], Loss: 1.4469 for learning rate: 0.001 after dropout
Accuracy of the network on the train images: 50.3600 % for epoch: 10 after dropout
Epoch [10/10], Step [400/600], Loss: 1.4535 for learning rate: 0.001 after dropout
Accuracy of the network on the train images: 51.1400 % for epoch: 10 after dropout
Epoch [10/10], Step [500/600], Loss: 1.4356 for learning rate: 0.001 after dropout
Accuracy of the network on the train images: 51.7200 % for epoch: 10 after dropout
Epoch [10/10], Step [600/600], Loss: 1.4477 for learning rate: 0.001 after dropout
Accuracy of the network on the train images: 51.4600 % for epoch: 10 after dropout
```

Graph in Tensorboard



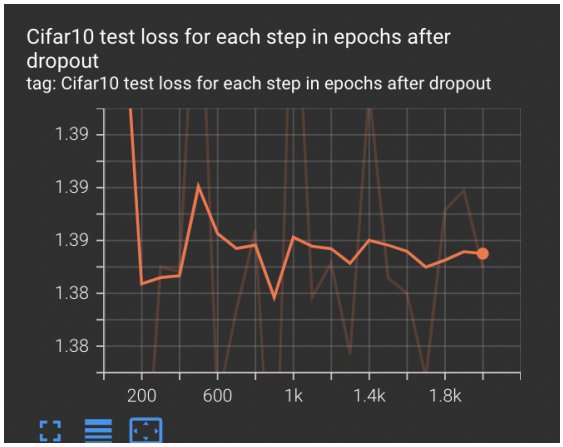


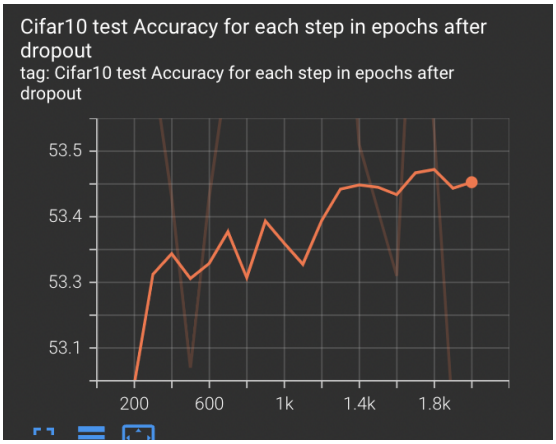
Testing loss and accuracy after dropout

Output

```
Epoch [6/10], Step [200/200], Loss: 1.3843 for learning rate: 0.001 after dropout
Accuracy of the network on the test images: 53.9200 % for epoch: 6 after dropout
Epoch [7/10], Step [100/200], Loss: 1.3774 for learning rate: 0.001 after dropout
Accuracy of the network on the test images: 53.8400 % for epoch: 7 after dropout
Epoch [7/10], Step [200/200], Loss: 1.3972 for learning rate: 0.001 after dropout
Accuracy of the network on the test images: 53.4600 % for epoch: 7 after dropout
Epoch [8/10], Step [100/200], Loss: 1.3832 for learning rate: 0.001 after dropout
Accuracy of the network on the test images: 53.3600 % for epoch: 8 after dropout
Epoch [8/10], Step [200/200], Loss: 1.3820 for learning rate: 0.001 after dropout
Accuracy of the network on the test images: 53.2600 % for epoch: 8 after dropout
Epoch [9/10], Step [100/200], Loss: 1.3756 for learning rate: 0.001 after dropout
Accuracy of the network on the test images: 53.8000 % for epoch: 9 after dropout
Epoch [9/10], Step [200/200], Loss: 1.3883 for learning rate: 0.001 after dropout
Accuracy of the network on the test images: 53.4800 % for epoch: 9 after dropout
Epoch [10/10], Step [100/200], Loss: 1.3898 for learning rate: 0.001 after dropout
Accuracy of the network on the test images: 53.0400 % for epoch: 10 after dropout
Epoch [10/10], Step [200/200], Loss: 1.3839 for learning rate: 0.001 after dropout
Accuracy of the network on the test images: 53.5200 % for epoch: 10 after dropout
```

Graph in Tensorboard





We observe that for training, the losses and accuracies have the same trend as observed for baseline. The losses decreases and accuracy increases after each epoch. The final train accuracy is 51 % which is lesser than the baseline model.

For testing, the observe a more visible trend what was observed in the baseline model. The test losses seems to decrease after each epoch, with spikes and testing accuracy shows an overall increasing trend reaching 53.5 % . The testing accuracy is greater than the training accuracy which implies that the model is not overfitting. The testing accuracy is greater than that observed in baseline model as well as what was observed with data augmentation and normalization.

Hence adding dropout layers along with data augmentation improves the performance of the model and as shown the best performance till now.

---

2. L1 Regularization

   L1 regularization involves penalizing the weights of the network to encourage sparse weights.

   $$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^{N} |w_i|$$

   This done during the training process. The model used is the same as the baseline model.

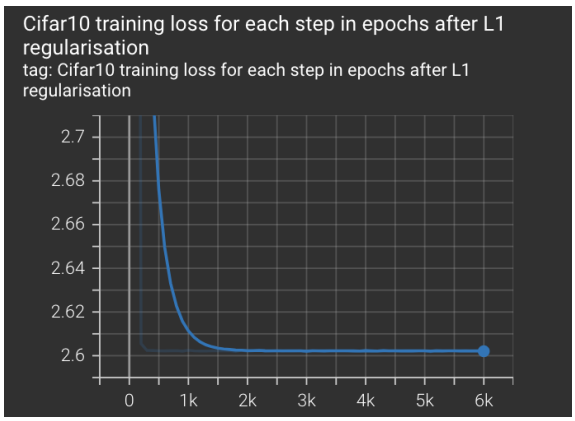
```python
#For each batch
  for i, (images, labels) in enumerate(train_loader):

      images = images.to(device)
      labels = labels.to(device)

      #Forward pass
      outputs = model(images)
      #Calculating the loss
      loss = cost(outputs, labels)
      # for L1 regularization

      l1_lambda = 0.001
      l1_norm = sum(p.abs().sum()
                    for p in model.parameters())

      loss = loss + l1_lambda * l1_norm
      # Backward and optimize
      optimizer.zero_grad()
      loss.backward()
      optimizer.step()
```

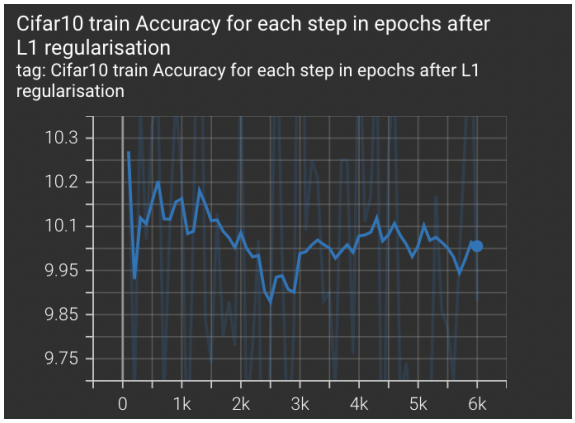
Training loss and accuracy after L1 regularisation

Output

```
Accuracy of the network on the train images: 10.6800 % for epoch: 9 after L1 regularisatio
Epoch [9/10], Step [400/600], Loss: 2.6026 for learning rate: 0.001 after L1 regularisatio
Accuracy of the network on the train images: 9.5600 % for epoch: 9 after L1 regularisation
Epoch [9/10], Step [500/600], Loss: 2.6021 for learning rate: 0.001 after L1 regularisatio
Accuracy of the network on the train images: 10.1200 % for epoch: 9 after L1 regularisatio
Epoch [9/10], Step [600/600], Loss: 2.6023 for learning rate: 0.001 after L1 regularisatio
Accuracy of the network on the train images: 9.8600 % for epoch: 9 after L1 regularisation
Epoch [10/10], Step [100/600], Loss: 2.6022 for learning rate: 0.001 after L1 regularisati
Accuracy of the network on the train images: 9.8200 % for epoch: 10 after L1 regularisatio
Epoch [10/10], Step [200/600], Loss: 2.6022 for learning rate: 0.001 after L1 regularisati
Accuracy of the network on the train images: 9.7000 % for epoch: 10 after L1 regularisatio
Epoch [10/10], Step [300/600], Loss: 2.6023 for learning rate: 0.001 after L1 regularisati
Accuracy of the network on the train images: 9.4400 % for epoch: 10 after L1 regularisatio
Epoch [10/10], Step [400/600], Loss: 2.6021 for learning rate: 0.001 after L1 regularisati
Accuracy of the network on the train images: 10.4200 % for epoch: 10 after L1 regularisati
Epoch [10/10], Step [500/600], Loss: 2.6022 for learning rate: 0.001 after L1 regularisati
Accuracy of the network on the train images: 10.5400 % for epoch: 10 after L1 regularisati
Epoch [10/10], Step [600/600], Loss: 2.6018 for learning rate: 0.001 after L1 regularisati
Accuracy of the network on the train images: 9.8800 % for epoch: 10 after L1 regularisatio
```

Graph in Tensorboard



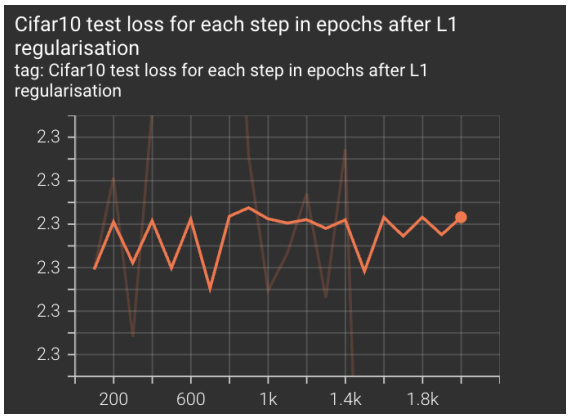


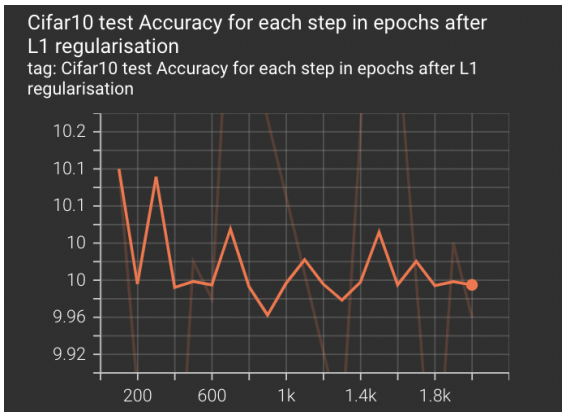
Testing loss and accuracy after L1 regularisation

Output

```
Accuracy of the network on the test images: 10.2000 % for epoch: 6 after L1 regularisation
Epoch [6/10], Step [200/200], Loss: 2.3026 for learning rate: 0.001 after L1 regularisatio
Accuracy of the network on the test images: 9.8000 % for epoch: 6 after L1 regularisation
Epoch [7/10], Step [100/200], Loss: 2.3026 for learning rate: 0.001 after L1 regularisatio
Accuracy of the network on the test images: 9.8400 % for epoch: 7 after L1 regularisation
Epoch [7/10], Step [200/200], Loss: 2.3027 for learning rate: 0.001 after L1 regularisatio
Accuracy of the network on the test images: 10.1600 % for epoch: 7 after L1 regularisation
Epoch [8/10], Step [100/200], Loss: 2.3024 for learning rate: 0.001 after L1 regularisatio
Accuracy of the network on the test images: 10.5200 % for epoch: 8 after L1 regularisation
Epoch [8/10], Step [200/200], Loss: 2.3028 for learning rate: 0.001 after L1 regularisatio
Accuracy of the network on the test images: 9.4800 % for epoch: 8 after L1 regularisation
Epoch [9/10], Step [100/200], Loss: 2.3025 for learning rate: 0.001 after L1 regularisatio
Accuracy of the network on the test images: 10.2600 % for epoch: 9 after L1 regularisation
Epoch [9/10], Step [200/200], Loss: 2.3027 for learning rate: 0.001 after L1 regularisatio
Accuracy of the network on the test images: 9.7400 % for epoch: 9 after L1 regularisation
Epoch [10/10], Step [100/200], Loss: 2.3025 for learning rate: 0.001 after L1 regularisati
Accuracy of the network on the test images: 10.0400 % for epoch: 10 after L1 regularisatio
Epoch [10/10], Step [200/200], Loss: 2.3027 for learning rate: 0.001 after L1 regularisati
Accuracy of the network on the test images: 9.9600 % for epoch: 10 after L1 regularisation
```

Graph in Tensorboard





We observe that during training we observe that losses decreases initially and then remains almost constant while accuracy show an overall decreasing trend with spikes. The training accuracy observed is around 10.2% which is lesser than that observed in baseline.This accuracy is lesser than the training accuracy of all the previously observed models.

For testing, the losses show a slightly increasing trend while accuracy shows a slight decrease with accuracy around 10 %. This is similar to the training accuracy observed. Hence while overfitting hasnot occurred, this accuracy is very much less than what was observed in baseline model. Hence L1 regularisation is not a good fit for this model and dataset.

---

3. L2 Regularization
   Involves penalizing the weights of the network to encourage them to fit on the unit circle. This regularisation is also done during training process and the model used is the baseline model itself.

$$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^{N} w_i^2$$

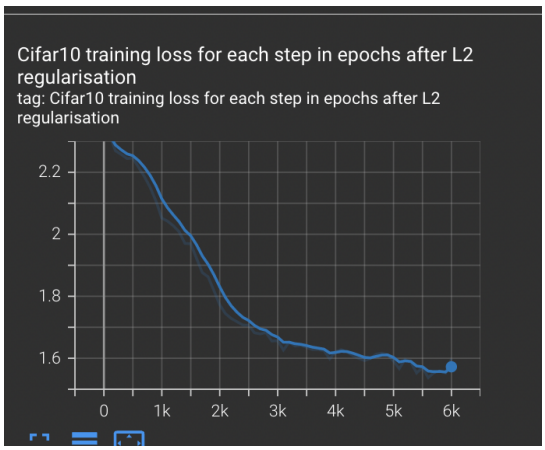
```
#For each batch
  for i, (images, labels) in enumerate(train_loader):

      images = images.to(device)
      labels = labels.to(device)

      #Forward pass
      outputs = model(images)
      #Calculating the loss
      loss = cost(outputs, labels)
      # for L2 regularization

      l2_lambda = 0.001
      l2_norm = sum(p.pow(2.0).sum()
                    for p in model.parameters())

      loss = loss + l2_lambda * l2_norm
      # Backward and optimize
      optimizer.zero_grad()
      loss.backward()
      optimizer.step()
```

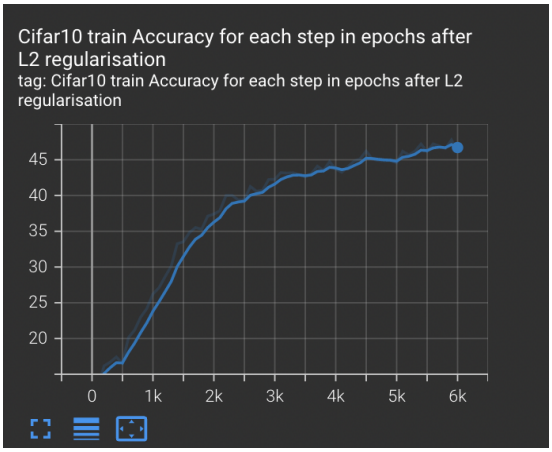
Training loss and accuracy after L2 regularisation

Output

```
Epoch [9/10], Step [300/600], Loss: 1.5664 for learning rate: 0.001 after L2 regularisation
Accuracy of the network on the train images: 46.1800 % for epoch: 9 after L2 regularisation
Epoch [9/10], Step [400/600], Loss: 1.5963 for learning rate: 0.001 after L2 regularisation
Accuracy of the network on the train images: 45.7000 % for epoch: 9 after L2 regularisation
Epoch [9/10], Step [500/600], Loss: 1.5874 for learning rate: 0.001 after L2 regularisation
Accuracy of the network on the train images: 46.1800 % for epoch: 9 after L2 regularisation
Epoch [9/10], Step [600/600], Loss: 1.5524 for learning rate: 0.001 after L2 regularisation
Accuracy of the network on the train images: 47.2000 % for epoch: 9 after L2 regularisation
Epoch [10/10], Step [100/600], Loss: 1.5697 for learning rate: 0.001 after L2 regularisation
Accuracy of the network on the train images: 46.1800 % for epoch: 10 after L2 regularisation
Epoch [10/10], Step [200/600], Loss: 1.5375 for learning rate: 0.001 after L2 regularisation
Accuracy of the network on the train images: 47.2200 % for epoch: 10 after L2 regularisation
Epoch [10/10], Step [300/600], Loss: 1.5536 for learning rate: 0.001 after L2 regularisation
Accuracy of the network on the train images: 46.9400 % for epoch: 10 after L2 regularisation
Epoch [10/10], Step [400/600], Loss: 1.5587 for learning rate: 0.001 after L2 regularisation
Accuracy of the network on the train images: 46.5800 % for epoch: 10 after L2 regularisation
Epoch [10/10], Step [500/600], Loss: 1.5529 for learning rate: 0.001 after L2 regularisation
Accuracy of the network on the train images: 47.7800 % for epoch: 10 after L2 regularisation
Epoch [10/10], Step [600/600], Loss: 1.5979 for learning rate: 0.001 after L2 regularisation
Accuracy of the network on the train images: 46.0800 % for epoch: 10 after L2 regularisation
```

Graph in Tensorboard



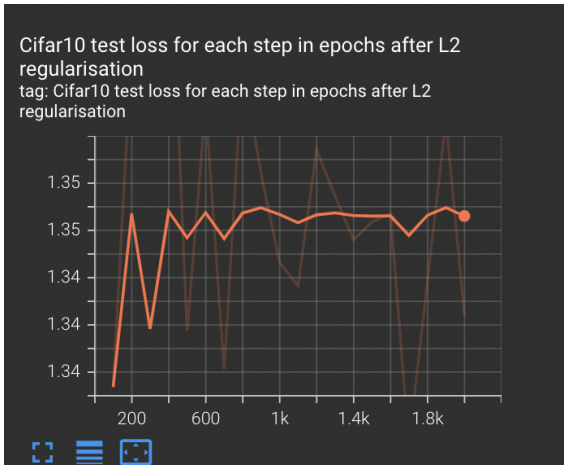


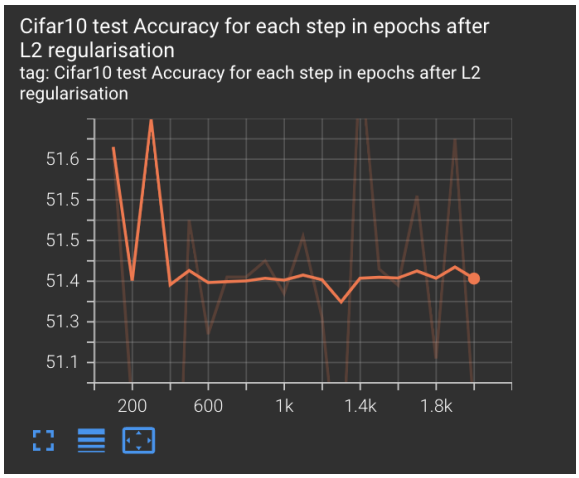
Testing loss and accuracy after L2 regularisation

Output

```
Epoch [6/10], Step [100/200], Loss: 1.3447 for learning rate: 0.001 after L2 regularisation
Accuracy of the network on the test images: 51.4600 % for epoch: 6 after L2 regularisation
Epoch [6/10], Step [200/200], Loss: 1.3504 for learning rate: 0.001 after L2 regularisation
Accuracy of the network on the test images: 51.2600 % for epoch: 6 after L2 regularisation
Epoch [7/10], Step [100/200], Loss: 1.3485 for learning rate: 0.001 after L2 regularisation
Accuracy of the network on the test images: 50.8200 % for epoch: 7 after L2 regularisation
Epoch [7/10], Step [200/200], Loss: 1.3466 for learning rate: 0.001 after L2 regularisation
Accuracy of the network on the test images: 51.9000 % for epoch: 7 after L2 regularisation
Epoch [8/10], Step [100/200], Loss: 1.3474 for learning rate: 0.001 after L2 regularisation
Accuracy of the network on the test images: 51.3800 % for epoch: 8 after L2 regularisation
Epoch [8/10], Step [200/200], Loss: 1.3477 for learning rate: 0.001 after L2 regularisation
Accuracy of the network on the test images: 51.3400 % for epoch: 8 after L2 regularisation
Epoch [9/10], Step [100/200], Loss: 1.3380 for learning rate: 0.001 after L2 regularisation
Accuracy of the network on the test images: 51.5600 % for epoch: 9 after L2 regularisation
Epoch [9/10], Step [200/200], Loss: 1.3570 for learning rate: 0.001 after L2 regularisation
Accuracy of the network on the test images: 51.1600 % for epoch: 9 after L2 regularisation
Epoch [10/10], Step [100/200], Loss: 1.3517 for learning rate: 0.001 after L2 regularisation
Accuracy of the network on the test images: 51.7000 % for epoch: 10 after L2 regularisation
Epoch [10/10], Step [200/200], Loss: 1.3434 for learning rate: 0.001 after L2 regularisation
Accuracy of the network on the test images: 51.0200 % for epoch: 10 after L2 regularisation
```

Graph in Tensorboard





We observe that during training, the losses decrease and accuracy increases after each epochs. The final accuracy obtained is 46 %. This is lesser than that obtained by

baseline model but better than L1 regularisation. But this is still less than that observed by dropout model.

While testing the loss initially increases and then remains almost constant. The accuracy initially shows high peak and then decreases and shows a constant trend. The final accuracy obtained is around 51 %. This accuracy is same as that obtained by the baseline model. The testing accuracy is also greater than training accuracy showing that the model is not over fitted. Hence we can say that L2 regularisation is better suited to this model and dataset than L1 regularisation.

But overall the best regularisation method is dropout. Hence for the next exercise dropout model is being used, with the other parameters and datasets same.

- ## Exercise 3: Optimizers (CNN)

Here we experiment with 2 Optimizers: SGD and ADAM with different learning rates to see how robust they are. As mentioned I have taken the best regularization- dropout model and best data preprocessing, data augmentation here. The training and testing processes remain the same.

I am experimenting with 2 learning rates 0.001 and 0.01 for both the optimizers.

1. ADAM with learning rate 0.001

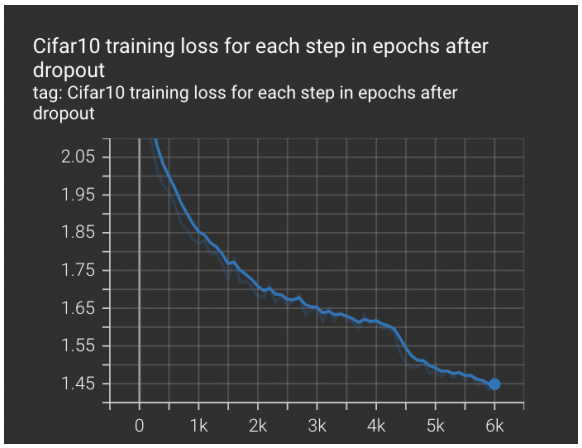
```
#Initializing the best model from ex 2
model = CNN_dropout(num_classes).to(device)

#Setting the loss function
cost = nn.CrossEntropyLoss()

#Setting the optimiser as SGD
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

#Getting the total number of images in train loader
total_step = len(train_loader)
```

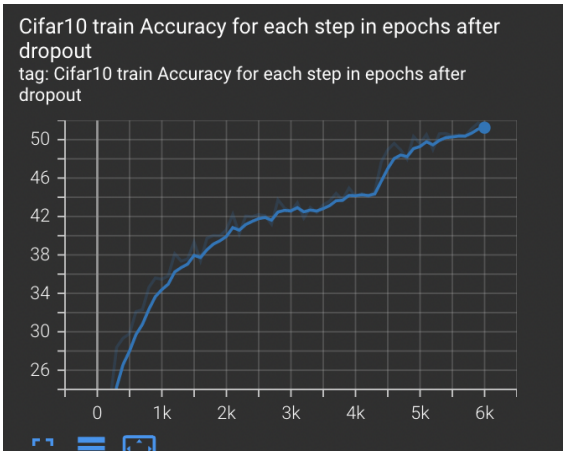
Training loss and accuracy using ADAM, learning rate - 0.001

Output

```
Epoch [9/10], Step [400/600], Loss: 1.4839 for learning rate: 0.001 after dropout
Accuracy of the network on the train images: 48.9800 % for epoch: 9 after dropout
Epoch [9/10], Step [500/600], Loss: 1.4677 for learning rate: 0.001 after dropout
Accuracy of the network on the train images: 50.6000 % for epoch: 9 after dropout
Epoch [9/10], Step [600/600], Loss: 1.4833 for learning rate: 0.001 after dropout
Accuracy of the network on the train images: 50.6400 % for epoch: 9 after dropout
Epoch [10/10], Step [100/600], Loss: 1.4608 for learning rate: 0.001 after dropout
Accuracy of the network on the train images: 50.3800 % for epoch: 10 after dropout
Epoch [10/10], Step [200/600], Loss: 1.4719 for learning rate: 0.001 after dropout
Accuracy of the network on the train images: 50.5000 % for epoch: 10 after dropout
Epoch [10/10], Step [300/600], Loss: 1.4469 for learning rate: 0.001 after dropout
Accuracy of the network on the train images: 50.3600 % for epoch: 10 after dropout
Epoch [10/10], Step [400/600], Loss: 1.4535 for learning rate: 0.001 after dropout
Accuracy of the network on the train images: 51.1400 % for epoch: 10 after dropout
Epoch [10/10], Step [500/600], Loss: 1.4356 for learning rate: 0.001 after dropout
Accuracy of the network on the train images: 51.7200 % for epoch: 10 after dropout
Epoch [10/10], Step [600/600], Loss: 1.4477 for learning rate: 0.001 after dropout
Accuracy of the network on the train images: 51.4600 % for epoch: 10 after dropout
```

Graph in Tensorboard



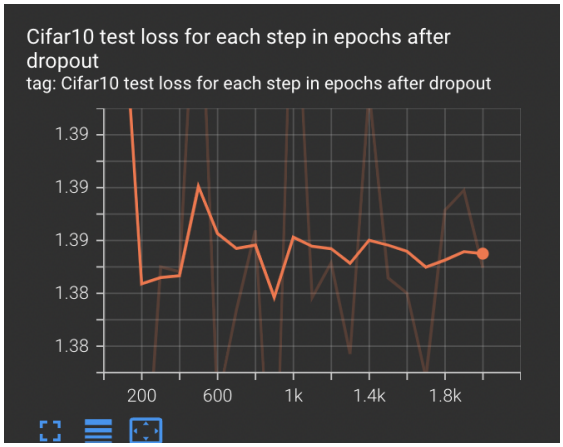


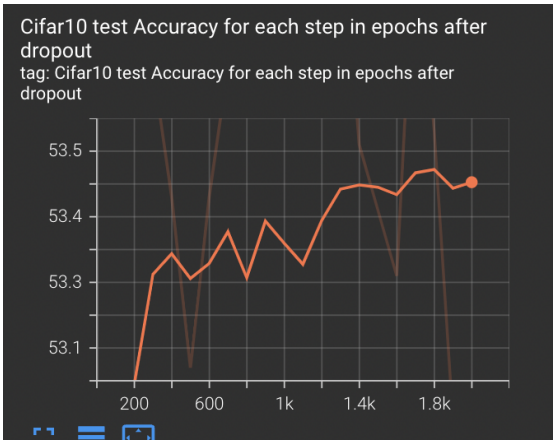
Testing loss and accuracy using ADAM, learning rate - 0.001

Output

```
Epoch [6/10], Step [200/200], Loss: 1.3843 for learning rate: 0.001 after dropout
Accuracy of the network on the test images: 53.9200 % for epoch: 6 after dropout
Epoch [7/10], Step [100/200], Loss: 1.3774 for learning rate: 0.001 after dropout
Accuracy of the network on the test images: 53.8400 % for epoch: 7 after dropout
Epoch [7/10], Step [200/200], Loss: 1.3972 for learning rate: 0.001 after dropout
Accuracy of the network on the test images: 53.4600 % for epoch: 7 after dropout
Epoch [8/10], Step [100/200], Loss: 1.3832 for learning rate: 0.001 after dropout
Accuracy of the network on the test images: 53.3600 % for epoch: 8 after dropout
Epoch [8/10], Step [200/200], Loss: 1.3820 for learning rate: 0.001 after dropout
Accuracy of the network on the test images: 53.2600 % for epoch: 8 after dropout
Epoch [9/10], Step [100/200], Loss: 1.3756 for learning rate: 0.001 after dropout
Accuracy of the network on the test images: 53.8000 % for epoch: 9 after dropout
Epoch [9/10], Step [200/200], Loss: 1.3883 for learning rate: 0.001 after dropout
Accuracy of the network on the test images: 53.4800 % for epoch: 9 after dropout
Epoch [10/10], Step [100/200], Loss: 1.3898 for learning rate: 0.001 after dropout
Accuracy of the network on the test images: 53.0400 % for epoch: 10 after dropout
Epoch [10/10], Step [200/200], Loss: 1.3839 for learning rate: 0.001 after dropout
Accuracy of the network on the test images: 53.5200 % for epoch: 10 after dropout
```

Graph in Tensorboard





We observe that during training, the losses decrease and accuracy increase over epochs as expected. The final accuracy is around 51%. While testing the losses initially decrease and then show spikes. The test accuracy shows an increasing trend with final accuracy around 53.5 %. This depicts a model which is not overfitted and having good test accuracy.

---

2. SGD with learning rate 0.001

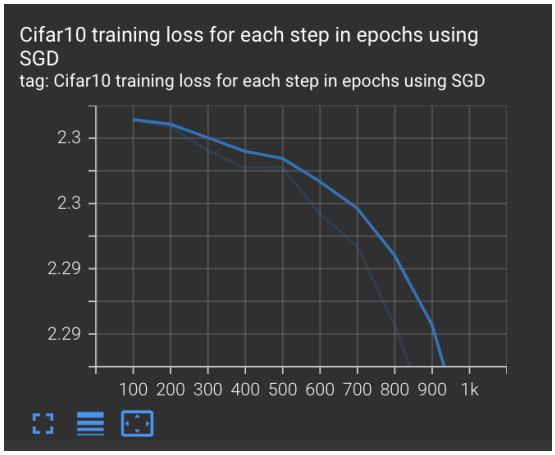
```
#Initializing the best model from ex 2
model = CNN_dropout(num_classes).to(device)

#Setting the loss function
cost = nn.CrossEntropyLoss()

#Setting the optimiser as SGD
optimizer = torch.optim.SGD(model.parameters(), lr=0.001)

#Getting the total number of images in train loader
total_step = len(train_loader)
```

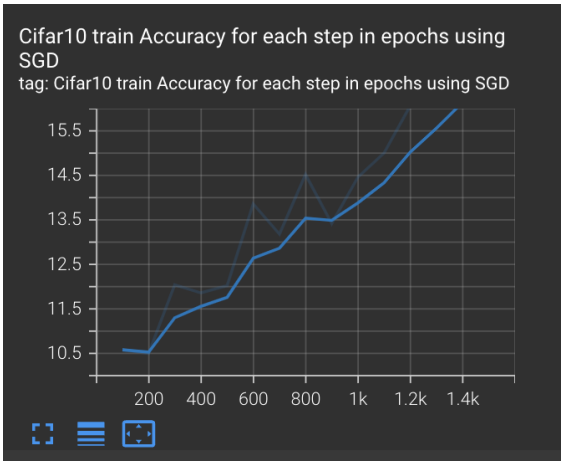
Training loss and accuracy using SGD, learning rate - 0.001

Output

```
Epoch [9/10], Step [300/600], Loss: 2.3022 for learning rate: 0.001 using SGD
Accuracy of the network on the train images: 9.8400 % for epoch: 9 using SGD
Epoch [9/10], Step [400/600], Loss: 2.3027 for learning rate: 0.001 using SGD
Accuracy of the network on the train images: 9.8800 % for epoch: 9 using SGD
Epoch [9/10], Step [500/600], Loss: 2.3029 for learning rate: 0.001 using SGD
Accuracy of the network on the train images: 10.0200 % for epoch: 9 using SGD
Epoch [9/10], Step [600/600], Loss: 2.3026 for learning rate: 0.001 using SGD
Accuracy of the network on the train images: 10.1800 % for epoch: 9 using SGD
Epoch [10/10], Step [100/600], Loss: 2.3025 for learning rate: 0.001 using SGD
Accuracy of the network on the train images: 10.6800 % for epoch: 10 using SGD
Epoch [10/10], Step [200/600], Loss: 2.3024 for learning rate: 0.001 using SGD
Accuracy of the network on the train images: 10.4600 % for epoch: 10 using SGD
Epoch [10/10], Step [300/600], Loss: 2.3024 for learning rate: 0.001 using SGD
Accuracy of the network on the train images: 10.2000 % for epoch: 10 using SGD
Epoch [10/10], Step [400/600], Loss: 2.3023 for learning rate: 0.001 using SGD
Accuracy of the network on the train images: 9.7200 % for epoch: 10 using SGD
Epoch [10/10], Step [500/600], Loss: 2.3027 for learning rate: 0.001 using SGD
Accuracy of the network on the train images: 10.7400 % for epoch: 10 using SGD
Epoch [10/10], Step [600/600], Loss: 2.3025 for learning rate: 0.001 using SGD
Accuracy of the network on the train images: 10.6600 % for epoch: 10 using SGD
```

Graph in Tensorboard



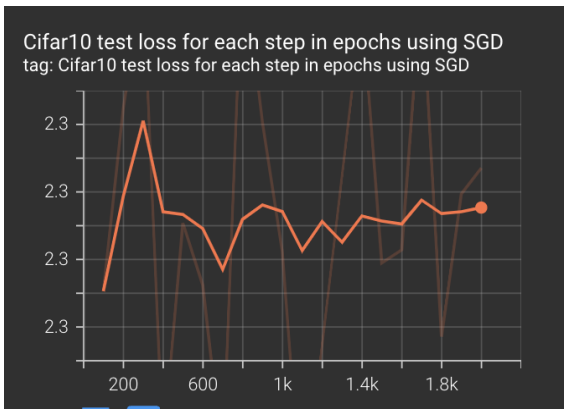


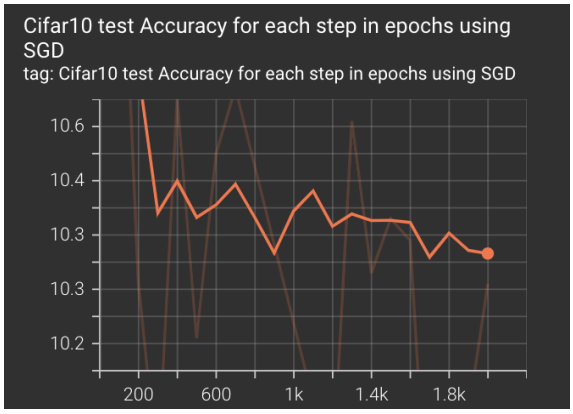
Testing loss and accuracy using SGD, learning rate - 0.001

Output

```
Epoch [6/10], Step [100/200], Loss: 2.3020 for learning rate: 0.001 using SGD
Accuracy of the network on the test images: 10.6800 % for epoch: 6 using SGD
Epoch [6/10], Step [200/200], Loss: 2.3027 for learning rate: 0.001 using SGD
Accuracy of the network on the test images: 9.9000 % for epoch: 6 using SGD
Epoch [7/10], Step [100/200], Loss: 2.3021 for learning rate: 0.001 using SGD
Accuracy of the network on the test images: 10.5600 % for epoch: 7 using SGD
Epoch [7/10], Step [200/200], Loss: 2.3027 for learning rate: 0.001 using SGD
Accuracy of the network on the test images: 10.2800 % for epoch: 7 using SGD
Epoch [8/10], Step [100/200], Loss: 2.3023 for learning rate: 0.001 using SGD
Accuracy of the network on the test images: 10.3800 % for epoch: 8 using SGD
Epoch [8/10], Step [200/200], Loss: 2.3024 for learning rate: 0.001 using SGD
Accuracy of the network on the test images: 10.3400 % for epoch: 8 using SGD
Epoch [9/10], Step [100/200], Loss: 2.3028 for learning rate: 0.001 using SGD
Accuracy of the network on the test images: 9.7400 % for epoch: 9 using SGD
Epoch [9/10], Step [200/200], Loss: 2.3022 for learning rate: 0.001 using SGD
Accuracy of the network on the test images: 10.7600 % for epoch: 9 using SGD
Epoch [10/10], Step [100/200], Loss: 2.3024 for learning rate: 0.001 using SGD
Accuracy of the network on the test images: 10.0200 % for epoch: 10 using SGD
Epoch [10/10], Step [200/200], Loss: 2.3025 for learning rate: 0.001 using SGD
Accuracy of the network on the test images: 10.2600 % for epoch: 10 using SGD
```

Graph in Tensorboard





We observe that for SGD with the learning rate of 0.001, the training losses decrease and training accuracy increase almost linearly. The final accuracy is around 16%. This is lower than the accuracy observed for ADAM with same learning rate.

For testing we see that the losses show an overall trend of increasing and the accuracy show an overall trend of decrease with each epoch. The final accuracy is around 10.3%. Here the testing accuracy is lesser than training accuracy showing an overfitted model. The accuracy is very much less than what was observed with Adam optimizer.

Hence we can see that for learning rate 0.001, for this model and dataset Adam optimizer is better than SGD.

---

3. ADAM with learning rate 0.01

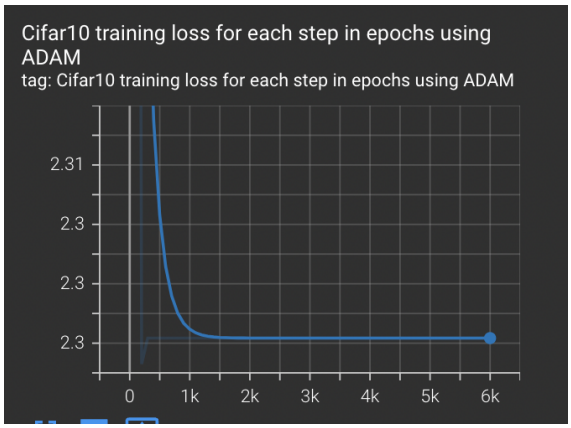
```
#Initializing the model as above
model = CNN_dropout(num_classes).to(device)

#Setting the loss function
cost = nn.CrossEntropyLoss()

#Setting the optimiser as Adam
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
```

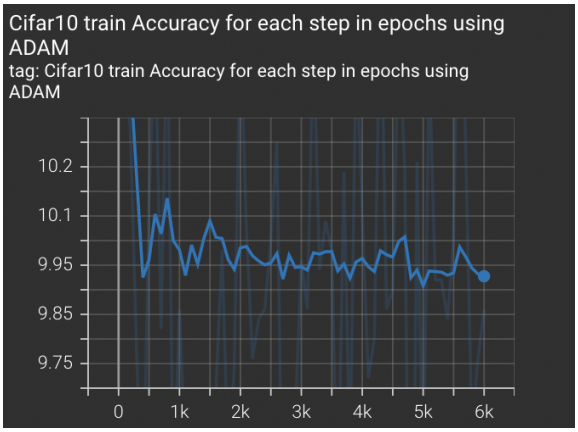
Training loss and accuracy using ADAM, learning rate - 0.01

Output

```
Epoch [9/10], Step [500/600], Loss: 2.3026 for learning rate: 0.01 using ADAM
Accuracy of the network on the train images: 9.9200 % for epoch: 9 using ADAM
Epoch [9/10], Step [600/600], Loss: 2.3026 for learning rate: 0.01 using ADAM
Accuracy of the network on the train images: 9.8400 % for epoch: 9 using ADAM
Epoch [10/10], Step [100/600], Loss: 2.3026 for learning rate: 0.01 using ADAM
Accuracy of the network on the train images: 10.0000 % for epoch: 10 using ADAM
Epoch [10/10], Step [200/600], Loss: 2.3026 for learning rate: 0.01 using ADAM
Accuracy of the network on the train images: 10.7400 % for epoch: 10 using ADAM
Epoch [10/10], Step [300/600], Loss: 2.3026 for learning rate: 0.01 using ADAM
Accuracy of the network on the train images: 9.6800 % for epoch: 10 using ADAM
Epoch [10/10], Step [400/600], Loss: 2.3026 for learning rate: 0.01 using ADAM
Accuracy of the network on the train images: 9.6200 % for epoch: 10 using ADAM
Epoch [10/10], Step [500/600], Loss: 2.3026 for learning rate: 0.01 using ADAM
Accuracy of the network on the train images: 9.7600 % for epoch: 10 using ADAM
Epoch [10/10], Step [600/600], Loss: 2.3026 for learning rate: 0.01 using ADAM
Accuracy of the network on the train images: 9.8600 % for epoch: 10 using ADAM
```

Graph in Tensorboard



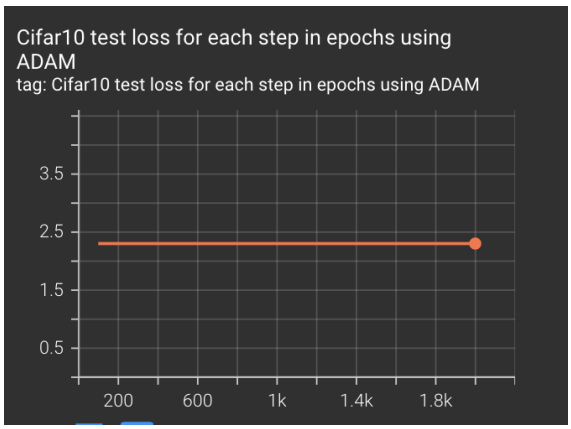


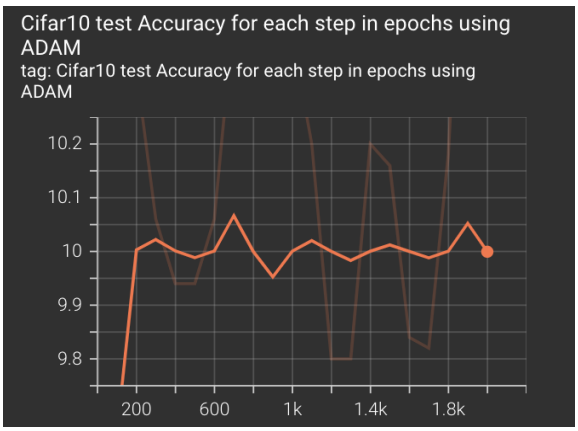
Testing loss and accuracy using ADAM, learning rate - 0.01

Output

```
Epoch [7/10], Step [100/200], Loss: 2.3026 for learning rate: 0.01 using ADAM
Accuracy of the network on the test images: 9.8000 % for epoch: 7 using ADAM
Epoch [7/10], Step [200/200], Loss: 2.3026 for learning rate: 0.01 using ADAM
Accuracy of the network on the test images: 10.2000 % for epoch: 7 using ADAM
Epoch [8/10], Step [100/200], Loss: 2.3026 for learning rate: 0.01 using ADAM
Accuracy of the network on the test images: 10.1600 % for epoch: 8 using ADAM
Epoch [8/10], Step [200/200], Loss: 2.3026 for learning rate: 0.01 using ADAM
Accuracy of the network on the test images: 9.8400 % for epoch: 8 using ADAM
Epoch [9/10], Step [100/200], Loss: 2.3026 for learning rate: 0.01 using ADAM
Accuracy of the network on the test images: 9.8200 % for epoch: 9 using ADAM
Epoch [9/10], Step [200/200], Loss: 2.3026 for learning rate: 0.01 using ADAM
Accuracy of the network on the test images: 10.1800 % for epoch: 9 using ADAM
Epoch [10/10], Step [100/200], Loss: 2.3026 for learning rate: 0.01 using ADAM
Accuracy of the network on the test images: 10.8600 % for epoch: 10 using ADAM
Epoch [10/10], Step [200/200], Loss: 2.3026 for learning rate: 0.01 using ADAM
Accuracy of the network on the test images: 9.1400 % for epoch: 10 using ADAM
```

Graph in Tensorboard





We observe that for Adam optimizer with lr 0.01, the training loss decreases initially and then remains constant while the training accuracy shows spikes with slightly decreasing trend with final accuracy around 9.97 %.

For testing the loss remains almost same while the accuracy initially increases and then remains same at around 10%. This shows that the model is not overfitted, but the accuracy is very much less than what was observed for lesser learning rate.
Hence as we increase learning rate, the Adam optimizer shows lesser accuracy for the same number of epochs and other parameters.

---

4. SGD with learning rate 0.01

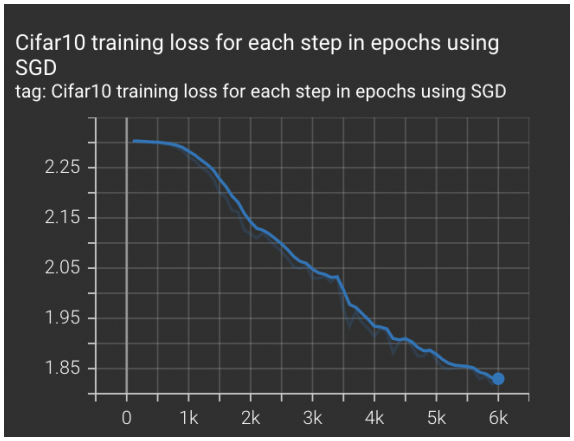
```
#Initializing the model as above
model = CNN_dropout(num_classes).to(device)

#Setting the loss function
cost = nn.CrossEntropyLoss()

#Setting the optimiser
optimizer = torch.optim.SGD(model.parameters(), lr=0.01)
learning_rate = 0.01
```

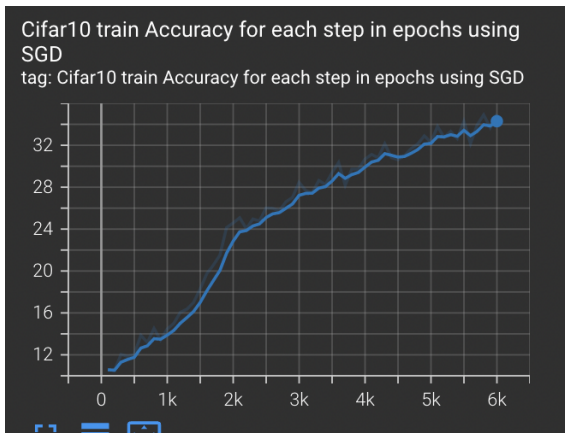
Training loss and accuracy using SGD, learning rate - 0.01

Output

```
Accuracy of the network on the train images: 32.7600 % for epoch: 9 using SGD
Epoch [9/10], Step [500/600], Loss: 1.8511 for learning rate: 0.01 using SGD
Accuracy of the network on the train images: 33.3200 % for epoch: 9 using SGD
Epoch [9/10], Step [600/600], Loss: 1.8536 for learning rate: 0.01 using SGD
Accuracy of the network on the train images: 32.6400 % for epoch: 9 using SGD
Epoch [10/10], Step [100/600], Loss: 1.8527 for learning rate: 0.01 using SGD
Accuracy of the network on the train images: 34.2800 % for epoch: 10 using SGD
Epoch [10/10], Step [200/600], Loss: 1.8470 for learning rate: 0.01 using SGD
Accuracy of the network on the train images: 32.1600 % for epoch: 10 using SGD
Epoch [10/10], Step [300/600], Loss: 1.8295 for learning rate: 0.01 using SGD
Accuracy of the network on the train images: 33.9200 % for epoch: 10 using SGD
Epoch [10/10], Step [400/600], Loss: 1.8344 for learning rate: 0.01 using SGD
Accuracy of the network on the train images: 34.9000 % for epoch: 10 using SGD
Epoch [10/10], Step [500/600], Loss: 1.8197 for learning rate: 0.01 using SGD
Accuracy of the network on the train images: 33.6800 % for epoch: 10 using SGD
Epoch [10/10], Step [600/600], Loss: 1.8272 for learning rate: 0.01 using SGD
Accuracy of the network on the train images: 34.9800 % for epoch: 10 using SGD
```

Graph in Tensorboard



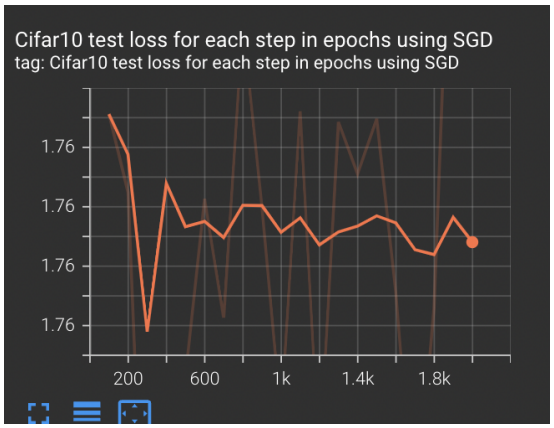


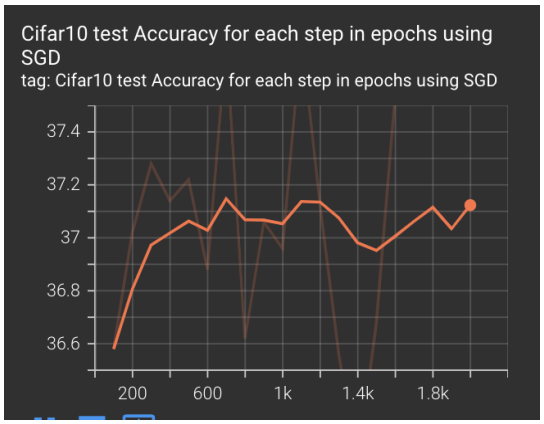
Testing loss and accuracy using SGD, learning rate - 0.01

Output

```
Accuracy of the network on the test images: 37.1200 % for epoch: 6 using SGD
Epoch [7/10], Step [100/200], Loss: 1.7649 for learning rate: 0.01 using SGD
Accuracy of the network on the test images: 36.5600 % for epoch: 7 using SGD
Epoch [7/10], Step [200/200], Loss: 1.7631 for learning rate: 0.01 using SGD
Accuracy of the network on the test images: 36.1400 % for epoch: 7 using SGD
Epoch [8/10], Step [100/200], Loss: 1.7650 for learning rate: 0.01 using SGD
Accuracy of the network on the test images: 36.6800 % for epoch: 8 using SGD
Epoch [8/10], Step [200/200], Loss: 1.7591 for learning rate: 0.01 using SGD
Accuracy of the network on the test images: 37.5400 % for epoch: 8 using SGD
Epoch [9/10], Step [100/200], Loss: 1.7513 for learning rate: 0.01 using SGD
Accuracy of the network on the test images: 37.6400 % for epoch: 9 using SGD
Epoch [9/10], Step [200/200], Loss: 1.7586 for learning rate: 0.01 using SGD
Accuracy of the network on the test images: 37.6800 % for epoch: 9 using SGD
Epoch [10/10], Step [100/200], Loss: 1.7755 for learning rate: 0.01 using SGD
Accuracy of the network on the test images: 36.1400 % for epoch: 10 using SGD
Epoch [10/10], Step [200/200], Loss: 1.7513 for learning rate: 0.01 using SGD
Accuracy of the network on the test images: 38.1400 % for epoch: 10 using SGD
```

Graph in Tensorboard





For SGD with learning rate 0.01, we can observe that the training losses decrease and training accuracy increase as expected. The final accuracy is around 33%.
The testing losses show an overall decreasing trend and the testing accuracy shows an increasing trend. The final testing accuracy is around 37.2% .

We can see that the model is not overfitted in this case and for learning rate 0.01 (higher) SGD performs better than ADAM. SGD with higher learning rate also performs better than SGD with lower learning rate (0.001). Thus we can say that for lower learning rates Adam is better while for higher learning rates SGD is better.

But overall we can say that the best performance is obtained without overfitting using Adam optimizer with learning rate 0.001, dropout layers and data augmentation on this model and dataset.

References:

- https://www.tensorflow.org/tensorboard/tensorboard_in_notebooks
- https://www.kaggle.com/code/taruntiwarihp/pytorch-cnns/notebook
- https://pytorch.org/tutorials/intermediate/tensorboard_tutorial.html
- https://www.tensorflow.org/api_docs/python/tf/summary/SummaryWriter
- https://medium.com/swlh/how-data-augmentation-improves-your-cnn-performance-an-experiment-in-pytorch-and-torchvision-e5fb36d038fb
- https://www.aiworkbox.com/lessons/normalize-cifar10-dataset-tensor
- https://wandb.ai/authors/ayusht/reports/Implementing-Dropout-in-PyTorch-With-Example--VmlldzoxNTgwOTE#:~:text=Add%20Dropout%20to%20a%20PyTorch,being%20deactivated%20%E2%80%93%20as%20a%20parameter.&text=We%20can%20apply%20dropout%20after%20any%20non%2Doutput%20layer.
- https://androidkt.com/how-to-add-l1-l2-regularization-in-pytorch-loss-function/
- https://github.com/Armour/pytorch-nn-practice/blob/master/utils/meanstd.py