Lab Course: Distributed Data Analytics
Exercise Sheet 7

Syed Wasif Murtaza Jafri- 311226

Network Analysis: Image Classification - Part 2

In the previous lab, we have implemented CNNs for an image classification task without paying much attention to the training behavior. Unchecked neural networks may or may not work based on their initialization. In this lab, we will try to exercise a fine-grain control over the parameters of the network. For this lab, you will implement the network following architecture and use the CIFAR dataset:

1. conv1: convolution and rectified linear activation (RELU)
2. pool1: max pooling
3. conv2: convolution and rectified linear activation (RELU)
4. conv3: convolution and rectified linear activation (RELU)
5. pool2: max pooling
6. FC1: fully connected layer with rectified linear activation (RELU)
7. FC2: fully connected layer with rectified linear activation (RELU)
8. FC3: fully connected layer with rectified linear activation (RELU)
9. softmax_layer: final output predictions i.e. classify into one of the ten classes.

NOTE: the kernel size, FC layer hidden units etc. is left up to you to design (bigger FC layers would lead to additional model complexity).
You will notice that the network is a little bit poorly designed and that is intentional. We would like the network to be more complex than is needed to overfit to our data. In addition to the increased complexity of the network, we will also reduce the amount of training data to 1/2 and also remove all data augmentations/normalizations.
With this network design and data constraint, you are required to get a baseline result to showcase the model behavior. Please **plot both the training and test accuracy and loss for all minibatches** (not just at the end of training). This can be achieved by interleaving a testing step after every 50-100 minibatch during model training.

In [5]:

```
import warnings
warnings.filterwarnings('ignore')
import torch
import torch.nn as nn
import torchvision.datasets as datasets  # Standard datasets
import torchvision.transforms as transforms  # Transformations we can perform on our dataset for augmentation
from torch import optim  # For optimizers like SGD, Adam, etc.
from torch import nn  # All neural network modules
from torch.utils.data import DataLoader,Dataset  # Gives easier dataset managment by creating mini batches etc.
import torch
from torch.utils.tensorboard import SummaryWriter
from torchvision import transforms
import matplotlib.pyplot as plt
torch.cuda.empty_cache()
import torch.nn.functional as F
import torchvision

import random
from timeit import default_timer as timer
```

In [54]:

```
def calculateMeanStd(cifar_trainset):
    images = [item[0] for item in cifar_trainset]
    images = torch.stack(images, dim=0).numpy()

    mean_r = images[:,0,:,:].mean()
    mean_g = images[:,1,:,:].mean()
    mean_b = images[:,2,:,:].mean()

    # calculate std over each channel (r,g,b)
    std_r = images[:,0,:,:].std()
    std_g = images[:,1,:,:].std()
    std_b = images[:,2,:,:].std()

    return [mean_r,mean_g,mean_b],[std_r,std_g,std_b]
```

In [8]:

```
def accuracy(outputs, labels):
    ''' Calculating accuray of prediction'''

    _, preds = torch.max(outputs, dim = 1) # class with highest probablity is the predicted output
    return(torch.tensor(torch.sum(preds == labels).item()/ len(preds)))
```

In [9]:

```
def trainModel(model,optimizer,criterion,learning_rate,train_loader,test_loader,num_epochs,desc):
    layout = {
    "Performance on training and test data": {
        "loss": ["Multiline", ["loss/train", "loss/test"]],
        "accuracy": ["Multiline", ["accuracy/train", "accuracy/test"]],
    },
    }
    writer = SummaryWriter(f'runs/{desc}/')
    writer.add_custom_scalars(layout)

    n_total_steps = len(train_loader)
    for epoch in range(num_epochs):
        lossSum = 0
        accSum = 0

        # iterating through all batches
        for i,sample_batched in enumerate(train_loader):

            # setting gradient to zero before forward pass
            optimizer.zero_grad()

            # model predict
            yhat=model(sample_batched[0].cuda())

            # model loss
            loss=criterion(yhat.squeeze().cuda(),sample_batched[1].squeeze().cuda())
            lossSum += loss.item()

            # model accuracy
            acc = accuracy(yhat.squeeze().cuda(),sample_batched[1].squeeze().cuda())

            accSum += acc.item()

            # calculating gradient through backprop
            loss.backward(retain_graph=True)

            # updating parameters
            optimizer.step()
                    # writing accuracy and loss to tensorboard logs
        lossAvg = lossSum /len(train_loader)
        accAvg = accSum/len(train_loader)
        writer.add_scalar("loss/train", lossAvg, epoch)
        writer.add_scalar("accuracy/train",accAvg, epoch)

        testAcc,testLoss = evalModel(model,optimizer,criterion,test_loader,epoch,writer)

        if (epoch+1) % 5 == 0:
            print (f'Epoch [{epoch+1}/{num_epochs}],Train Loss: {lossAvg:.4f},Train Accuracy: {accAvg:.4f},Test Loss: {testLoss:.4f},Test Accuracy: {testAcc:.4f}')

    writer.flush()
    print('Finished Training')
    return model
def evalModel(model,optimizer,criterion,test_loader,epoch,writer):
    accSum = 0
    lossSum = 0
    # for evaluating model on test data with no gradient calculation
    with torch.no_grad():
        for i,sample_batched in enumerate(test_loader):
            # setting gradient to zero before forward pass
            optimizer.zero_grad()

            # model predict
            yhat=model(sample_batched[0].cuda())

            # model loss
            loss=criterion(yhat.squeeze().cuda(),sample_batched[1].squeeze().cuda())
            lossSum += loss
            # model acc
            acc = accuracy(yhat.squeeze().cuda(),sample_batched[1].squeeze().cuda())
            accSum += acc

            # writing accuracy and loss to tensorboard logs

    totalloss = lossSum/len(test_loader)
    totalAccuracy = accSum/len(test_loader)
    if(writer!=None):
        writer.add_scalar("loss/test", totalloss, epoch)
        writer.add_scalar("accuracy/test",totalAccuracy, epoch)


    return totalAccuracy.item(),totalloss.item()
```

In [12]:

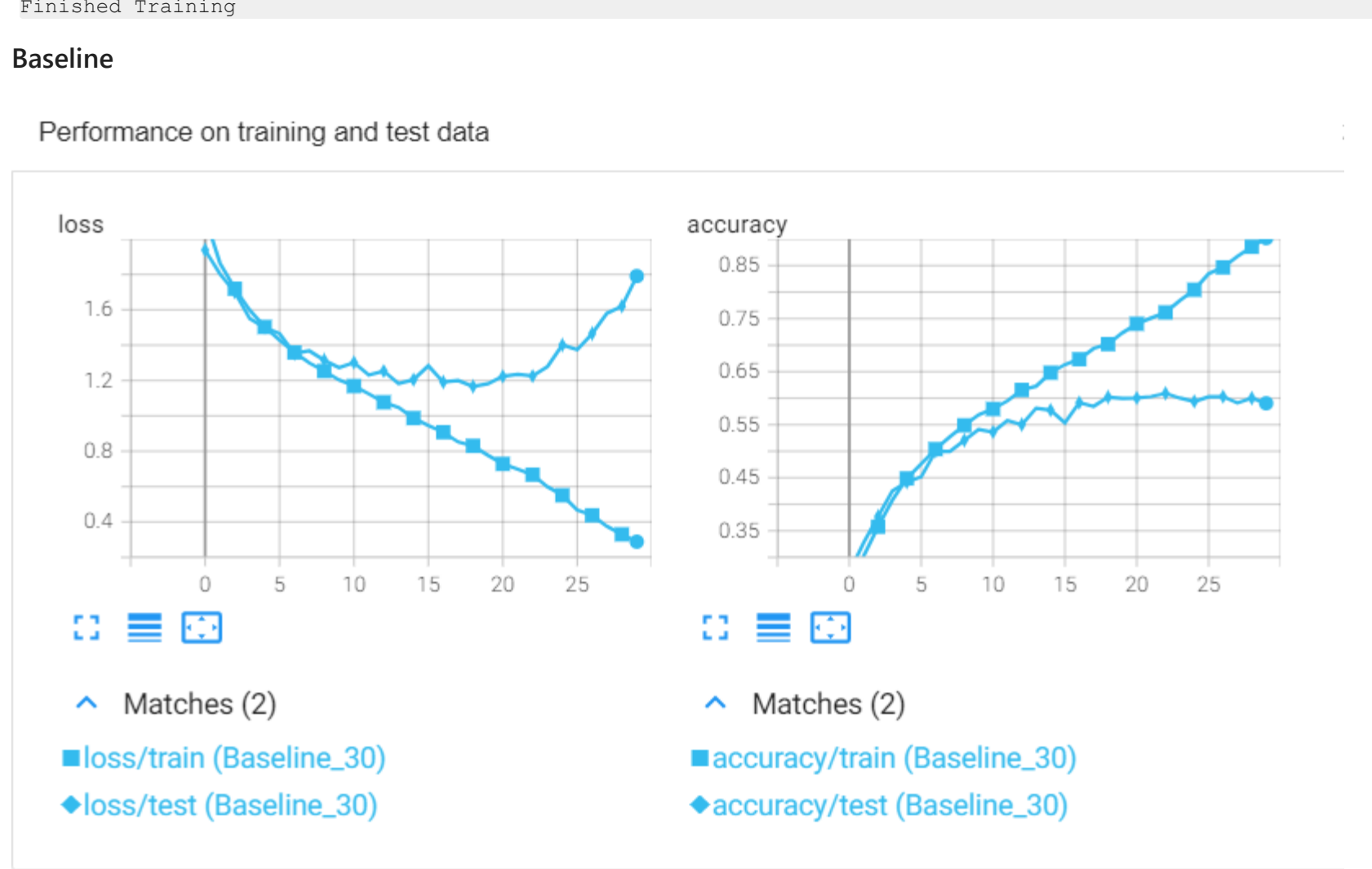
```
class CNN(nn.Module):
    def __init__(self, input_channels=1, num_of_classes=10):
        super(CNN, self).__init__()
        self.relu = nn.ReLU()
        self.softmax = nn.LogSoftmax()

        self.conv1 = nn.Conv2d(
            in_channels = input_channels,
            out_channels = 16,
            kernel_size = (5,5),
            stride=(1,1),
            padding=(0,0)

        )
        self.pool1 = nn.MaxPool2d(kernel_size=(2,2),stride=(2,2))
        self.conv2 = nn.Conv2d(
            in_channels = 16,
            out_channels = 32,
            kernel_size = (5,5),
            stride=(1,1),
            padding=(0,0)
        )
        self.conv3 = nn.Conv2d(
            in_channels = 32,
            out_channels = 64,
            kernel_size = (5,5),
            stride=(1,1),
            padding=(0,0)
        )
        self.pool2 = nn.MaxPool2d(kernel_size=(2,2),stride=(2,2))
        self.FC1 = nn.Linear(64*3*3,1024)
        self.FC2 = nn.Linear(1024,512)
        self.FC3= nn.Linear(512,256)
        self.Output = nn.Linear(256,10)


    def forward(self,x):

        x = self.relu(self.conv1(x))
        x = self.pool1(x)
        x = self.relu(self.conv2(x))
        x = self.relu(self.conv3(x))
        x = self.pool1(x)

        x = x.view(-1, 64 *3* 3)

        x = self.relu(self.FC1(x))
        x = self.relu(self.FC2(x))
        x = self.relu(self.FC3(x))

        x = self.softmax(self.Output(x))

        return x
```

In [14]:

```
torch.cuda.empty_cache()

# setting device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(device)

in_channels = 3 # input channels of image
num_classes = 10 # number of output class [0-10]
num_epochs = 30
batch_size= 512

transform = transforms.Compose([transforms.Resize((32,32)),
    transforms.ToTensor()
])

# getting training and test data
train_dataset = datasets.CIFAR10(root="dataset/", train=True, transform=transform,download=True)
# spliting dataset into half with random indexes
indexList = random.sample(list(range(0,len(train_dataset))), (len(train_dataset)//2))
train_dataset_half = torch.utils.data.Subset(train_dataset,indexList)

test_dataset = datasets.CIFAR10(root="dataset/", train=False, transform=transform,download=True)
train_loader = DataLoader(dataset=train_dataset_half,batch_size=batch_size, shuffle=True)
test_loader = DataLoader(dataset=test_dataset, batch_size=batch_size ,shuffle=True)

# different setting of learning rates
learning_rate =0.001

criterion = nn.NLLLoss()

# Initialize network
model = CNN(input_channels=in_channels, num_of_classes=num_classes).to(device)

# Initializing Optimizer
optimizer= torch.optim.Adam(model.parameters(),lr=learning_rate)

# Initializing writer for tensorboard
desc = 'Baseline'+'_'+str(num_epochs)

print('-------------------------------------------------')
print('|-------------Baseline Experiment --------------|')
print('-------------------------------------------------')
model = trainModel(model, optimizer,criterion, learning_rate, train_loader,test_loader,
num_epochs, desc)
```

```
cuda
Files already downloaded and verified
Files already downloaded and verified
-------------------------------------------------
|-------------Baseline Experiment --------------|
-------------------------------------------------
Epoch [5/30],Train Loss: 1.5031,Train Accuracy: 0.4488,Test Loss: 1.5002,Test Accuracy: 0.4415
Epoch [10/30],Train Loss: 1.2029,Train Accuracy: 0.5685,Test Loss: 1.2735,Test Accuracy: 0.5408
Epoch [15/30],Train Loss: 0.9874,Train Accuracy: 0.6480,Test Loss: 1.2046,Test Accuracy: 0.5776
Epoch [20/30],Train Loss: 0.7771,Train Accuracy: 0.7231,Test Loss: 1.1815,Test Accuracy: 0.5998
Epoch [25/30],Train Loss: 0.5510,Train Accuracy: 0.8044,Test Loss: 1.4008,Test Accuracy: 0.5939
Epoch [30/30],Train Loss: 0.2882,Train Accuracy: 0.9018,Test Loss: 1.7907,Test Accuracy: 0.5904
Finished Training
```

Baseline



Comment : Model is overfitting training data after 15 epochs as test loss is increasing and test accuracy becomes constant and training accuracy increases.

Exercise 1: Normalization Effect

Now that we have a weak baseline, we can start to investigate how to improve the model performance. We will first try to address the data aspect of model improvement.
Improving data:

1. Data Augmentation: the process of artificially 'increasing' our dataset by adding translation, scaling and flipping to the images to fabricate examples for training.
2. Normalization: Normalizing the input data helps remove the dataset artifacts that can cause poor model performance.

In this task, you are required to add data augmentation and normalization to the dataset and run the training. Compare the training performance (accuracy and loss) to the baseline first with only data augmentation, secondly with only normalization, and lastly with both. **Comment on the difference you observe in the training behavior and the final accuracy as seen on the test set.** NOTE: You should use tensorboard plots.

In [51]:

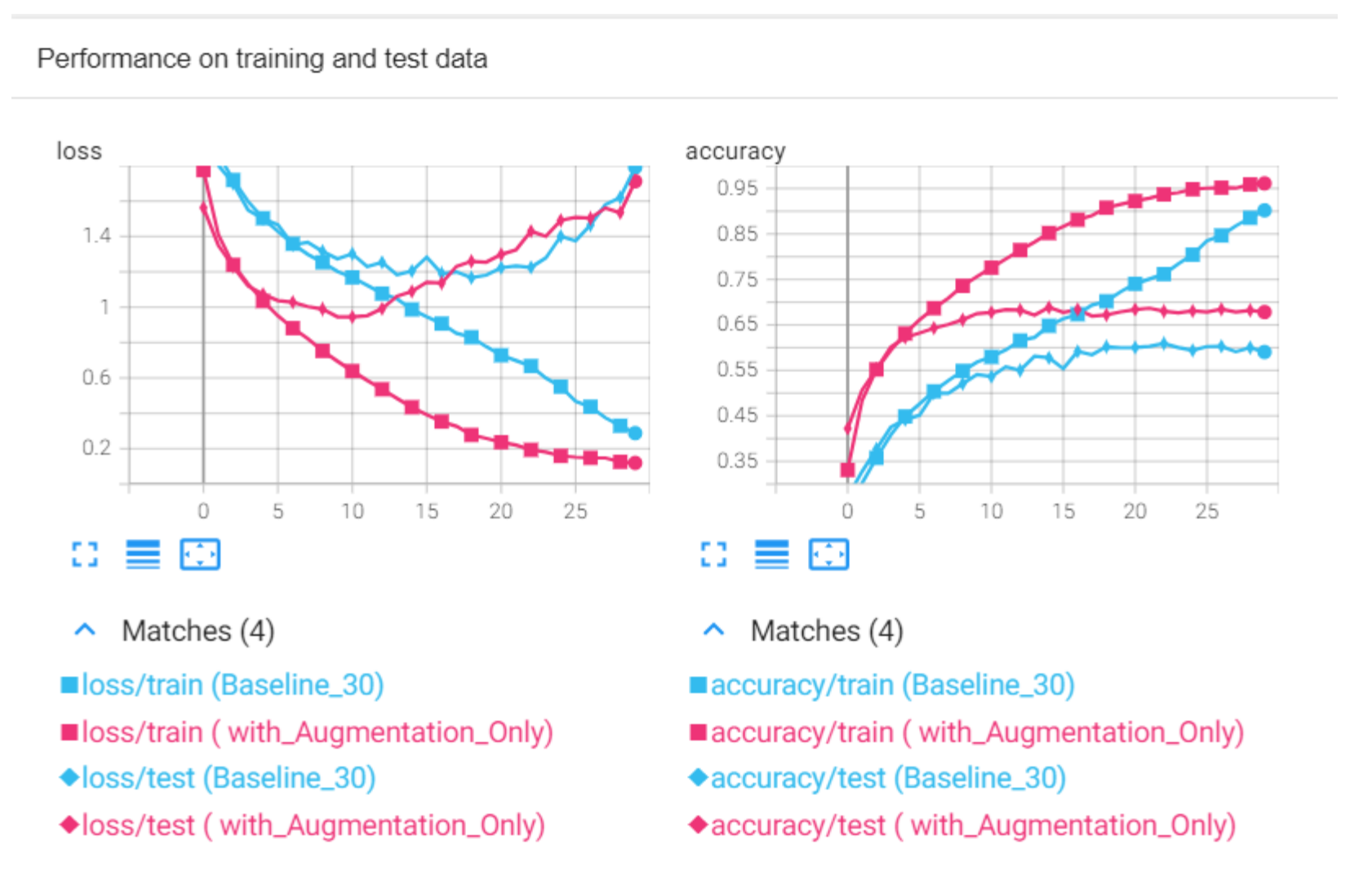
```
def getTrainData(augm,norm):
    ## calculating mean and std for normalization
    transform = transforms.Compose([
        transforms.ToTensor()
    ])
    train_data = datasets.CIFAR10(root="dataset/",
train=True,transform=transform,download=False)
    meanPerChannel,stdPerChannel =calculateMeanStd2(train_data)

    if augm == True and norm==False:
        transform = transforms.Compose([transforms.Resize((32,32)),
            transforms.ToTensor(),
        ])
        train_dataset_org = datasets.CIFAR10(root="dataset/", train=True,
transform=transform,download=False)
        transform = transforms.Compose([transforms.Resize((32,32)),
            transforms.ToTensor(),
            transforms.ColorJitter(hue=.05, saturation=.05),
            transforms.RandomHorizontalFlip(),
        ])
        train_dataset_aug = datasets.CIFAR10(root="dataset/", train=True,
transform=transform,download=False)
        train_dataset = torch.utils.data.ConcatDataset([train_dataset_org, train_dataset_aug])
        desc = ' with_Augmentation_Only'

    elif augm == False and norm==True:
        transform = transforms.Compose([transforms.Resize((32,32)),
        transforms.ToTensor(),
        transforms.Normalize(mean=  meanPerChannel, std=stdPerChannel)
    ])
        train_dataset = datasets.CIFAR10(root="dataset/", train=True,
transform=transform,download=False)
        meanPerChannel2,stdPerChannel2 =calculateMeanStd(train_dataset)
        desc = ' with_Normalization_Only'


    elif augm == True and norm==True:
        transform = transforms.Compose([transforms.Resize((32,32)),
            transforms.ToTensor(),
            transforms.Normalize(mean=  meanPerChannel, std=stdPerChannel)
        ])
        train_dataset_org = datasets.CIFAR10(root="dataset/", train=True,
transform=transform,download=False)
        transform = transforms.Compose([transforms.Resize((32,32)),
            transforms.ToTensor(),
            transforms.ColorJitter(hue=.05, saturation=.05),
            transforms.RandomHorizontalFlip(),
            transforms.Normalize(mean=  meanPerChannel, std=stdPerChannel)
        ])
        train_dataset_aug = datasets.CIFAR10(root="dataset/", train=True,
transform=transform,download=False)
        train_dataset = torch.utils.data.ConcatDataset([train_dataset_org, train_dataset_aug])
        desc = ' with_both_Augmentation_and_Normalization'
    return train_dataset,desc,meanPerChannel,stdPerChannel,transform
```

In [53]:

```
# setting device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

in_channels = 3 # input channels of image
num_classes = 10 # number of output class [0-10]
num_epochs = 30
batch_size= 512
config_aug_norm =[(True,False),(False,True),(True,True)]

for augm,norm in config_aug_norm:

    # getting training and test data
    train_dataset,desc,meanPerChannel,stdPerChannel,transform = getTrainData(augm,norm)
    # using same tranform for test
    test_dataset = datasets.CIFAR10(root="dataset/", train=False,
transform=transform,download=False)
    train_loader = DataLoader(dataset=train_dataset,batch_size=batch_size, shuffle=True)
    test_loader = DataLoader(dataset=test_dataset, batch_size=batch_size ,shuffle=True)

    # different setting of learning rates
    learning_rate = 0.001
    n_total_steps = len(train_loader)
    criterion = nn.NLLLoss()

    # Initialize network
    model = CNN(input_channels=in_channels, num_of_classes=num_classes).to(device)

    # Initializing Optimizer
    optimizer= torch.optim.Adam(model.parameters(),lr=learning_rate)

    # Initializing writer for tensorboard

    print('-------------------------------------------------')
    print('|------Experiment'+str(desc)+'--------|')
    print('-------------------------------------------------')

    model = trainModel(model, optimizer,criterion, learning_rate, train_loader,test_loader,
num_epochs, desc)
```

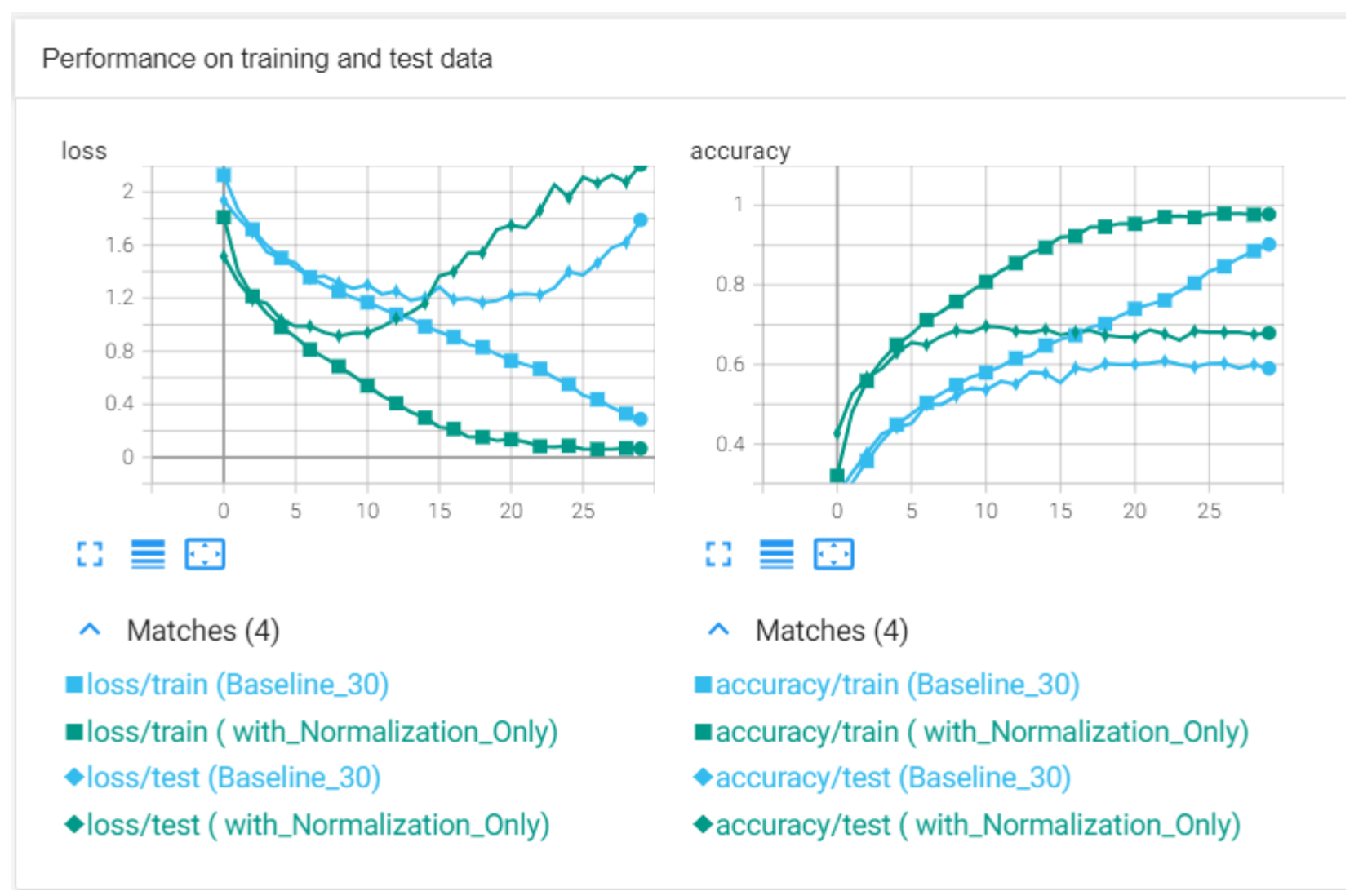
```
-------------------------------------------------
|------Experiment with_Augmentation_Only--------|
-------------------------------------------------
Epoch [5/30],Train Loss: 1.0352,Train Accuracy: 0.6310,Test Loss: 1.0736,Test Accuracy: 0.6222
Epoch [10/30],Train Loss: 0.6948,Train Accuracy: 0.7563,Test Loss: 0.9453,Test Accuracy: 0.6748
Epoch [15/30],Train Loss: 0.4347,Train Accuracy: 0.8520,Test Loss: 1.0892,Test Accuracy: 0.6884
Epoch [20/30],Train Loss: 0.2583,Train Accuracy: 0.9152,Test Loss: 1.2546,Test Accuracy: 0.6787
Epoch [25/30],Train Loss: 0.1595,Train Accuracy: 0.9483,Test Loss: 1.4912,Test Accuracy: 0.6806
Epoch [30/30],Train Loss: 0.1197,Train Accuracy: 0.9611,Test Loss: 1.7120,Test Accuracy: 0.6782
Finished Training
-------------------------------------------------
|------Experiment with_Normalization_Only--------|
-------------------------------------------------
Epoch [5/30],Train Loss: 0.9828,Train Accuracy: 0.6494,Test Loss: 1.0373,Test Accuracy: 0.6287
Epoch [10/30],Train Loss: 0.6156,Train Accuracy: 0.7845,Test Loss: 0.9374,Test Accuracy: 0.6813
Epoch [15/30],Train Loss: 0.2989,Train Accuracy: 0.8943,Test Loss: 1.1615,Test Accuracy: 0.6884
Epoch [20/30],Train Loss: 0.1263,Train Accuracy: 0.9544,Test Loss: 1.7174,Test Accuracy: 0.6692
Epoch [25/30],Train Loss: 0.0871,Train Accuracy: 0.9700,Test Loss: 1.9585,Test Accuracy: 0.6843
Epoch [30/30],Train Loss: 0.0663,Train Accuracy: 0.9778,Test Loss: 2.2082,Test Accuracy: 0.6792
Finished Training
-------------------------------------------------
|------Experiment with_both_Augmentation_and_Normalization--------|
-------------------------------------------------
Epoch [5/30],Train Loss: 0.7666,Train Accuracy: 0.7293,Test Loss: 0.9306,Test Accuracy: 0.6826
Epoch [10/30],Train Loss: 0.3891,Train Accuracy: 0.8669,Test Loss: 0.9237,Test Accuracy: 0.7151
Epoch [15/30],Train Loss: 0.1994,Train Accuracy: 0.9346,Test Loss: 1.1733,Test Accuracy: 0.7162
Epoch [20/30],Train Loss: 0.1218,Train Accuracy: 0.9608,Test Loss: 1.3723,Test Accuracy: 0.7163
Epoch [25/30],Train Loss: 0.0879,Train Accuracy: 0.9712,Test Loss: 1.4858,Test Accuracy: 0.7129
Epoch [30/30],Train Loss: 0.0822,Train Accuracy: 0.9718,Test Loss: 1.6161,Test Accuracy: 0.7214
Finished Training
```

With Augmentation Only



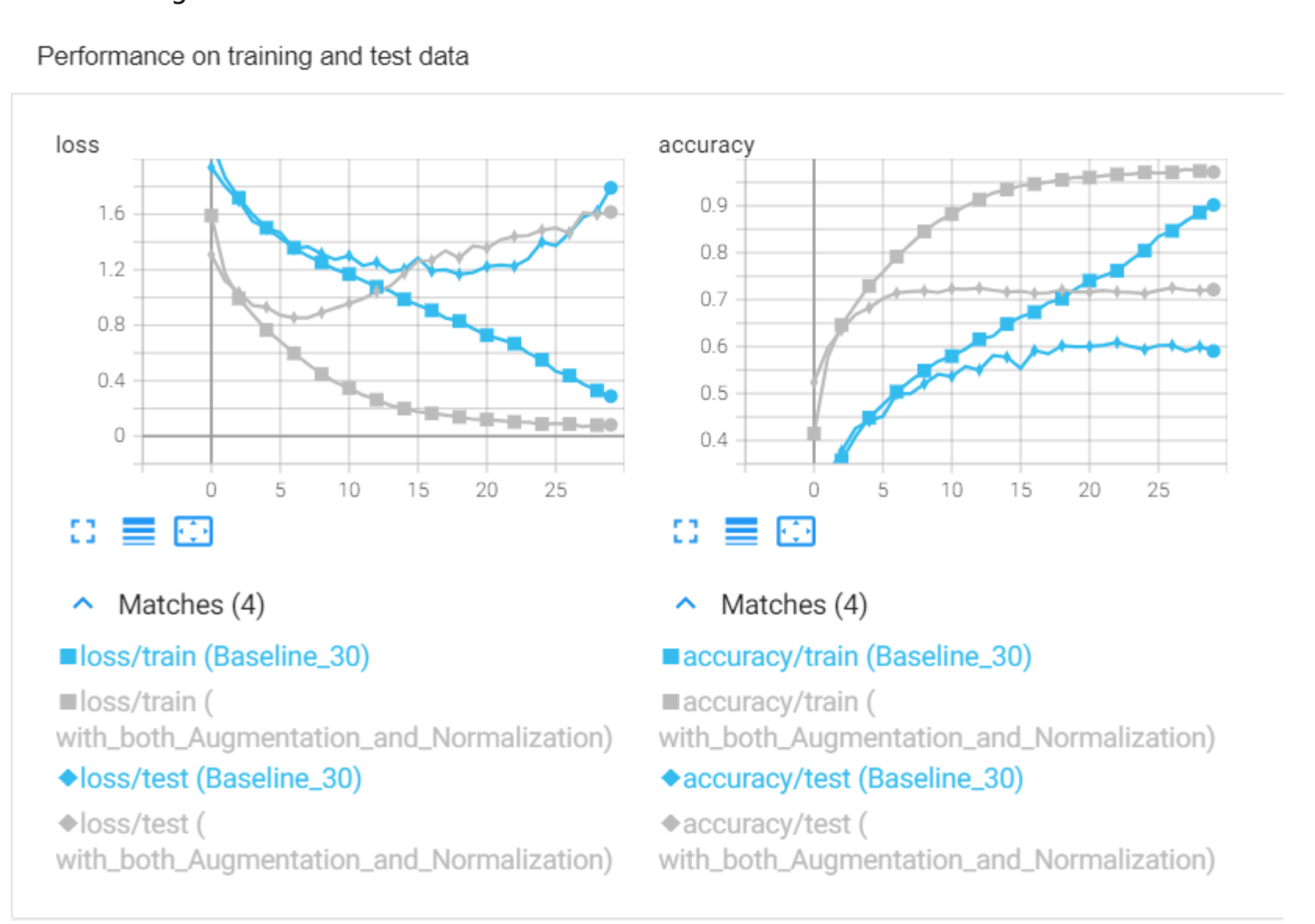
Comment : With data augmentation, more data is available for training and overfitting is reduced. also test accuracy is increased in comparison with baseline.

With Normalization Only



Comment : With normalization model is trained quickly in just around 10 epochs, in later epochs it is overfitting but overall test accuracy is better than baseline.

With both Augmentation and Normalization



Comment : When both normalization and augmentation of data is combined it gives the best result.Model is trained quickly also reduces overfitting resulting in better test accuracy than baseline.

Exercise 2: Network Regularization

Having tried to fix the dataset problems, we can have a look at the network itself. Regularization techniques 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. Here we will look at the dropout technique.

- Dropout: 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.

Similar to the first task, we are interested in seeing how the addition of network regularization can improve model training behavior and overall performance. You are required to add a dropout layer and report the training and testing loss and accuracy similar to the exercise above. **Comment on the impact of this modification when compared to the baseline approach.**

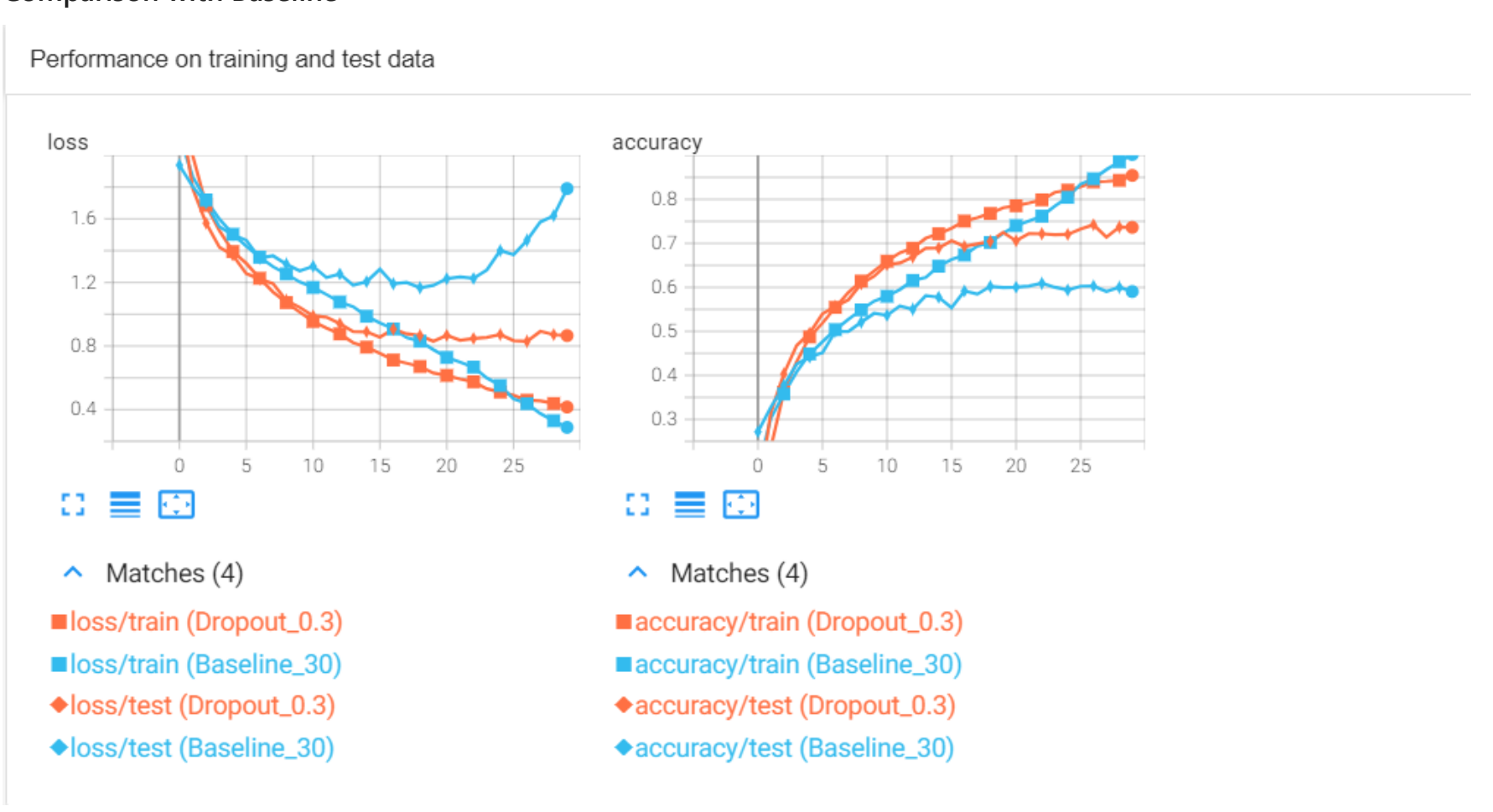
In [27]:

```
class CNNwithDropout(nn.Module):
    def __init__(self, input_channels=1, num_of_classes=10,dropout=0.5):
        super(CNNwithDropout, self).__init__()
        self.relu = nn.ReLU()
        self.softmax = nn.LogSoftmax()
        self.dropout = dropout

        self.conv1 = nn.Conv2d(
            in_channels = input_channels,
            out_channels = 64,
            kernel_size = (5,5),
            stride=(1,1),
            padding=(0,0)

        )
        self.pool1 = nn.MaxPool2d(kernel_size=(2,2),stride=(2,2))
        self.conv2 = nn.Conv2d(
            in_channels = 64,
            out_channels = 128,
            kernel_size = (5,5),
            stride=(1,1),
            padding=(0,0)
        )
        self.conv3 = nn.Conv2d(
            in_channels = 128,
            out_channels = 256,
            kernel_size = (5,5),
            stride=(1,1),
            padding=(0,0)
        )
        self.pool2 = nn.MaxPool2d(kernel_size=(2,2),stride=(2,2))
        self.dropout1 = nn.Dropout(self.dropout)
        self.dropout2 = nn.Dropout(self.dropout-0.1)
        self.dropout3 = nn.Dropout(self.dropout-0.2)
        self.FC1 = nn.Linear(256*3*3,512)
        self.FC2 = nn.Linear(512,256)
        self.FC3= nn.Linear(256,84)

    def forward(self,x):

        x = self.relu(self.conv1(x)) # dimension after : [1, 6, 28, 28]
        x = self.pool1(x)  # dimension after : [1, 6, 14, 14]
        x = self.relu(self.conv2(x))
        x = self.relu(self.conv3(x))# dimension after : [1, 16,10, 10]
        x = self.pool1(x)
        #print(x.shape)
        # flattening dimension (120,1,1) to 120
        x = x.view(-1, 256 *3* 3)   # dimension after : [1, 120]

        x = self.dropout1(x)
        x = self.relu(self.FC1(x))
        x = self.dropout2(x)
        x = self.relu(self.FC2(x))
        x = self.dropout3(x)
        x = self.relu(self.FC3(x))# dimension after : [1, 84]
        x = self.softmax(x)   # dimension after : [1, 10]


        return x
```

```python
# setting device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

in_channels = 3 # input channels of image
num_classes = 10 # number of output class [0-10]
num_epochs = 30
batch_size= 512

transform = transforms.Compose([transforms.Resize((32,32)),
    transforms.ToTensor()
])

# getting training and test data
train_dataset = datasets.CIFAR10(root="dataset/", train=True, transform=transform,download=True)
test_dataset = datasets.CIFAR10(root="dataset/", train=False, transform=transform,download=True)
train_loader = DataLoader(dataset=train_dataset,batch_size=batch_size, shuffle=True)
test_loader = DataLoader(dataset=test_dataset, batch_size=batch_size ,shuffle=True)

# different setting of learning rates
learning_rate =0.001
n_total_steps = len(train_loader)
criterion = nn.NLLLoss()
dropoutList = [0.3,0.4,0.5]

for dropout in dropoutList:
    desc = 'Dropout_'+str(dropout)
    # Initialize network
    model = CNNwithDropout(input_channels=in_channels, num_of_classes=num_classes,dropout =
dropout).to(device)

    # Initializing Optimizer
    optimizer= torch.optim.Adam(model.parameters(),lr=learning_rate)



    print('--------------------------------------------------')
    print('|------ Experiment with Dropout = '+str(dropout)+'--------|')
    print('--------------------------------------------------')



    model = trainModel(model, optimizer,criterion, learning_rate, train_loader,test_loader,
num_epochs, desc)

    writer = None
    epoch = None
    totalAccuracy = evalModel(model,optimizer,criterion,test_loader,epoch,writer)
    print('Final test Accuracy'+desc+':'+str(totalAccuracy))
```

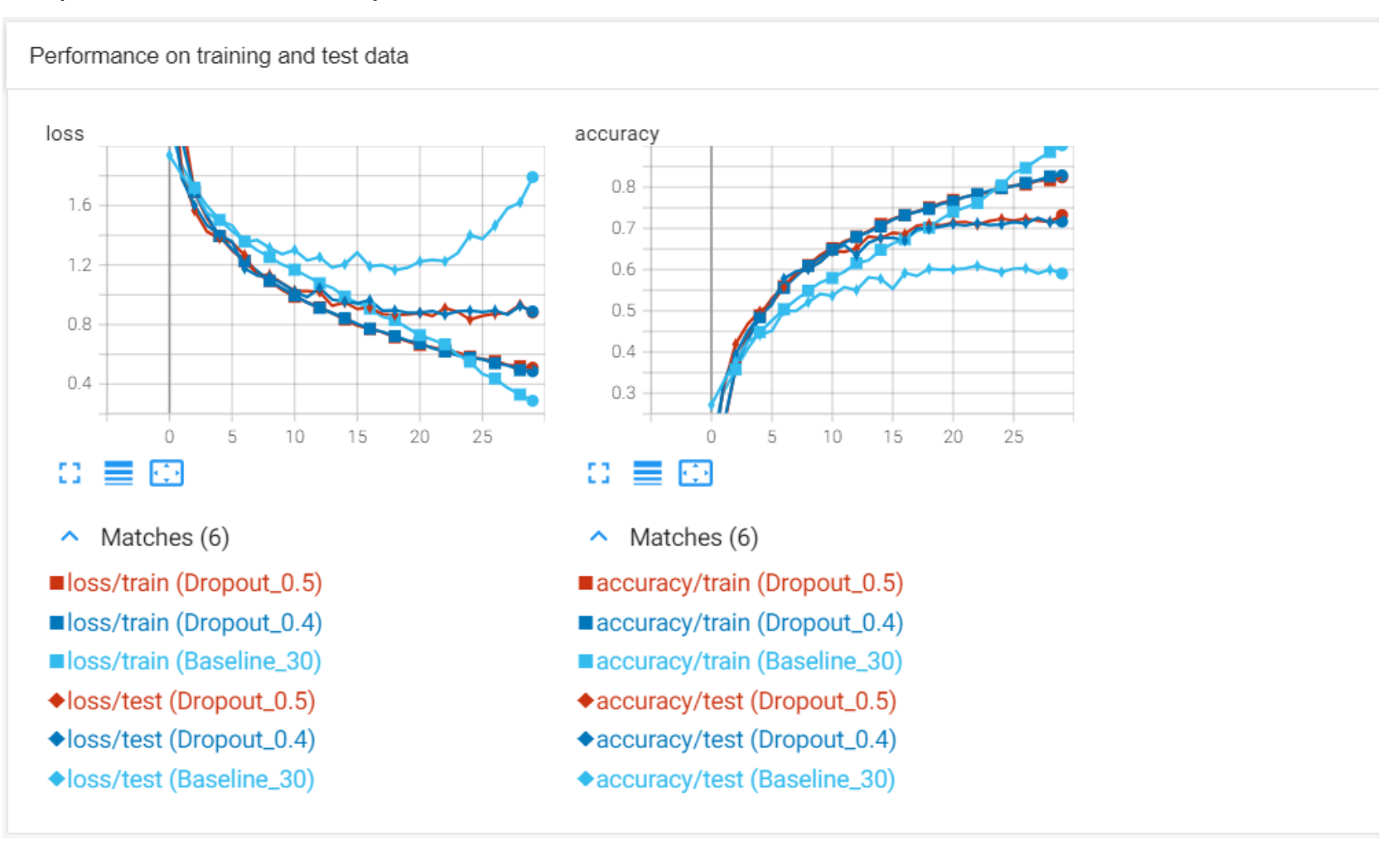
```
Files already downloaded and verified
Files already downloaded and verified
--------------------------------------------------
|------ Experiment with Dropout = 0.3--------|
--------------------------------------------------
Epoch [5/30],Train Loss: 1.3943,Train Accuracy: 0.4873,Test Loss: 1.3714,Test Accuracy: 0.4958
Epoch [10/30],Train Loss: 1.0138,Train Accuracy: 0.6371,Test Loss: 1.0436,Test Accuracy: 0.6252
Epoch [15/30],Train Loss: 0.7931,Train Accuracy: 0.7217,Test Loss: 0.8894,Test Accuracy: 0.6893
Epoch [20/30],Train Loss: 0.6300,Train Accuracy: 0.7808,Test Loss: 0.8298,Test Accuracy: 0.7244
Epoch [25/30],Train Loss: 0.5106,Train Accuracy: 0.8209,Test Loss: 0.8725,Test Accuracy: 0.7202
Epoch [30/30],Train Loss: 0.4161,Train Accuracy: 0.8544,Test Loss: 0.8657,Test Accuracy: 0.7363
Finished Training
Final test AccuracyDropout_0.3:(0.7333179712295532, 0.8759010434150696)
--------------------------------------------------
|------ Experiment with Dropout = 0.4--------|
--------------------------------------------------
Epoch [5/30],Train Loss: 1.3949,Train Accuracy: 0.4842,Test Loss: 1.4028,Test Accuracy: 0.4873
Epoch [10/30],Train Loss: 1.0451,Train Accuracy: 0.6289,Test Loss: 1.0760,Test Accuracy: 0.6178
Epoch [15/30],Train Loss: 0.8428,Train Accuracy: 0.7047,Test Loss: 0.9534,Test Accuracy: 0.6766
Epoch [20/30],Train Loss: 0.6934,Train Accuracy: 0.7594,Test Loss: 0.8801,Test Accuracy: 0.7043
Epoch [25/30],Train Loss: 0.5762,Train Accuracy: 0.7981,Test Loss: 0.8935,Test Accuracy: 0.7095
Epoch [30/30],Train Loss: 0.4862,Train Accuracy: 0.8287,Test Loss: 0.8882,Test Accuracy: 0.7166
Finished Training
Final test AccuracyDropout_0.4:(0.718422532081604, 0.8820830583572388)
--------------------------------------------------
|------ Experiment with Dropout = 0.5--------|
--------------------------------------------------
Epoch [5/30],Train Loss: 1.3949,Train Accuracy: 0.4892,Test Loss: 1.3792,Test Accuracy: 0.4988
Epoch [10/30],Train Loss: 1.0337,Train Accuracy: 0.6351,Test Loss: 1.0771,Test Accuracy: 0.6195
Epoch [15/30],Train Loss: 0.8339,Train Accuracy: 0.7107,Test Loss: 0.9489,Test Accuracy: 0.6761
Epoch [20/30],Train Loss: 0.6882,Train Accuracy: 0.7616,Test Loss: 0.8671,Test Accuracy: 0.7073
Epoch [25/30],Train Loss: 0.5853,Train Accuracy: 0.7974,Test Loss: 0.8336,Test Accuracy: 0.7233
Epoch [30/30],Train Loss: 0.5097,Train Accuracy: 0.8237,Test Loss: 0.8815,Test Accuracy: 0.7324
Finished Training
Final test AccuracyDropout_0.5:(0.7274242043495178, 0.8679054379463196)
```

Comparison with Baseline



Comment : When comparing with baseline, test accuracy increases after adding dropout because it regularizes the neurons in layers and model does not overfit the training data, which was the the case in baseline after 15 epochs as test loss is increasing.

Comparison with different dropouts



Comment : When experimenting with different dropout rates, test accuracy is best for dropout=0.5.

Bonus

Perform the same analysis as for the dropout technique above, also on the following regularization techniques:

- L1 regularization: Penalizing the weights of the network to encourage sparse weights. (1.5 points)
- L2 regularization: Penalizing the weights of the network to encourage them to fit on the unit circle. (1.5 points)

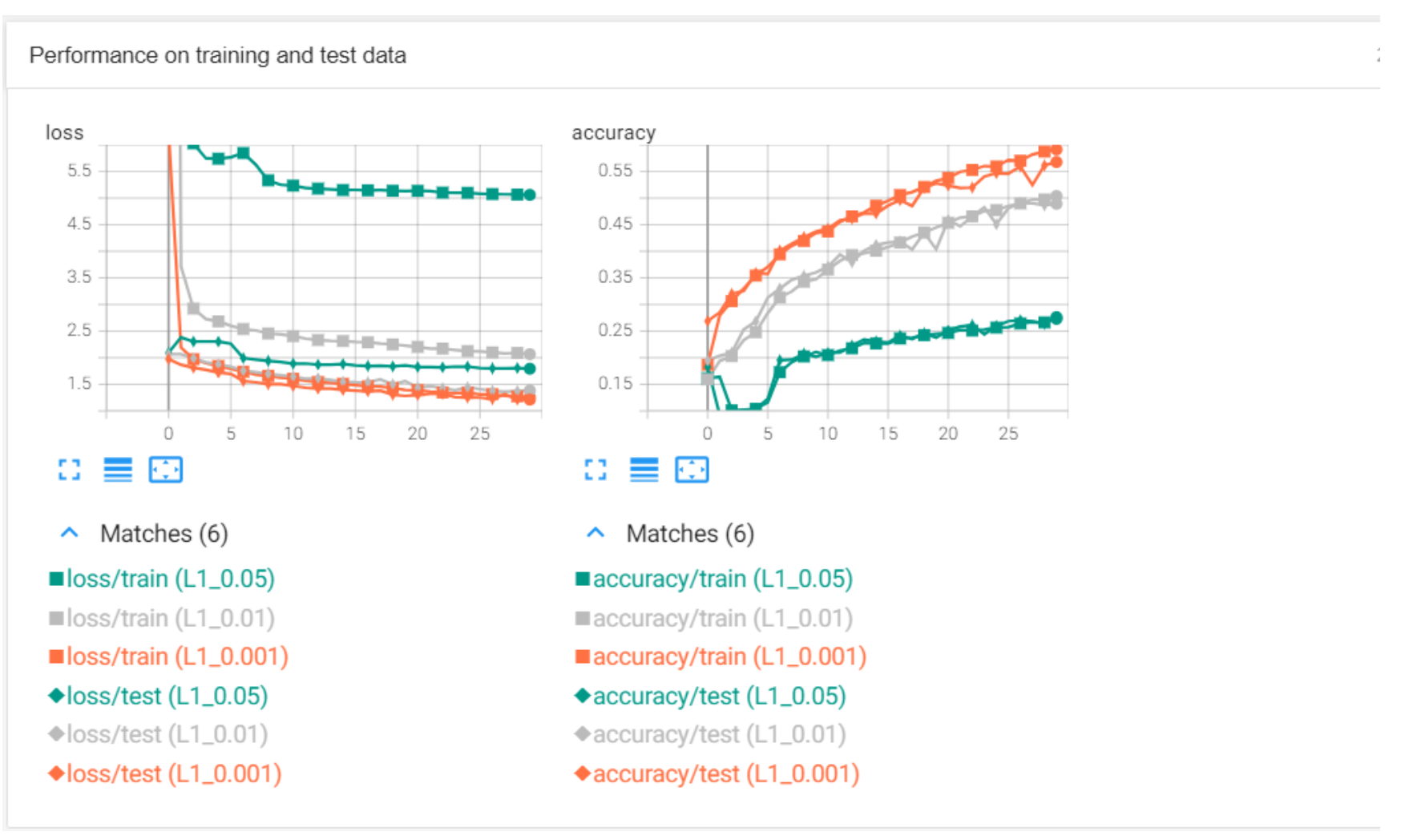
To get any bonus points here you will need to plot and comment on the training and testing loss as well as the accuracy for both the regularization techniques (L1 and L2).

```python
def
trainModelRegularizer(model,optimizer,criterion,learning_rate,train_loader,test_loader,num_epochs,

    layout = {
    "Performance on training and test data": {
        "loss": ["Multiline", ["loss/train", "loss/test"]],
        "accuracy": ["Multiline", ["accuracy/train", "accuracy/test"]],
    },
    }
    writer = SummaryWriter(f'runs/{desc}/')
    writer.add_custom_scalars(layout)

    n_total_steps = len(train_loader)
    for epoch in range(num_epochs):
        lossSum = 0
        accSum = 0

        # iterating through all batches
        for i,sample_batched in enumerate(train_loader):

            # setting gradient to zero before forward pass
            optimizer.zero_grad()

            # model predict
            yhat=model(sample_batched[0].cuda())

            # model loss
            nLLLoss=criterion(yhat.squeeze().cuda(),sample_batched[1].squeeze().cuda())

            all_linear1_params = torch.cat([x.view(-1) for x in model.FC1.parameters()])
            l1_regularization = lambda1 * torch.norm(all_linear1_params, regularizer)
            #l1_norm = sum(p.abs().sum() for p in model.parameters())

            loss = nLLLoss + l1_regularization
            lossSum += loss.item()

            # model accuracy
            acc = accuracy(yhat.squeeze().cuda(),sample_batched[1].squeeze().cuda())

            accSum += acc.item()

            # calculating gradient through backprop
            loss.backward(retain_graph=True)

            # updating parameters
            optimizer.step()
                    # writing accuracy and loss to tensorboard logs
        lossAvg = lossSum /len(train_loader)
        accAvg = accSum/len(train_loader)
        writer.add_scalar("loss/train", lossAvg, epoch)
        writer.add_scalar("accuracy/train",accAvg, epoch)

        testAcc,testLoss = evalModel(model,optimizer,criterion,test_loader,epoch,writer)

        if (epoch+1) % 5 == 0:
            print (f'Epoch [{epoch+1}/{num_epochs}],Train Loss: {lossAvg:.4f},Train Accuracy:
{accAvg:.4f},Test Loss: {testLoss:.4f},Test Accuracy: {testAcc:.4f}')

    writer.flush()
    print('Finished Training')
    return model
```

```python
# setting device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(device)

in_channels = 3 # input channels of image
num_classes = 10 # number of output class [0-10]
num_epochs = 30
batch_size= 512

transform = transforms.Compose([transforms.Resize((32,32)),
    transforms.ToTensor()
])

# getting training and test data
train_dataset = datasets.CIFAR10(root="dataset/", train=True, transform=transform,download=True)
# spliting dataset into half with random indexes
indexList = random.sample(list(range(0,len(train_dataset))), (len(train_dataset)//2))
train_dataset_half = torch.utils.data.Subset(train_dataset,indexList)

test_dataset = datasets.CIFAR10(root="dataset/", train=False, transform=transform,download=True)
train_loader = DataLoader(dataset=train_dataset_half,batch_size=batch_size, shuffle=True)
test_loader = DataLoader(dataset=test_dataset, batch_size=batch_size ,shuffle=True)

config =[(1,0.05),(1,0.01),(1,0.001),(2,0.05),(2,0.01),(2,0.001)]
# different setting of learning rates
learning_rate =0.001

criterion = nn.NLLLoss()
for regularizer,lamda in config:
    # Initialize network
    model = CNN(input_channels=in_channels, num_of_classes=num_classes).to(device)

    # Initializing Optimizer
    optimizer= torch.optim.Adam(model.parameters(),lr=learning_rate)

    # Initializing writer for tensorboard
    desc = 'L'+str(regularizer)+'_'+str(lamda)

    print('--------------------------------------------------')
    print('|--------------'+desc+' Experiment ---------------|')
    print('--------------------------------------------------')
    model = trainModelRegularizer(model, optimizer,criterion, learning_rate,
train_loader,test_loader, num_epochs, desc,regularizer,lamda)
```

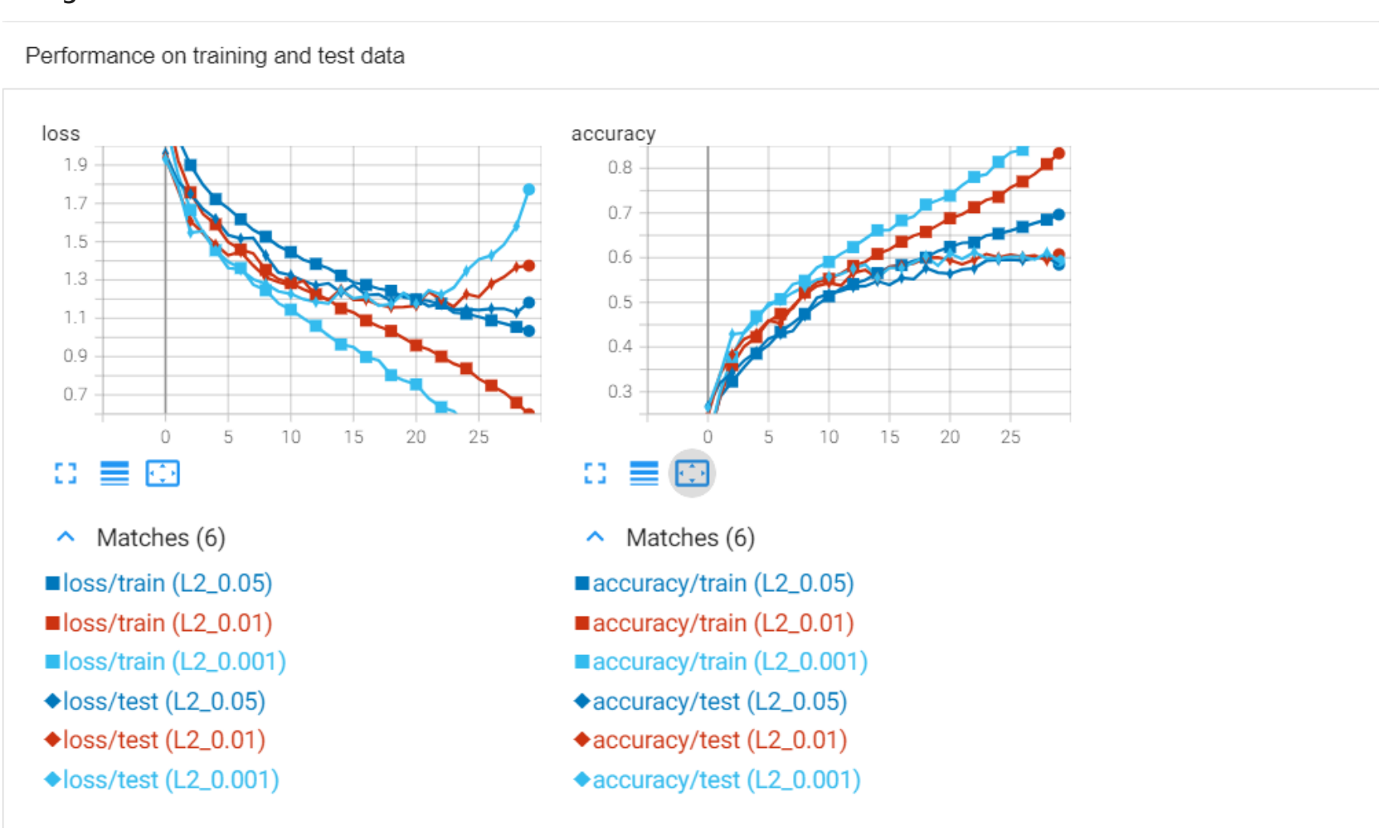
```
cuda
Files already downloaded and verified
Files already downloaded and verified
--------------------------------------------------
|--------------L1_0.05 Experiment ---------------|
--------------------------------------------------
Epoch [5/30],Train Loss: 5.7393,Train Accuracy: 0.1039,Test Loss: 2.3022,Test Accuracy: 0.1039
Epoch [10/30],Train Loss: 5.2505,Train Accuracy: 0.2108,Test Loss: 1.9218,Test Accuracy: 0.2018
Epoch [15/30],Train Loss: 5.1509,Train Accuracy: 0.2268,Test Loss: 1.8809,Test Accuracy: 0.2306
Epoch [20/30],Train Loss: 5.1308,Train Accuracy: 0.2445,Test Loss: 1.8551,Test Accuracy: 0.2376
Epoch [25/30],Train Loss: 5.0988,Train Accuracy: 0.2566,Test Loss: 1.8329,Test Accuracy: 0.2596
Epoch [30/30],Train Loss: 5.0603,Train Accuracy: 0.2731,Test Loss: 1.7930,Test Accuracy: 0.2763
Finished Training
--------------------------------------------------
|--------------L1_0.01 Experiment ---------------|
--------------------------------------------------
Epoch [5/30],Train Loss: 2.6826,Train Accuracy: 0.2480,Test Loss: 1.8766,Test Accuracy: 0.2682
Epoch [10/30],Train Loss: 2.4381,Train Accuracy: 0.3474,Test Loss: 1.6723,Test Accuracy: 0.3604
Epoch [15/30],Train Loss: 2.3134,Train Accuracy: 0.4011,Test Loss: 1.5483,Test Accuracy: 0.4120
Epoch [20/30],Train Loss: 2.2264,Train Accuracy: 0.4448,Test Loss: 1.5621,Test Accuracy: 0.4038
Epoch [25/30],Train Loss: 2.1248,Train Accuracy: 0.4773,Test Loss: 1.4435,Test Accuracy: 0.4507
Epoch [30/30],Train Loss: 2.0666,Train Accuracy: 0.5038,Test Loss: 1.3850,Test Accuracy: 0.4892
Finished Training
--------------------------------------------------
|--------------L1_0.001 Experiment ---------------|
--------------------------------------------------
Epoch [5/30],Train Loss: 1.8451,Train Accuracy: 0.3543,Test Loss: 1.7199,Test Accuracy: 0.3592
Epoch [10/30],Train Loss: 1.6225,Train Accuracy: 0.4344,Test Loss: 1.5022,Test Accuracy: 0.4370
Epoch [15/30],Train Loss: 1.5102,Train Accuracy: 0.4856,Test Loss: 1.3937,Test Accuracy: 0.4710
Epoch [20/30],Train Loss: 1.3947,Train Accuracy: 0.5324,Test Loss: 1.2819,Test Accuracy: 0.5277
Epoch [25/30],Train Loss: 1.3418,Train Accuracy: 0.5594,Test Loss: 1.2528,Test Accuracy: 0.5467
Epoch [30/30],Train Loss: 1.2601,Train Accuracy: 0.5913,Test Loss: 1.2136,Test Accuracy: 0.5677
Finished Training
--------------------------------------------------
|--------------L2_0.05 Experiment ---------------|
--------------------------------------------------
Epoch [5/30],Train Loss: 1.7221,Train Accuracy: 0.3854,Test Loss: 1.6200,Test Accuracy: 0.3896
Epoch [10/30],Train Loss: 1.4777,Train Accuracy: 0.4940,Test Loss: 1.3399,Test Accuracy: 0.5099
Epoch [15/30],Train Loss: 1.3231,Train Accuracy: 0.5656,Test Loss: 1.2347,Test Accuracy: 0.5475
Epoch [20/30],Train Loss: 1.2110,Train Accuracy: 0.6150,Test Loss: 1.2296,Test Accuracy: 0.5667
Epoch [25/30],Train Loss: 1.1242,Train Accuracy: 0.6537,Test Loss: 1.1470,Test Accuracy: 0.5950
Epoch [30/30],Train Loss: 1.0341,Train Accuracy: 0.6965,Test Loss: 1.1822,Test Accuracy: 0.5844
Finished Training
--------------------------------------------------
|--------------L2_0.01 Experiment ---------------|
--------------------------------------------------
Epoch [5/30],Train Loss: 1.5906,Train Accuracy: 0.4224,Test Loss: 1.4859,Test Accuracy: 0.4303
Epoch [10/30],Train Loss: 1.3051,Train Accuracy: 0.5448,Test Loss: 1.2891,Test Accuracy: 0.5367
Epoch [15/30],Train Loss: 1.1520,Train Accuracy: 0.6083,Test Loss: 1.2514,Test Accuracy: 0.5549
Epoch [20/30],Train Loss: 0.9968,Train Accuracy: 0.6727,Test Loss: 1.1592,Test Accuracy: 0.6002
Epoch [25/30],Train Loss: 0.8382,Train Accuracy: 0.7359,Test Loss: 1.2264,Test Accuracy: 0.6008
Epoch [30/30],Train Loss: 0.5997,Train Accuracy: 0.8336,Test Loss: 1.3749,Test Accuracy: 0.6074
Finished Training
--------------------------------------------------
|--------------L2_0.001 Experiment ---------------|
--------------------------------------------------
Epoch [5/30],Train Loss: 1.4553,Train Accuracy: 0.4689,Test Loss: 1.4555,Test Accuracy: 0.4591
Epoch [10/30],Train Loss: 1.1787,Train Accuracy: 0.5777,Test Loss: 1.2408,Test Accuracy: 0.5511
Epoch [15/30],Train Loss: 0.9638,Train Accuracy: 0.6613,Test Loss: 1.2459,Test Accuracy: 0.5552
Epoch [20/30],Train Loss: 0.7757,Train Accuracy: 0.7284,Test Loss: 1.2329,Test Accuracy: 0.5821
Epoch [25/30],Train Loss: 0.5447,Train Accuracy: 0.8143,Test Loss: 1.3475,Test Accuracy: 0.5983
Epoch [30/30],Train Loss: 0.3185,Train Accuracy: 0.8976,Test Loss: 1.7729,Test Accuracy: 0.5935
Finished Training
```

L1 regularization with different Lambda



Comment : Model is not overfitting after L1 regularization and test and training accuracy curves are very close for every value of lambda.Best result is achieved with smaller labmda 0.001

L2 regularization with different Lambda



Comment : for value of lambda 0.05 , training and test curves are closer and hence with low variance it is not overfitting as much as compared to other values of lambda.

Exercise 3: Optimizers

In the last part, you will experiment with two different optimizers i.e. SGD and AdamOptimizer, more specifically, how robust they are to the initial learning rate. The choices for your initial learning rate are left up to you. Please compare and contrast the behavior of these two optimizers specifically on how they react when presented with different learning rates. You should plot the training curves as have been requested in the exercises above.

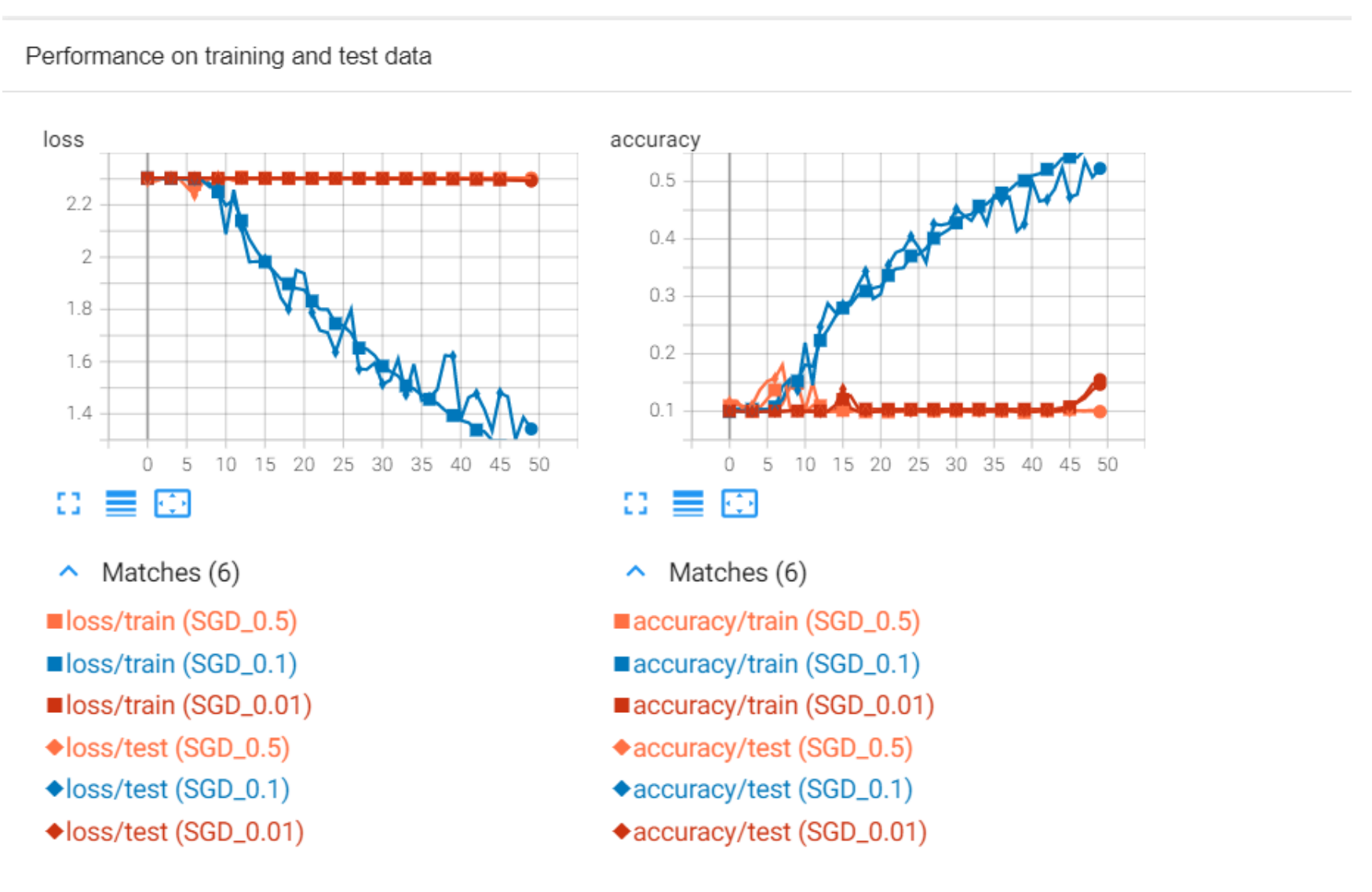
```python
# setting device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

in_channels = 3 # input channels of image
num_classes = 10 # number of output class [0-10]
num_epochs = 50
batch_size= 512

transform = transforms.Compose([transforms.Resize((32,32)),
    transforms.ToTensor()
])

# getting training and test data
train_dataset = datasets.CIFAR10(root="dataset/", train=True, transform=transform,download=True)
# spliting dataset into half with random indexes
indexList = random.sample(list(range(0,len(train_dataset))), (len(train_dataset)//2))
train_dataset_half = torch.utils.data.Subset(train_dataset,indexList)

test_dataset = datasets.CIFAR10(root="dataset/", train=False, transform=transform,download=True)
train_loader = DataLoader(dataset=train_dataset_half,batch_size=batch_size, shuffle=True)
test_loader = DataLoader(dataset=test_dataset, batch_size=batch_size ,shuffle=True)

# different setting of learning rates
learning_rate =0.001
n_total_steps = len(train_loader)
criterion = nn.NLLLoss()
grid = [('Adam',0.01),('Adam',0.001),('Adam',0.0001),('SGD',0.5),('SGD',0.1),('SGD',0.01)]

for opt,lr in grid:

    # Initialize network
    model = CNN(input_channels=in_channels, num_of_classes=num_classes).to(device)

    # Initializing Optimizer
    if(opt=='Adam'):
        optimizer= torch.optim.Adam(model.parameters(),lr=lr)
    else:
        optimizer= torch.optim.SGD(model.parameters(),lr=lr)

    # Initializing writer for tensorboard

    print('----------------------------------------------------------------------------')
    print('|----------- Experiment with Optimizer = '+str(opt)+'---Learning Rate = '+str(lr)+'--
--------|')
    print('----------------------------------------------------------------------------')
    desc =opt+'_'+str(lr)
    model = trainModel(model, optimizer,criterion, learning_rate, train_loader,test_loader,
num_epochs, desc)

    writer = None
    epoch = None
    totalAccuracy = evalModel(model,optimizer,criterion,test_loader,epoch,writer)
    print('Final test Accuracy'+desc+':'+str(totalAccuracy))
```

```
Files already downloaded and verified
Files already downloaded and verified
----------------------------------------------------------------------------
|----------- Experiment with Optimizer = Adam---Learning Rate = 0.01----------|
----------------------------------------------------------------------------
Epoch [5/50],Train Loss: 1.9263,Train Accuracy: 0.2834,Test Loss: 1.8832,Test Accuracy: 0.3098
Epoch [10/50],Train Loss: 1.7809,Train Accuracy: 0.3401,Test Loss: 1.7784,Test Accuracy: 0.3481
Epoch [15/50],Train Loss: 1.6766,Train Accuracy: 0.3849,Test Loss: 1.7311,Test Accuracy: 0.3775
Epoch [20/50],Train Loss: 1.5892,Train Accuracy: 0.4215,Test Loss: 1.6808,Test Accuracy: 0.3856
Epoch [25/50],Train Loss: 1.5212,Train Accuracy: 0.4456,Test Loss: 1.6899,Test Accuracy: 0.3942
Epoch [30/50],Train Loss: 1.4167,Train Accuracy: 0.4830,Test Loss: 1.7430,Test Accuracy: 0.4001
Epoch [35/50],Train Loss: 1.3828,Train Accuracy: 0.4972,Test Loss: 1.8027,Test Accuracy: 0.3851
Epoch [40/50],Train Loss: 1.2907,Train Accuracy: 0.5281,Test Loss: 1.9013,Test Accuracy: 0.3819
Epoch [45/50],Train Loss: 1.2289,Train Accuracy: 0.5517,Test Loss: 2.0087,Test Accuracy: 0.3804
Epoch [50/50],Train Loss: 1.1645,Train Accuracy: 0.5763,Test Loss: 2.0987,Test Accuracy: 0.3824
Finished Training
Final test AccuracyAdam_0.01:(0.3822667896747589, 2.09332537651062)
----------------------------------------------------------------------------
|----------- Experiment with Optimizer = Adam---Learning Rate = 0.001----------|
----------------------------------------------------------------------------
Epoch [5/50],Train Loss: 1.5788,Train Accuracy: 0.4000,Test Loss: 1.5331,Test Accuracy: 0.4226
Epoch [10/50],Train Loss: 1.2725,Train Accuracy: 0.5295,Test Loss: 1.3155,Test Accuracy: 0.5177
Epoch [15/50],Train Loss: 1.0553,Train Accuracy: 0.6182,Test Loss: 1.2078,Test Accuracy: 0.5668
Epoch [20/50],Train Loss: 0.8621,Train Accuracy: 0.6914,Test Loss: 1.1357,Test Accuracy: 0.6027
Epoch [25/50],Train Loss: 0.6708,Train Accuracy: 0.7592,Test Loss: 1.1942,Test Accuracy: 0.6076
Epoch [30/50],Train Loss: 0.4794,Train Accuracy: 0.8295,Test Loss: 1.4144,Test Accuracy: 0.5967
Epoch [35/50],Train Loss: 0.2868,Train Accuracy: 0.8986,Test Loss: 1.7489,Test Accuracy: 0.5956
Epoch [40/50],Train Loss: 0.1695,Train Accuracy: 0.9434,Test Loss: 2.1415,Test Accuracy: 0.5997
Epoch [45/50],Train Loss: 0.0605,Train Accuracy: 0.9825,Test Loss: 2.6190,Test Accuracy: 0.5976
Epoch [50/50],Train Loss: 0.1220,Train Accuracy: 0.9575,Test Loss: 2.7908,Test Accuracy: 0.5912
Finished Training
Final test AccuracyAdam_0.001:(0.5911419987678528, 2.781834363937378)
----------------------------------------------------------------------------
|----------- Experiment with Optimizer = Adam---Learning Rate = 0.0001----------|
----------------------------------------------------------------------------
Epoch [5/50],Train Loss: 1.8049,Train Accuracy: 0.3242,Test Loss: 1.7867,Test Accuracy: 0.3346
Epoch [10/50],Train Loss: 1.6535,Train Accuracy: 0.3839,Test Loss: 1.6549,Test Accuracy: 0.3760
Epoch [15/50],Train Loss: 1.5514,Train Accuracy: 0.4239,Test Loss: 1.5463,Test Accuracy: 0.4221
Epoch [20/50],Train Loss: 1.4869,Train Accuracy: 0.4503,Test Loss: 1.4833,Test Accuracy: 0.4490
Epoch [25/50],Train Loss: 1.4374,Train Accuracy: 0.4717,Test Loss: 1.4315,Test Accuracy: 0.4728
Epoch [30/50],Train Loss: 1.3838,Train Accuracy: 0.4907,Test Loss: 1.3970,Test Accuracy: 0.4883
Epoch [35/50],Train Loss: 1.3451,Train Accuracy: 0.5082,Test Loss: 1.3563,Test Accuracy: 0.5028
Epoch [40/50],Train Loss: 1.3055,Train Accuracy: 0.5207,Test Loss: 1.3243,Test Accuracy: 0.5145
Epoch [45/50],Train Loss: 1.2687,Train Accuracy: 0.5382,Test Loss: 1.3007,Test Accuracy: 0.5256
Epoch [50/50],Train Loss: 1.2478,Train Accuracy: 0.5426,Test Loss: 1.2687,Test Accuracy: 0.5388
Finished Training
Final test AccuracyAdam_0.0001:(0.537304699420929, 1.2709239721298218)
----------------------------------------------------------------------------
|----------- Experiment with Optimizer = SGD---Learning Rate = 0.5----------|
----------------------------------------------------------------------------
Epoch [5/50],Train Loss: 2.3022,Train Accuracy: 0.1061,Test Loss: 2.3016,Test Accuracy: 0.1369
Epoch [10/50],Train Loss: 2.2563,Train Accuracy: 0.1485,Test Loss: 2.3085,Test Accuracy: 0.0995
Epoch [15/50],Train Loss: 2.3029,Train Accuracy: 0.0996,Test Loss: 2.3030,Test Accuracy: 0.1001
Epoch [20/50],Train Loss: 2.3030,Train Accuracy: 0.1023,Test Loss: 2.3029,Test Accuracy: 0.1000
Epoch [25/50],Train Loss: 2.3029,Train Accuracy: 0.1022,Test Loss: 2.3027,Test Accuracy: 0.1004
Epoch [30/50],Train Loss: 2.3029,Train Accuracy: 0.1011,Test Loss: 2.3028,Test Accuracy: 0.1005
Epoch [35/50],Train Loss: 2.3029,Train Accuracy: 0.1040,Test Loss: 2.3029,Test Accuracy: 0.1001
Epoch [40/50],Train Loss: 2.3029,Train Accuracy: 0.0979,Test Loss: 2.3028,Test Accuracy: 0.0993
Epoch [45/50],Train Loss: 2.3030,Train Accuracy: 0.1014,Test Loss: 2.3028,Test Accuracy: 0.1000
Epoch [50/50],Train Loss: 2.3028,Train Accuracy: 0.0988,Test Loss: 2.3029,Test Accuracy: 0.0997
Finished Training
Final test AccuracySGD_0.5:(0.10015510022640228, 2.3028056621551514)
----------------------------------------------------------------------------
|----------- Experiment with Optimizer = SGD---Learning Rate = 0.1----------|
----------------------------------------------------------------------------
Epoch [5/50],Train Loss: 2.3016,Train Accuracy: 0.1036,Test Loss: 2.3014,Test Accuracy: 0.1006
Epoch [10/50],Train Loss: 2.2500,Train Accuracy: 0.1527,Test Loss: 2.2669,Test Accuracy: 0.1366
Epoch [15/50],Train Loss: 2.0209,Train Accuracy: 0.2648,Test Loss: 1.9829,Test Accuracy: 0.2717
Epoch [20/50],Train Loss: 1.8806,Train Accuracy: 0.3143,Test Loss: 1.9506,Test Accuracy: 0.2957
Epoch [25/50],Train Loss: 1.7459,Train Accuracy: 0.3704,Test Loss: 1.6363,Test Accuracy: 0.4042
Epoch [30/50],Train Loss: 1.6238,Train Accuracy: 0.4164,Test Loss: 1.5950,Test Accuracy: 0.4266
Epoch [35/50],Train Loss: 1.4964,Train Accuracy: 0.4608,Test Loss: 1.5907,Test Accuracy: 0.4261
Epoch [40/50],Train Loss: 1.3941,Train Accuracy: 0.5013,Test Loss: 1.6211,Test Accuracy: 0.4259
Epoch [45/50],Train Loss: 1.2991,Train Accuracy: 0.5400,Test Loss: 1.3352,Test Accuracy: 0.5236
Epoch [50/50],Train Loss: 1.2140,Train Accuracy: 0.5696,Test Loss: 1.3423,Test Accuracy: 0.5226
Finished Training
Final test AccuracySGD_0.1:(0.5244140625, 1.339640736579895)
----------------------------------------------------------------------------
|----------- Experiment with Optimizer = SGD---Learning Rate = 0.01----------|
----------------------------------------------------------------------------
Epoch [5/50],Train Loss: 2.3028,Train Accuracy: 0.1006,Test Loss: 2.3030,Test Accuracy: 0.0992
Epoch [10/50],Train Loss: 2.3024,Train Accuracy: 0.1006,Test Loss: 2.3025,Test Accuracy: 0.1002
Epoch [15/50],Train Loss: 2.3021,Train Accuracy: 0.1026,Test Loss: 2.3023,Test Accuracy: 0.1107
Epoch [20/50],Train Loss: 2.3019,Train Accuracy: 0.1036,Test Loss: 2.3021,Test Accuracy: 0.0991
Epoch [25/50],Train Loss: 2.3015,Train Accuracy: 0.1036,Test Loss: 2.3017,Test Accuracy: 0.0999
Epoch [30/50],Train Loss: 2.3011,Train Accuracy: 0.1036,Test Loss: 2.3012,Test Accuracy: 0.1000
Epoch [35/50],Train Loss: 2.3004,Train Accuracy: 0.1036,Test Loss: 2.3005,Test Accuracy: 0.0998
Epoch [40/50],Train Loss: 2.2992,Train Accuracy: 0.1035,Test Loss: 2.2992,Test Accuracy: 0.1002
Epoch [45/50],Train Loss: 2.2972,Train Accuracy: 0.1061,Test Loss: 2.2970,Test Accuracy: 0.1017
Epoch [50/50],Train Loss: 2.2930,Train Accuracy: 0.1466,Test Loss: 2.2922,Test Accuracy: 0.1552
Finished Training
Final test AccuracySGD_0.01:(0.15521599352359772, 2.2923312187194824)
```

Adam with different learning rates

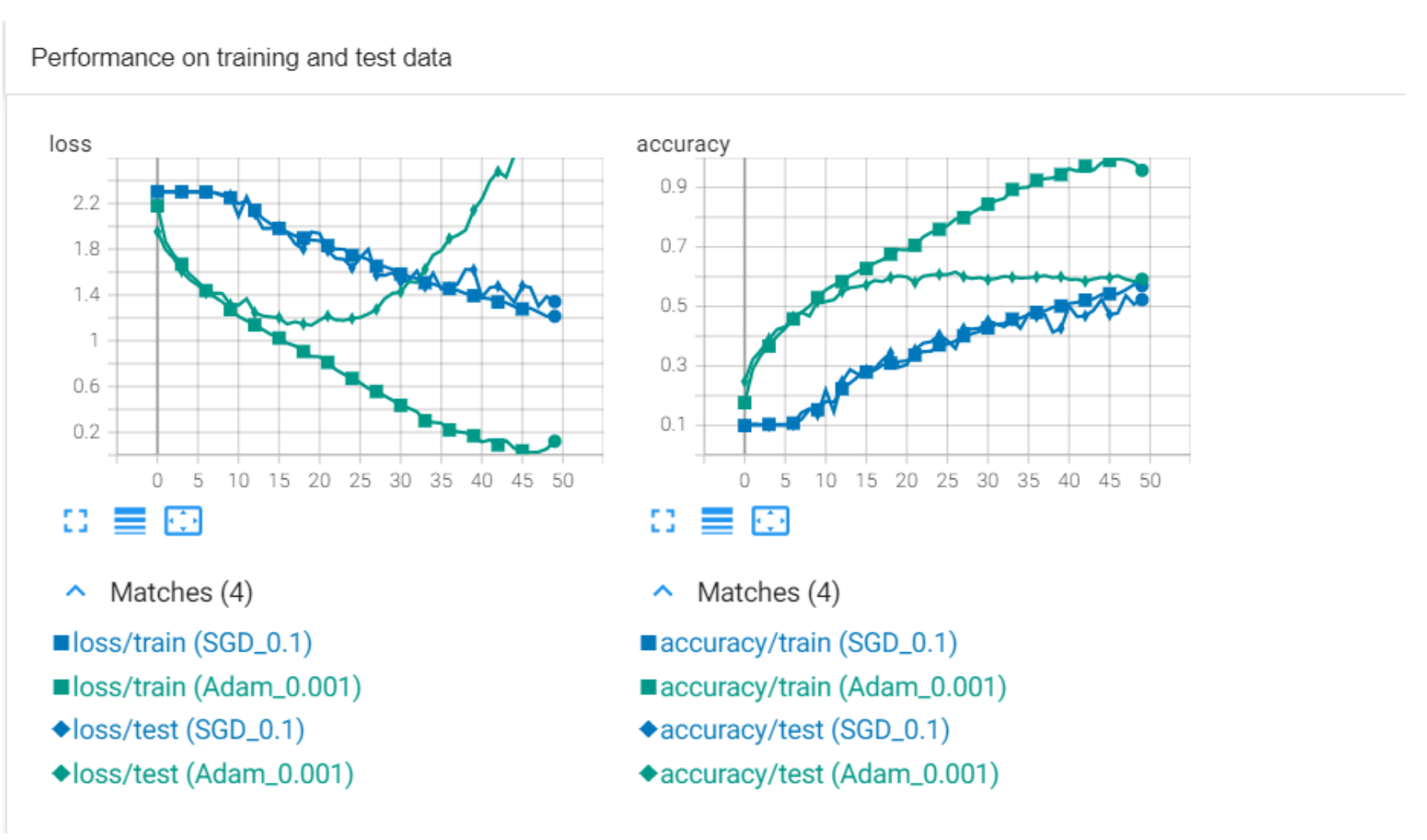
Comment : Adams best test accuracy is achieved with learning rate 0.001 however difference btw train and test accuracy is also hight for learning rate 0.001. With smaller learning rate 0.0001 this difference is small.However Adam is quite robust to learning rate.

SGD with different learning rates



Comment : SGD best test accuracy is achieved with learning rate 0.1. Taking smaller learning rate, decreases the performance.

Adam vs SGD



Comment : Adam works better on smaller learning rate in comparison to SGD which works better on 0.1. However, difference in training and test curves is smaller with SGD.