Lab Course: Distributed Data Analytics
Exercise Sheet 6
Group 2 - Monday

Submitted by: Sruthy Annie Santhosh, 312213

Topic:Distributed Machine Learning (Supervised)

- 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 jupyter notebook. The same was observed in other M1 chips also
The output images attached are screenshots.

- **Exercise 1: PyTorch Network Analysis**

As given in the exercise sheet , I have implemented the LeNet model in paper: LeNet 1998 paper: LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

LeNet architecture from paper:

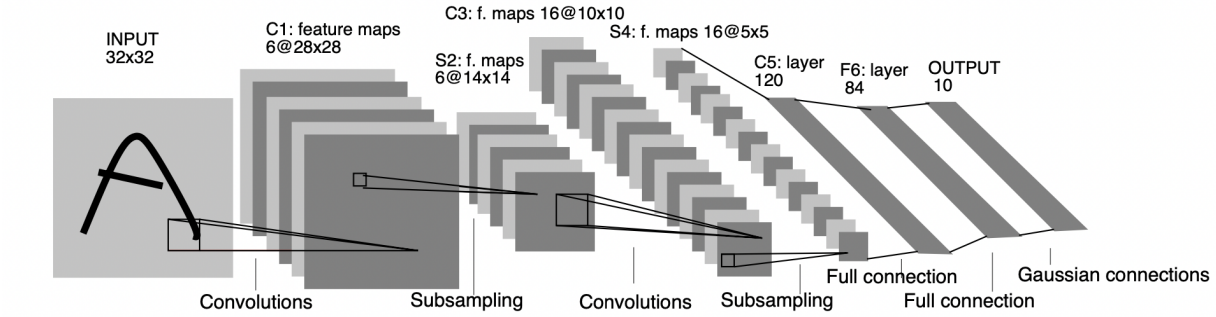


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Since the model takes input of size (32*32), I have resized the MNIST images to that size. Also the number of channels are assigned according to the given dataset. Kernel size 5 and stride 1 are taken as given in the model. All the network parameters and layers of the model are the same as given in fig above taken from the paper.

Datasets

MNIST Dataset: size (28 * 28) and number of channels : 1
CIFAR10 Dataset: size (32 * 32) and number of channels : 3

As given in the exercise sheet, I have fixed the optimiser as ADAM and have tried training the model for different learning rates like [0.1, 0.01, 0.001]. The loss function used in Cross Entropy Loss.

I have created the logs for storing the losses and accuracies using tensorflow summary. The tensorboard is loaded in the notebook.

```
#Loading tensorboard in notebook
%load_ext tensorboard
```

```
#Creating Logs for storing the train and test values for both datasets
current_time = str(datetime.datetime.now().timestamp())
train_log_dir_mnist = 'logs/tensorboard/mnist/train/' + current_time
test_log_dir_mnist = 'logs/tensorboard/mnist/test/' + current_time
train_summary_writer_mnist = summary.create_file_writer(train_log_dir_mnist)
test_summary_writer_mnist = summary.create_file_writer(test_log_dir_mnist)

train_log_dir_cifar10 = 'logs/tensorboard/cifar10/train/' + current_time
test_log_dir_cifar10 = 'logs/tensorboard/cifar10/test/' + current_time
train_summary_writer_cifar10 = summary.create_file_writer(train_log_dir_cifar10)
test_summary_writer_cifar10 = summary.create_file_writer(test_log_dir_cifar10)
```

```
#Using Summarywriter
writer = SummaryWriter("Lab 7 Tensorboard")
```

Mnist and Cifar10 datasets are loaded using torchvision libraries and resized as mentioned before. Then using dataloader, the required dataset is loaded (train loader and test loader).

LeNet model is defined as below.

```
#Defining the convolutional neural network as given in paper
class LeNet5(nn.Module):
    def __init__(self, num_classes,dataset='mnist'):
        #Initializing the number of channels depending on the dataset
        if(dataset=='mnist'):
            no_channels = 1
        elif(dataset=='cifar10'):
            no_channels = 3
        super(LeNet5, self).__init__()

        #Initialising each layer with the network params
        self.layer1 = nn.Sequential(
            nn.Conv2d(no_channels, 6, kernel_size=5, stride=1, padding=0),
            nn.BatchNorm2d(6),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size = 2, stride = 2))
        self.layer2 = nn.Sequential(
            nn.Conv2d(6, 16, kernel_size=5, stride=1, padding=0),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size = 2, stride = 2))

        self.fc = nn.Linear(400, 120)
        self.relu = nn.ReLU()
        self.fc1 = nn.Linear(120, 84)
        self.relu1 = nn.ReLU()
        self.fc2 = nn.Linear(84, num_classes)

    def forward(self, x):
        #Calling each layer
        out = self.layer1(x)
        out = self.layer2(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)
        return out
```

The model is initialized as follows:

```
#Initializing the model
model = LeNet5(num_classes,'mnist').to(device)

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

#Setting the optimiser
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate[2])

#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 if sound at half 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.

```
print ('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f} for learning rate: {}'
        .format(epoch+1, num_epochs, i+1, total_step, running_loss/469,learning_rate[2]))
print('Accuracy of the network on the train images: {:.4f} % for epoch: {}'.format(100 * correct / total,epoch+1))
with train_summary_writer_mnist.as_default():

  tf.summary.scalar('Mnist training loss for each step in epochs for learning rate 0.001 ', running_loss / 469, epoch * total_step + i)
  tf.summary.scalar('Mnist train Accuracy for each step in epochs for learning rate 0.001 ', 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. 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 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

    print ('Average test Loss: {:.4f} for learning rate 0.001'
                    .format(running_loss/total_step))
    print('Accuracy of the network on the test images: {:.4f} % for learning rate 0.001'.format(100 * correct / total))
    with test_summary_writer_mnist.as_default():

            tf.summary.scalar('Mnist average test loss for learning rate 0.001', running_loss / total_step, 1)
            tf.summary.scalar('Mnist average test Accuracy for learning rate 0.001 ', 100 * correct / total, 1)
```

The same is carried out for CIFAR10 datasets and for different learning rates. The outputs and graphs obtained from tensorboard are plotted below:

MNIST Dataset Training losses and accuracies for learning rate 0.1

Output

```
Epoch [6/10], Step [938/938], Loss: 2.3100 for learning rate: 0.1
Accuracy of the network on the train images: 10.2054 % for epoch: 6
```

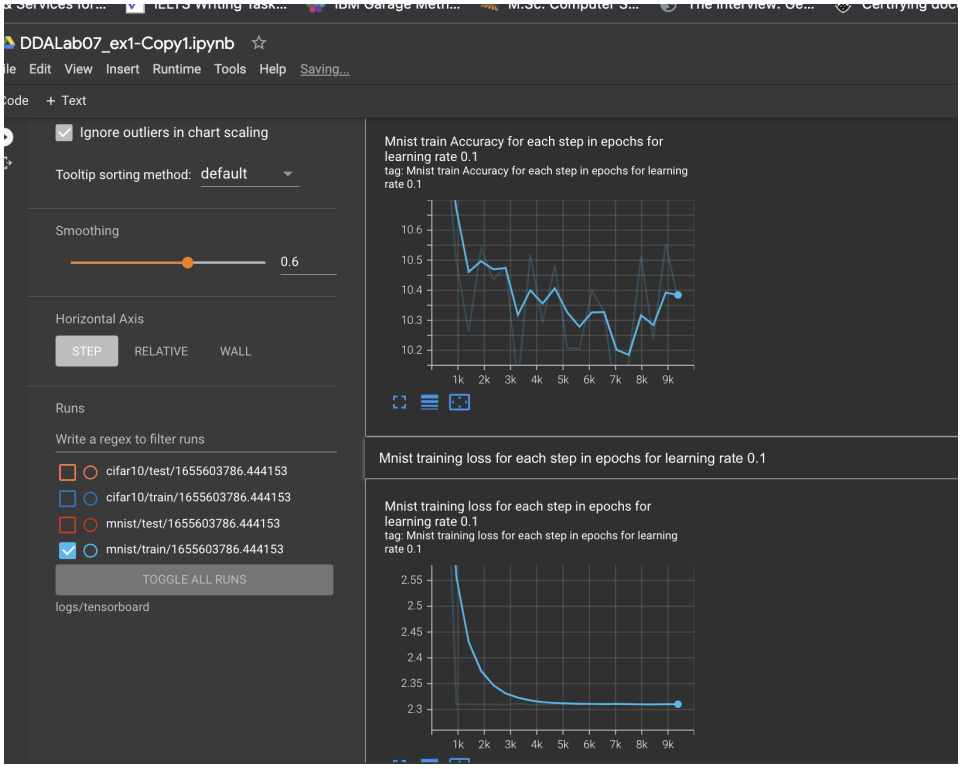
```
Epoch [7/10], Step [469/938], Loss: 2.3106 for learning rate: 0.1
Accuracy of the network on the train images: 10.3978 % for epoch: 7
```

```
Epoch [7/10], Step [938/938], Loss: 2.3101 for learning rate: 0.1
Accuracy of the network on the train images: 10.3288 % for epoch: 7
```

```
Epoch [8/10], Step [938/938], Loss: 2.3099 for learning rate: 0.1
Accuracy of the network on the train images: 10.1588 % for epoch: 8
```

```
Epoch [10/10], Step [938/938], Loss: 2.3103 for learning rate: 0.1
Accuracy of the network on the train images: 10.3722 % for epoch: 10
```

Graph in Tensorboard

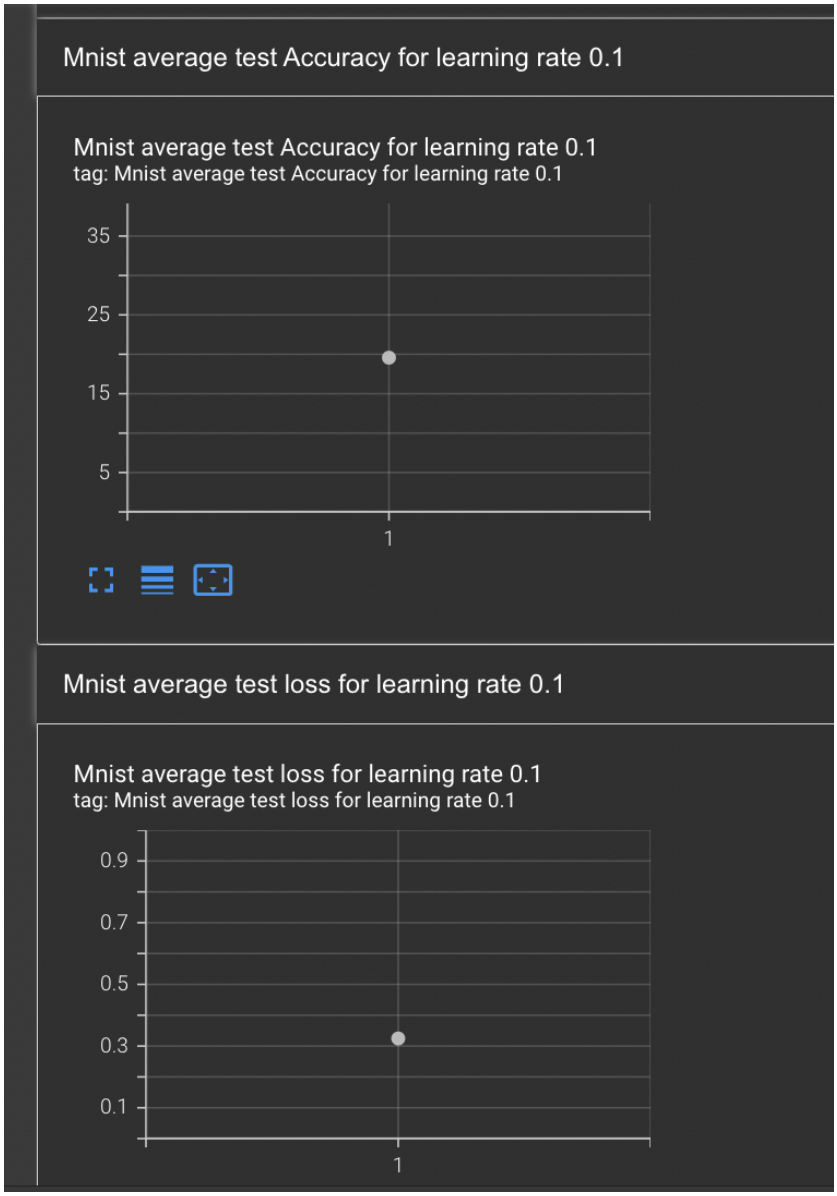


MNIST Dataset Testing losses and accuracies for learning rate 0.1

Output

```
Average test Loss: 0.3246 for learning rate 0.1
Accuracy of the network on the test images: 19.5700 % for learning rate 0.1
```

Graph in Tensorboard

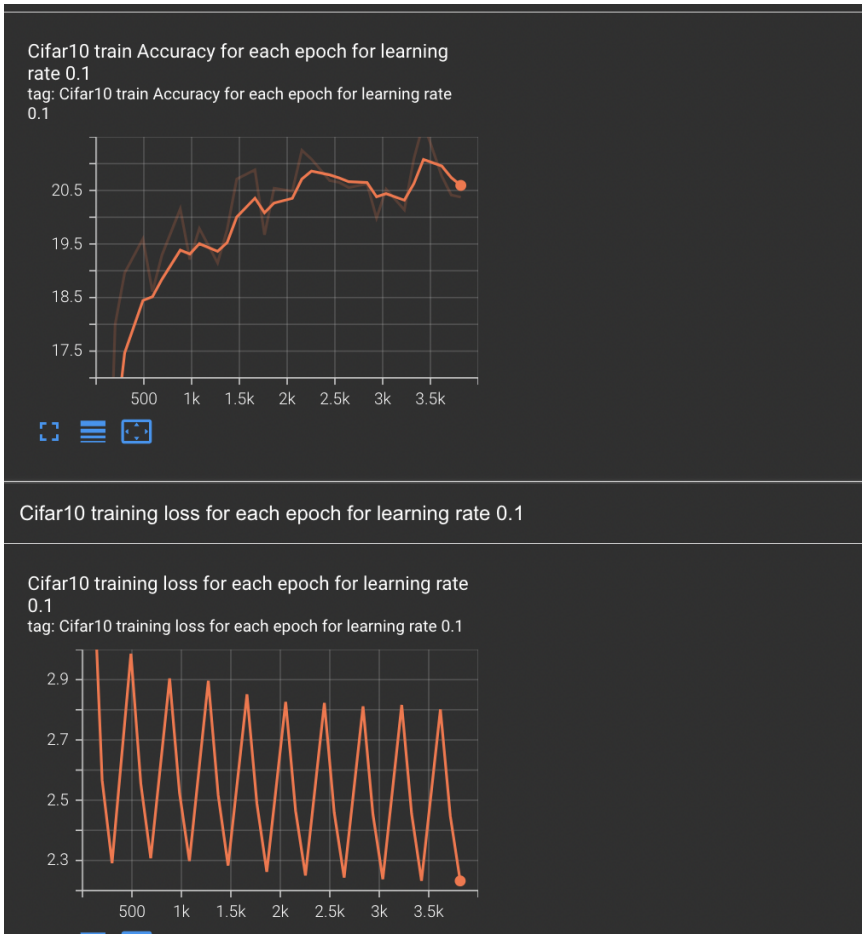


CIFAR10 Dataset Training losses and accuracies for learning rate 0.1

Output

```
Accuracy of the network on the train images: 20.5000 % for learning rate 0.1
Epoch [3/10], Step [100/391], Loss: 1.8558 for learning rate 0.1
Accuracy of the network on the train images: 21.3648 % for learning rate 0.1
Epoch [3/10], Step [200/391], Loss: 2.0125 for learning rate 0.1
Accuracy of the network on the train images: 20.7266 % for learning rate 0.1
Epoch [3/10], Step [300/391], Loss: 1.9199 for learning rate 0.1
Accuracy of the network on the train images: 21.7266 % for learning rate 0.1
Epoch [4/10], Step [100/391], Loss: 1.9142 for learning rate 0.1
Accuracy of the network on the train images: 21.9672 % for learning rate 0.1
Epoch [4/10], Step [200/391], Loss: 1.8810 for learning rate 0.1
Accuracy of the network on the train images: 22.3672 % for learning rate 0.1
Epoch [4/10], Step [300/391], Loss: 1.9066 for learning rate 0.1
Accuracy of the network on the train images: 22.7500 % for learning rate 0.1
Epoch [5/10], Step [100/391], Loss: 1.8683 for learning rate 0.1
Accuracy of the network on the train images: 22.8361 % for learning rate 0.1
Epoch [5/10], Step [200/391], Loss: 1.9194 for learning rate 0.1
Accuracy of the network on the train images: 21.3750 % for learning rate 0.1
Epoch [5/10], Step [300/391], Loss: 1.8897 for learning rate 0.1
Accuracy of the network on the train images: 22.6719 % for learning rate 0.1
Epoch [6/10], Step [100/391], Loss: 1.8717 for learning rate 0.1
Accuracy of the network on the train images: 22.0369 % for learning rate 0.1
Epoch [6/10], Step [200/391], Loss: 1.9702 for learning rate 0.1
Accuracy of the network on the train images: 21.0312 % for learning rate 0.1
Epoch [6/10], Step [300/391], Loss: 1.9961 for learning rate 0.1
Accuracy of the network on the train images: 21.0547 % for learning rate 0.1
Epoch [7/10], Step [100/391], Loss: 1.8770 for learning rate 0.1
Accuracy of the network on the train images: 21.4098 % for learning rate 0.1
Epoch [7/10], Step [200/391], Loss: 1.9143 for learning rate 0.1
Accuracy of the network on the train images: 21.8281 % for learning rate 0.1
Accuracy of the network on the train images: 21.8125 % for learning rate 0.1
Epoch [8/10], Step [300/391], Loss: 1.9172 for learning rate 0.1
Accuracy of the network on the train images: 21.6875 % for learning rate 0.1
Epoch [9/10], Step [100/391], Loss: 1.9565 for learning rate 0.1
Accuracy of the network on the train images: 21.5410 % for learning rate 0.1
Epoch [9/10], Step [200/391], Loss: 2.1949 for learning rate 0.1
Accuracy of the network on the train images: 20.2734 % for learning rate 0.1
Epoch [9/10], Step [300/391], Loss: 2.0282 for learning rate 0.1
Accuracy of the network on the train images: 21.6250 % for learning rate 0.1
Epoch [10/10], Step [100/391], Loss: 1.8178 for learning rate 0.1
Accuracy of the network on the train images: 21.0082 % for learning rate 0.1
Epoch [10/10], Step [200/391], Loss: 2.0377 for learning rate 0.1
Accuracy of the network on the train images: 20.5391 % for learning rate 0.1
Epoch [10/10], Step [300/391], Loss: 1.7952 for learning rate 0.1
Accuracy of the network on the train images: 21.4531 % for learning rate 0.1
```

Graph in Tensorboard

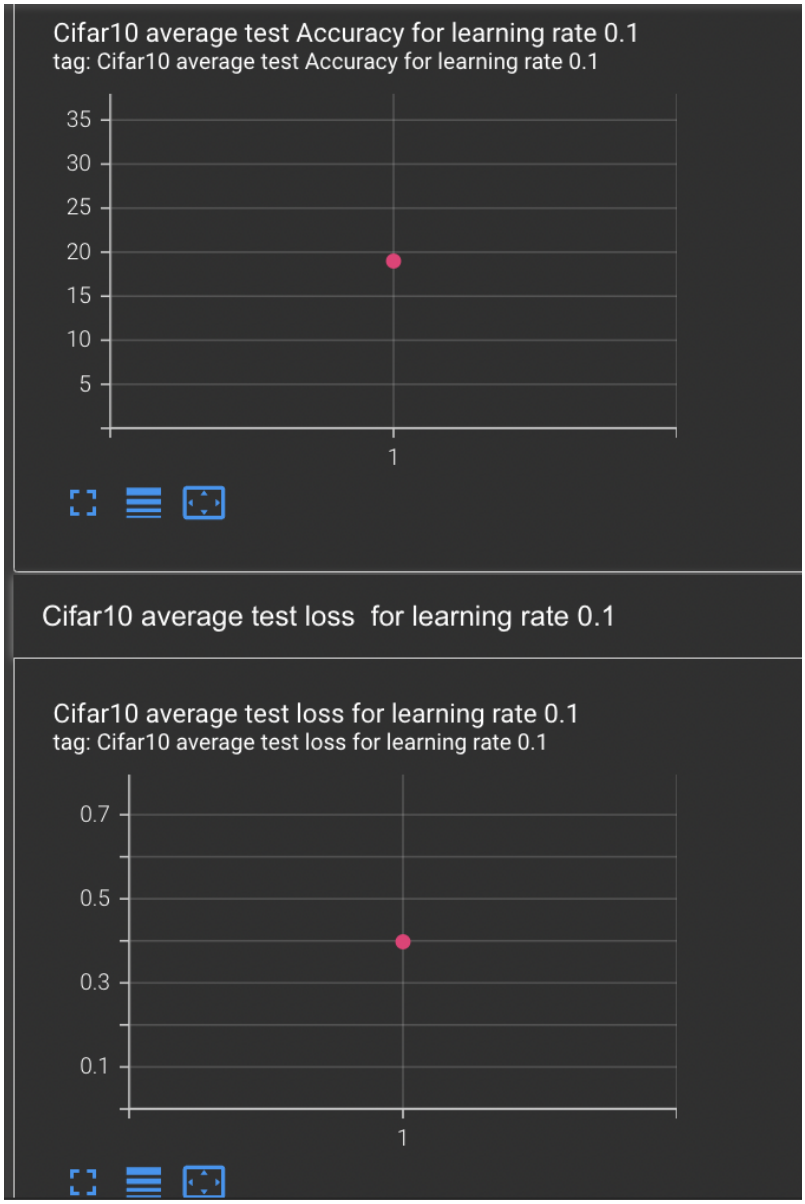


CIFAR10 Dataset Testing losses and accuracies for learning rate 0.1

Output

```
Average test Loss: 0.3978 for learning rate 0.1
Accuracy of the network on the test images: 18.9900 %
```

Graph in Tensorboard



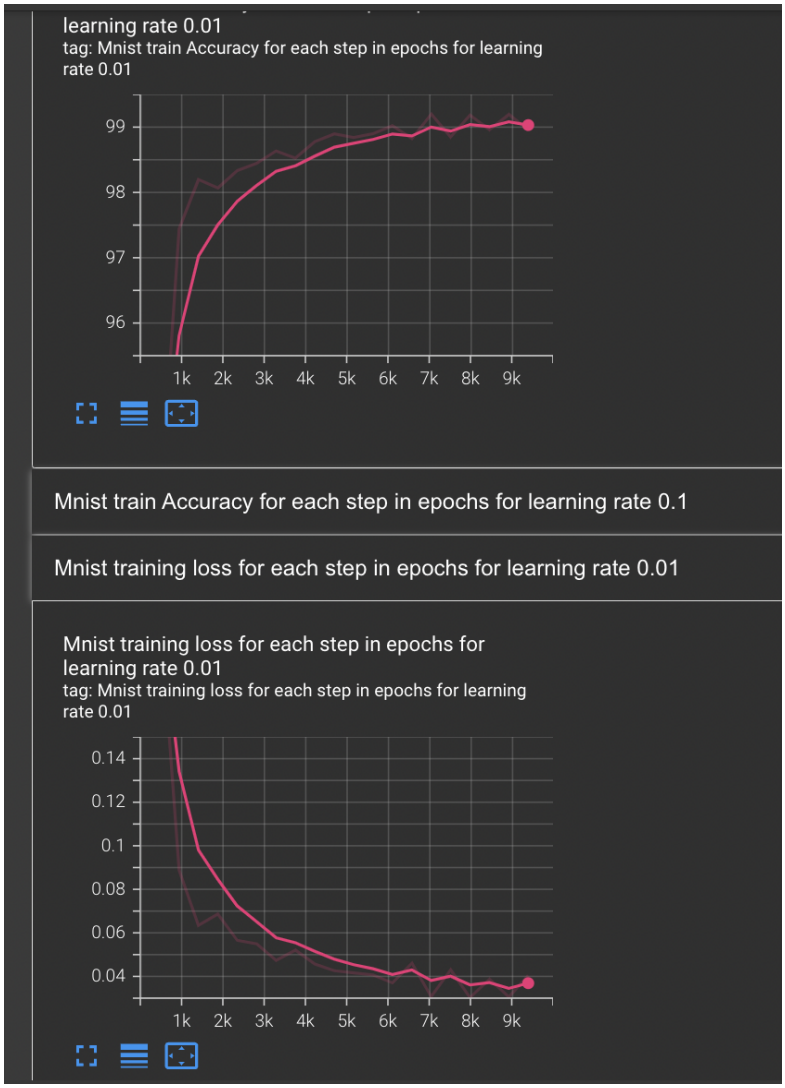
MNIST Dataset Training losses and accuracies for learning rate 0.01

Output

```
Epoch [1/10], Step [469/938], Loss: 0.2096 for learning rate: 0.01
Accuracy of the network on the train images: 93.0770 % for epoch: 1
Epoch [1/10], Step [938/938], Loss: 0.0887 for learning rate: 0.01
Accuracy of the network on the train images: 97.4420 % for epoch: 1
Epoch [2/10], Step [469/938], Loss: 0.0634 for learning rate: 0.01
Accuracy of the network on the train images: 98.1976 % for epoch: 2
Epoch [2/10], Step [938/938], Loss: 0.0687 for learning rate: 0.01
Accuracy of the network on the train images: 98.0690 % for epoch: 2
Epoch [3/10], Step [469/938], Loss: 0.0565 for learning rate: 0.01
Accuracy of the network on the train images: 98.3342 % for epoch: 3
Epoch [3/10], Step [938/938], Loss: 0.0549 for learning rate: 0.01
Accuracy of the network on the train images: 98.4458 % for epoch: 3
```

```
Epoch [8/10], Step [469/938], Loss: 0.0308 for learning rate: 0.01
Accuracy of the network on the train images: 99.2004 % for epoch: 8
Epoch [8/10], Step [938/938], Loss: 0.0430 for learning rate: 0.01
Accuracy of the network on the train images: 98.8494 % for epoch: 8
Epoch [9/10], Step [469/938], Loss: 0.0303 for learning rate: 0.01
Accuracy of the network on the train images: 99.1838 % for epoch: 9
Epoch [9/10], Step [938/938], Loss: 0.0387 for learning rate: 0.01
Accuracy of the network on the train images: 98.9594 % for epoch: 9
Epoch [10/10], Step [469/938], Loss: 0.0306 for learning rate: 0.01
Accuracy of the network on the train images: 99.1938 % for epoch: 10
Epoch [10/10], Step [938/938], Loss: 0.0404 for learning rate: 0.01
Accuracy of the network on the train images: 98.9561 % for epoch: 10
```

Graph in Tensorboard

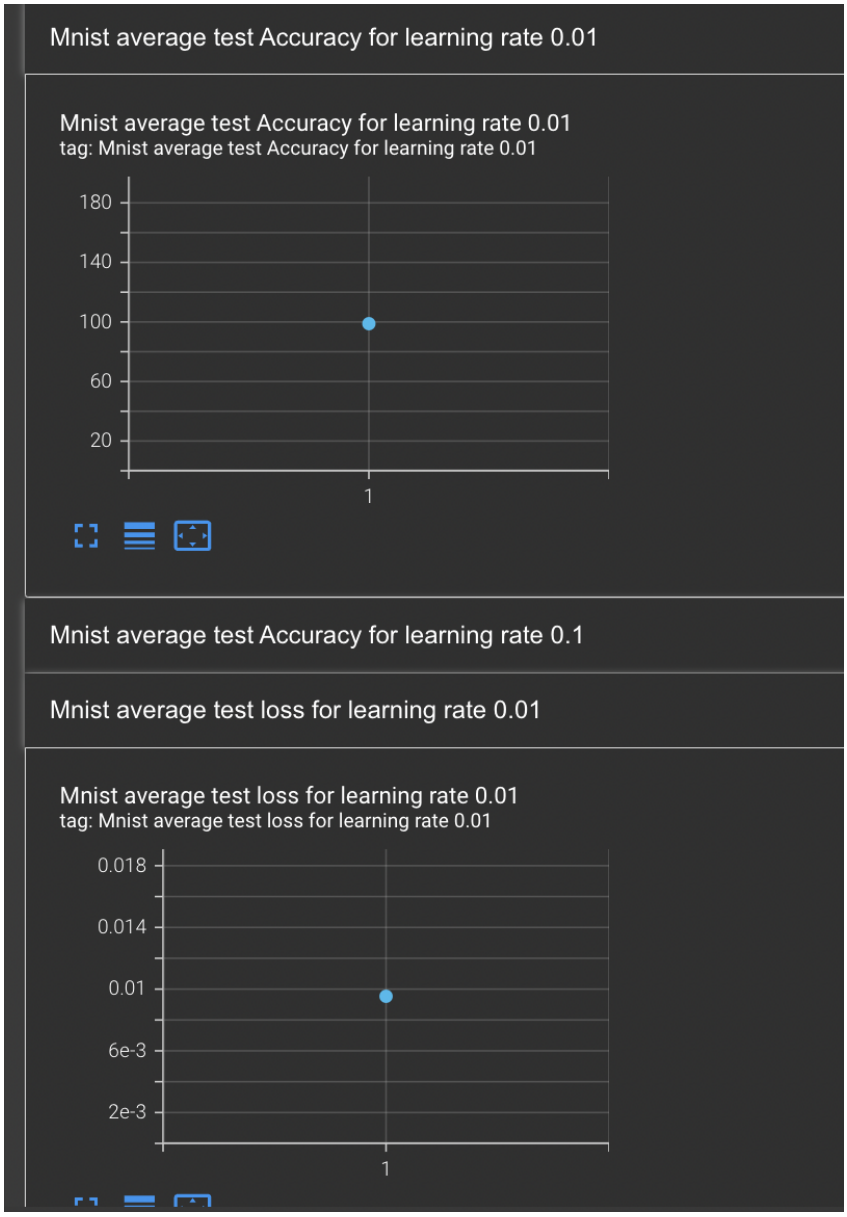


MNIST Dataset Testing losses and accuracies for learning rate 0.01

Output

```
Average test Loss: 0.0095 for learning rate 0.01
Accuracy of the network on the test images: 98.8300 % for learning rate 0.01
```

Graph in Tensorboard



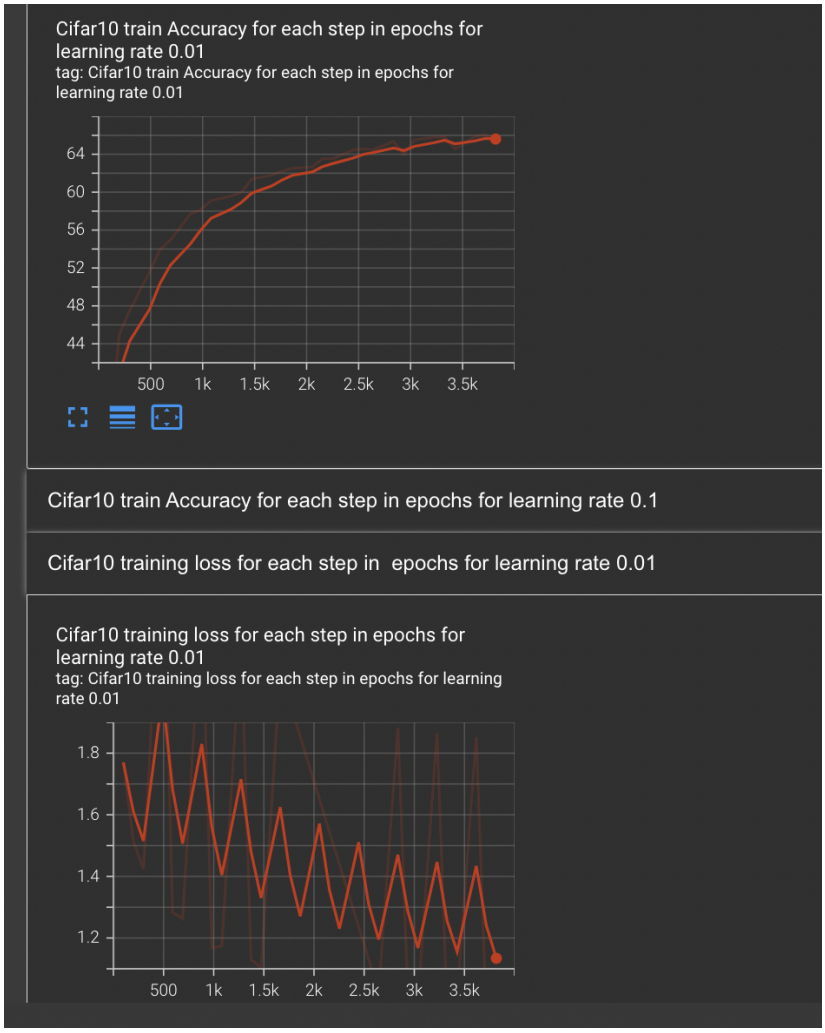
CIFAR10 Dataset Training losses and accuracies for learning rate 0.01

Output

```
Epoch [1/10], Step [200/391], Loss: 1.5141 for learning rate 0.01
Accuracy of the network on the train images: 45.0000 % for learning rate 0.01
Epoch [1/10], Step [300/391], Loss: 1.4241 for learning rate 0.01
Accuracy of the network on the train images: 47.5234 % for learning rate 0.01
Epoch [2/10], Step [100/391], Loss: 2.5470 for learning rate 0.01
Accuracy of the network on the train images: 51.5861 % for learning rate 0.01
Epoch [2/10], Step [200/391], Loss: 1.2831 for learning rate 0.01
Accuracy of the network on the train images: 53.9375 % for learning rate 0.01
Epoch [2/10], Step [300/391], Loss: 1.2628 for learning rate 0.01
Accuracy of the network on the train images: 54.9375 % for learning rate 0.01
Epoch [3/10], Step [100/391], Loss: 2.2913 for learning rate 0.01
Accuracy of the network on the train images: 57.6885 % for learning rate 0.01
Epoch [3/10], Step [200/391], Loss: 1.1670 for learning rate 0.01
Accuracy of the network on the train images: 58.1328 % for learning rate 0.01
Epoch [3/10], Step [300/391], Loss: 1.1746 for learning rate 0.01
Accuracy of the network on the train images: 59.1016 % for learning rate 0.01
```

```
Epoch [8/10], Step [200/391], Loss: 1.0121 for learning rate 0.01
Accuracy of the network on the train images: 63.9766 % for learning rate 0.01
Epoch [8/10], Step [300/391], Loss: 0.9891 for learning rate 0.01
Accuracy of the network on the train images: 65.5078 % for learning rate 0.01
Epoch [9/10], Step [100/391], Loss: 1.8650 for learning rate 0.01
Accuracy of the network on the train images: 65.7951 % for learning rate 0.01
Epoch [9/10], Step [200/391], Loss: 0.9688 for learning rate 0.01
Accuracy of the network on the train images: 65.8516 % for learning rate 0.01
Epoch [9/10], Step [300/391], Loss: 1.0055 for learning rate 0.01
Accuracy of the network on the train images: 64.5547 % for learning rate 0.01
Epoch [10/10], Step [100/391], Loss: 1.8520 for learning rate 0.01
Accuracy of the network on the train images: 65.8525 % for learning rate 0.01
Epoch [10/10], Step [200/391], Loss: 0.9508 for learning rate 0.01
Accuracy of the network on the train images: 66.0625 % for learning rate 0.01
Epoch [10/10], Step [300/391], Loss: 0.9739 for learning rate 0.01
Accuracy of the network on the train images: 65.5000 % for learning rate 0.01
```

Graph in Tensorboard

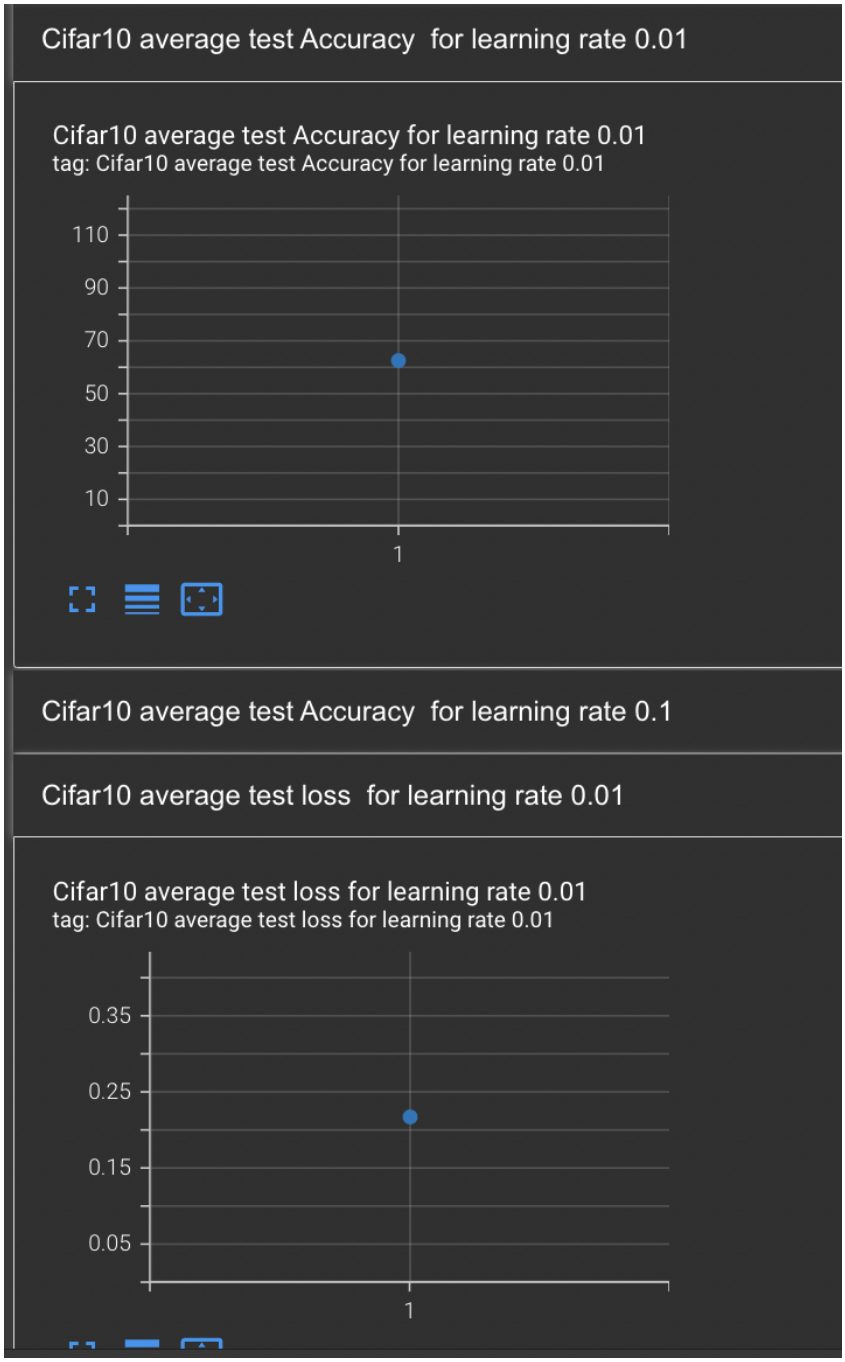


CIFAR10 Dataset Testing losses and accuracies for learning rate 0.01

Output

```
Average test Loss: 0.2170 for learning rate 0.01
Accuracy of the network on the test images: 62.5200 %
```

Graph in Tensorboard

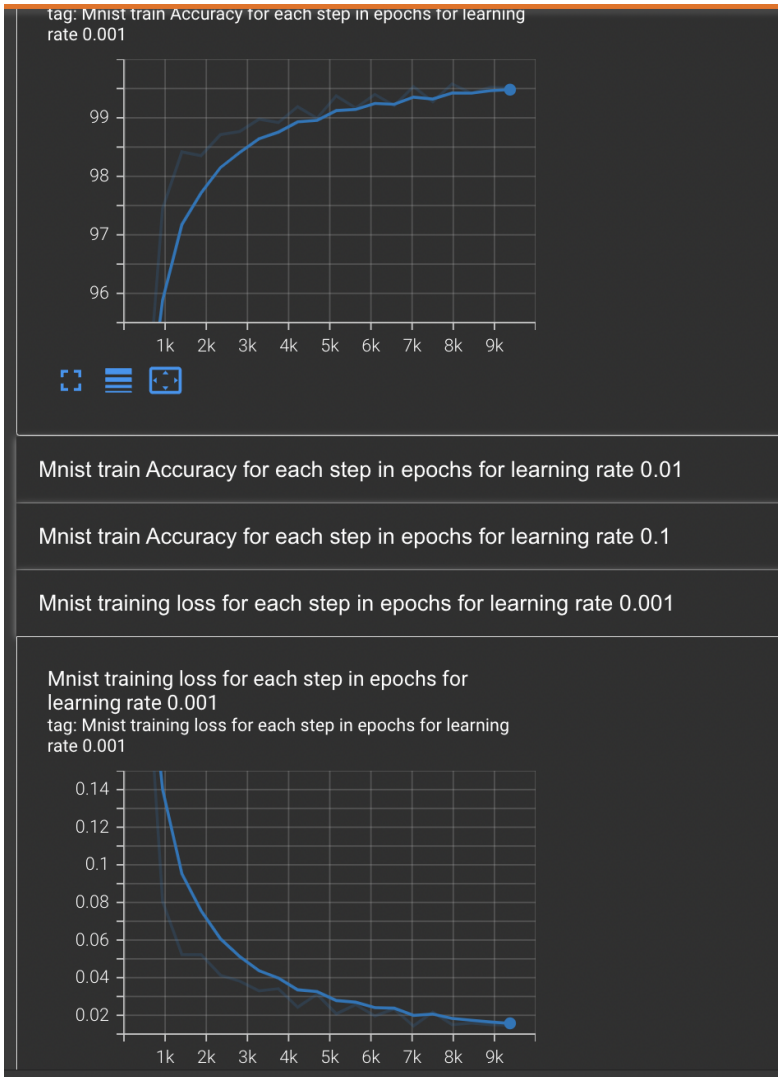


MNIST Dataset Training losses and accuracies for learning rate 0.001

Output

```
Epoch [1/10], Step [469/938], Loss: 0.2396 for learning rate: 0.001
Accuracy of the network on the train images: 93.2669 % for epoch: 1
Epoch [1/10], Step [938/938], Loss: 0.0805 for learning rate: 0.001
Accuracy of the network on the train images: 97.4520 % for epoch: 1
Epoch [2/10], Step [469/938], Loss: 0.0523 for learning rate: 0.001
Accuracy of the network on the train images: 98.4175 % for epoch: 2
Epoch [2/10], Step [938/938], Loss: 0.0522 for learning rate: 0.001
Accuracy of the network on the train images: 98.3525 % for epoch: 2
Epoch [3/10], Step [469/938], Loss: 0.0413 for learning rate: 0.001
Accuracy of the network on the train images: 98.7140 % for epoch: 3
Epoch [3/10], Step [938/938], Loss: 0.0381 for learning rate: 0.001
Accuracy of the network on the train images: 98.7660 % for epoch: 3
Epoch [4/10], Step [469/938], Loss: 0.0330 for learning rate: 0.001
Accuracy of the network on the train images: 98.9772 % for epoch: 4
Epoch [4/10], Step [938/938], Loss: 0.0341 for learning rate: 0.001
Accuracy of the network on the train images: 98.9194 % for epoch: 4
Epoch [5/10], Step [469/938], Loss: 0.0242 for learning rate: 0.001
Accuracy of the network on the train images: 99.1904 % for epoch: 5
Epoch [5/10], Step [938/938], Loss: 0.0312 for learning rate: 0.001
Accuracy of the network on the train images: 98.9928 % for epoch: 5
Epoch [6/10], Step [469/938], Loss: 0.0208 for learning rate: 0.001
Accuracy of the network on the train images: 99.3737 % for epoch: 6
Epoch [6/10], Step [938/938], Loss: 0.0257 for learning rate: 0.001
Accuracy of the network on the train images: 99.1696 % for epoch: 6
Epoch [7/10], Step [469/938], Loss: 0.0196 for learning rate: 0.001
Accuracy of the network on the train images: 99.4003 % for epoch: 7
Epoch [7/10], Step [938/938], Loss: 0.0233 for learning rate: 0.001
Accuracy of the network on the train images: 99.2096 % for epoch: 7
Epoch [8/10], Step [469/938], Loss: 0.0143 for learning rate: 0.001
Accuracy of the network on the train images: 99.5336 % for epoch: 8
Epoch [8/10], Step [938/938], Loss: 0.0217 for learning rate: 0.001
Accuracy of the network on the train images: 99.2763 % for epoch: 8
Epoch [9/10], Step [469/938], Loss: 0.0150 for learning rate: 0.001
Accuracy of the network on the train images: 99.5769 % for epoch: 9
Epoch [9/10], Step [938/938], Loss: 0.0159 for learning rate: 0.001
Accuracy of the network on the train images: 99.4197 % for epoch: 9
Epoch [10/10], Step [469/938], Loss: 0.0149 for learning rate: 0.001
Accuracy of the network on the train images: 99.5269 % for epoch: 10
Epoch [10/10], Step [938/938], Loss: 0.0147 for learning rate: 0.001
Accuracy of the network on the train images: 99.5064 % for epoch: 10
```

Graph in Tensorboard

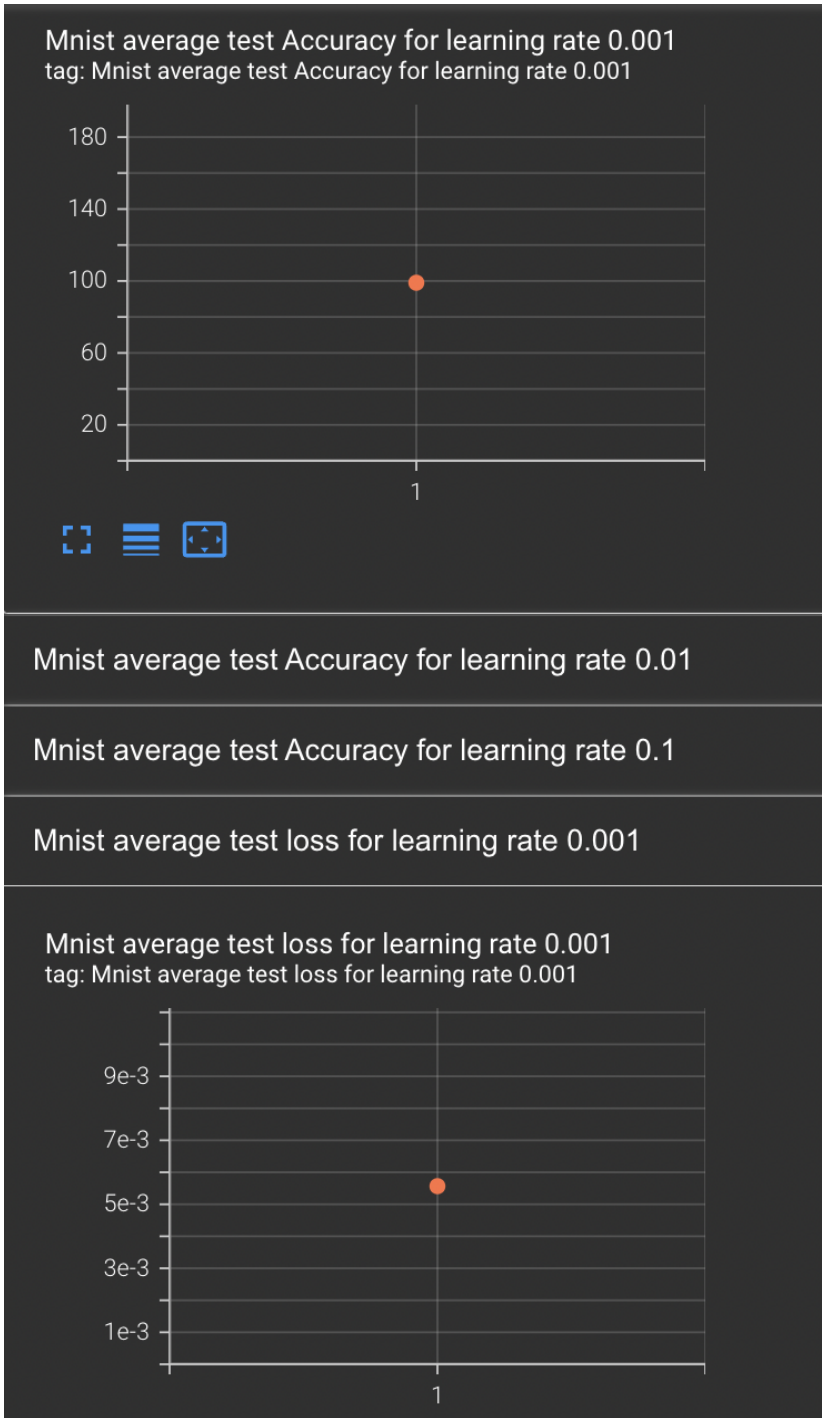


MNIST Dataset Testing losses and accuracies for learning rate 0.001

Output

```
Average test Loss: 0.0056 for learning rate 0.001
Accuracy of the network on the test images: 99.0300 % for learning rate 0.001
```

Graph in Tensorboard



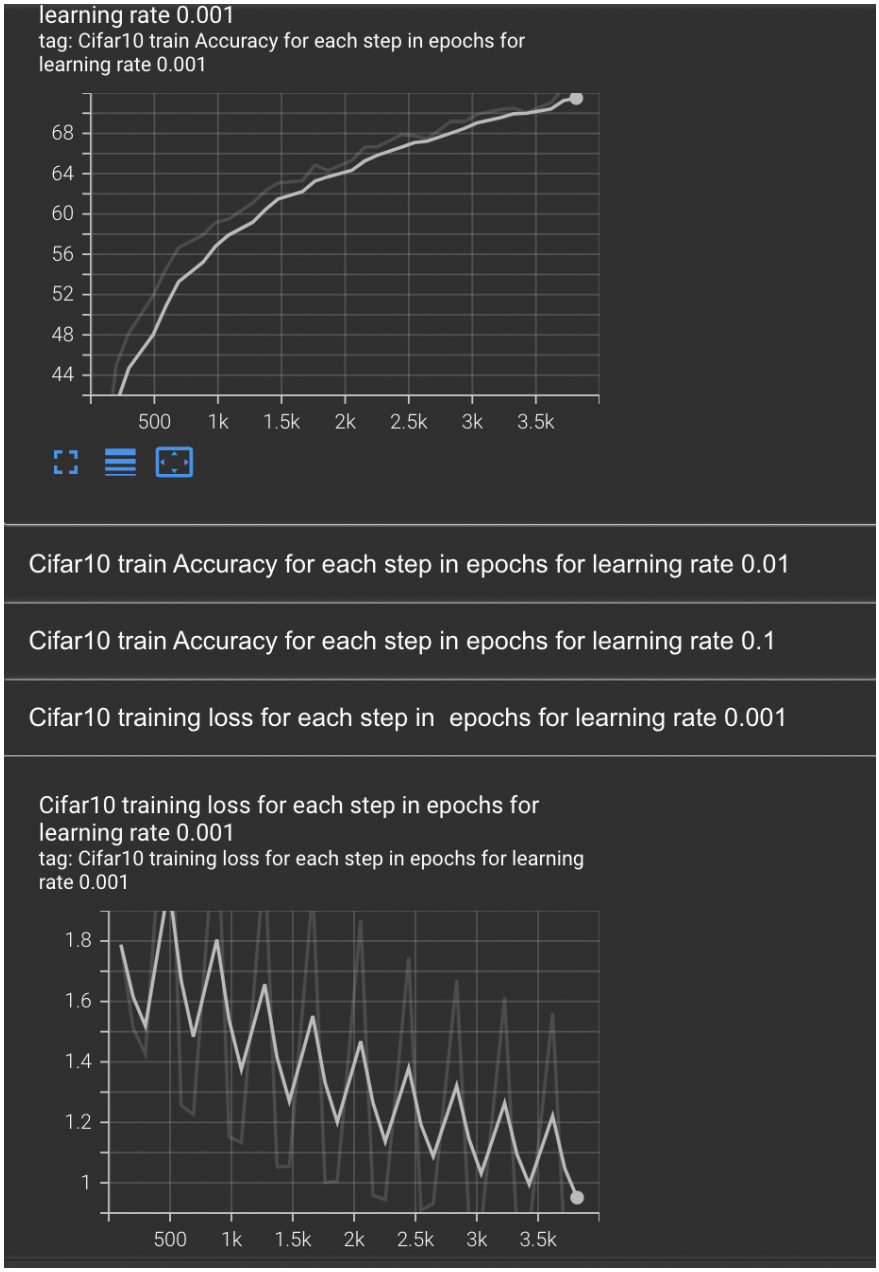
CIFAR10 Dataset Training losses and accuracies for learning rate 0.001

Output

```
Epoch [1/10], Step [100/391], Loss: 1.7879 for learning rate 0.001
Accuracy of the network on the train images: 34.5234 % for learning rate 0.001
Epoch [1/10], Step [200/391], Loss: 1.5116 for learning rate 0.001
Accuracy of the network on the train images: 44.9922 % for learning rate 0.001
Epoch [1/10], Step [300/391], Loss: 1.4261 for learning rate 0.001
Accuracy of the network on the train images: 48.1875 % for learning rate 0.001
Epoch [2/10], Step [100/391], Loss: 2.5367 for learning rate 0.001
Accuracy of the network on the train images: 51.9057 % for learning rate 0.001
Epoch [2/10], Step [200/391], Loss: 1.2564 for learning rate 0.001
Accuracy of the network on the train images: 54.5391 % for learning rate 0.001
Epoch [2/10], Step [300/391], Loss: 1.2257 for learning rate 0.001
Accuracy of the network on the train images: 56.6406 % for learning rate 0.001
```

```
Epoch [9/10], Step [100/391], Loss: 1.6132 for learning rate 0.001
Accuracy of the network on the train images: 70.3893 % for learning rate 0.001
Epoch [9/10], Step [200/391], Loss: 0.8409 for learning rate 0.001
Accuracy of the network on the train images: 70.4922 % for learning rate 0.001
Epoch [9/10], Step [300/391], Loss: 0.8461 for learning rate 0.001
Accuracy of the network on the train images: 70.0781 % for learning rate 0.001
Epoch [10/10], Step [100/391], Loss: 1.5611 for learning rate 0.001
Accuracy of the network on the train images: 71.0533 % for learning rate 0.001
Epoch [10/10], Step [200/391], Loss: 0.7845 for learning rate 0.001
Accuracy of the network on the train images: 72.5078 % for learning rate 0.001
Epoch [10/10], Step [300/391], Loss: 0.8072 for learning rate 0.001
Accuracy of the network on the train images: 71.8750 % for learning rate 0.001
```

Graph in Tensorboard

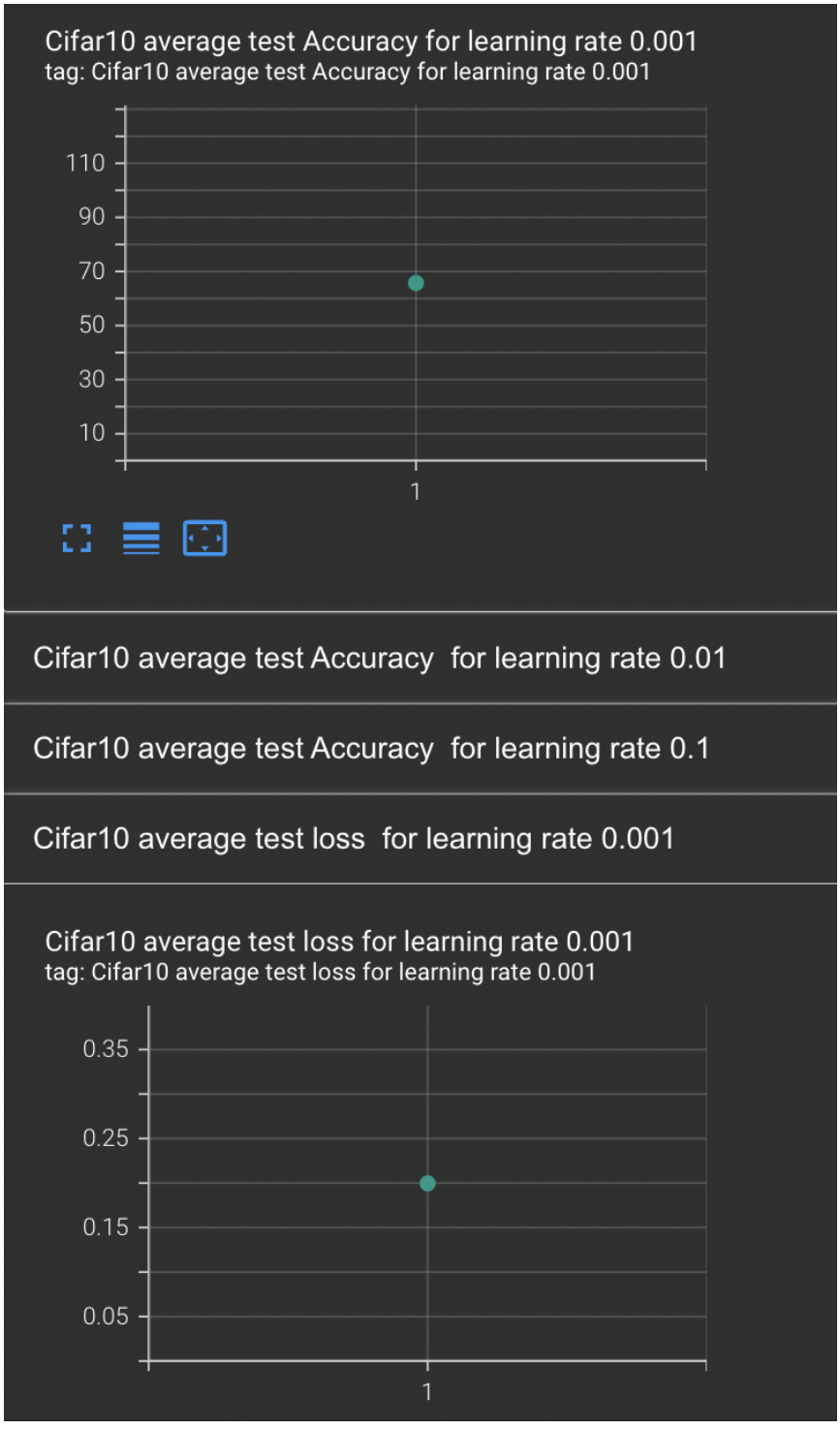


CIFAR10 Dataset Testing losses and accuracies for learning rate 0.001

Output

```
Average test Loss: 0.1997 for learning rate 0.001
Accuracy of the network on the test images: 65.7300 %
```

Graph in Tensorboard



Observations:

While training, the losses decrease as we increase the epoch. Smoother decrease in loss is observed when the learning rate is less. We also see that for MNIST, the decrease is more than for CIFAR10. In CIFAR10, spikes are also observed in between even though the trend is decreasing.

Training accuracy improves after each epoch for all datasets and all learning rates. For lesser learning rate values, the increase is smoother and higher for both datasets.

While testing, the losses are almost similar in value to the loss obtained at the last epoch of training. The test loss has not increased much. This shows that the model was not overfit. Also, loss is lesser when learning rate is lesser. Also CIFAR10 has higher losses(very less difference) than MNIST. Thus the model may be better suited for MNIST than CIFAR10.

The testing accuracy seems to be slightly less than its corresponding training accuracy. But this is expected. Since the difference in values is not much, model is suited. When learning rate is higher, the accuracy graph tends to be with spikes although overall trend is increasing.

Hence we can say that learning rate of 0.001 is best suited for these datasets and these network parameters and model is not being overfit also.

- **Exercise 2: Custom Task**

In this exercise I have extended the LeNet model used in exercise 1 to an image regression setting. Since we are using the MNIST dataset, I have resized the image and set the kernel and number of channels accordingly. Using the MNIST dataset, we design a network that will consume (N, K, C,W, H) batches of images where N, K, C,W, H are batch size, number of images to sum, channels, width and height respectively. The output of the network will be the sum of the number in the K many images.

I have taken K=3 and N=10, learning rate is 0.001 and the optimizer is Adam.

The model is adapted to accept the new dimension K using the view function. First we use collate function in the DataLoader to stack images and add new dimension K. The true target of the model is the sum of K many images in the batch. The leftover images which don't belong to the K numbered groups are dropped to avoid errors.

```
#Function to create dataset with new dimension K

def collate(batch):
    #forming the data by stacking and reshaping them to add N groups having K elements each
    data = torch.stack([img[0] for img in batch]).view(N,K,32,32)
    #forming the target label by taking sum of each group in N
    target = torch.LongTensor([img[1] for img in batch]).view(N,K).sum(axis=-1).type(torch.float)
    return [data, target]


#Loading the datasets using collate
#Here each batch would have N * K images each
train_loader = DataLoader(train_dataset_mnist, batch_size=N*K,shuffle=True,collate_fn=collate,num_workers=2,drop_last=True)
test_loader = DataLoader(test_dataset_mnist, batch_size=N*K,num_workers=2,collate_fn=collate,drop_last=True)
```

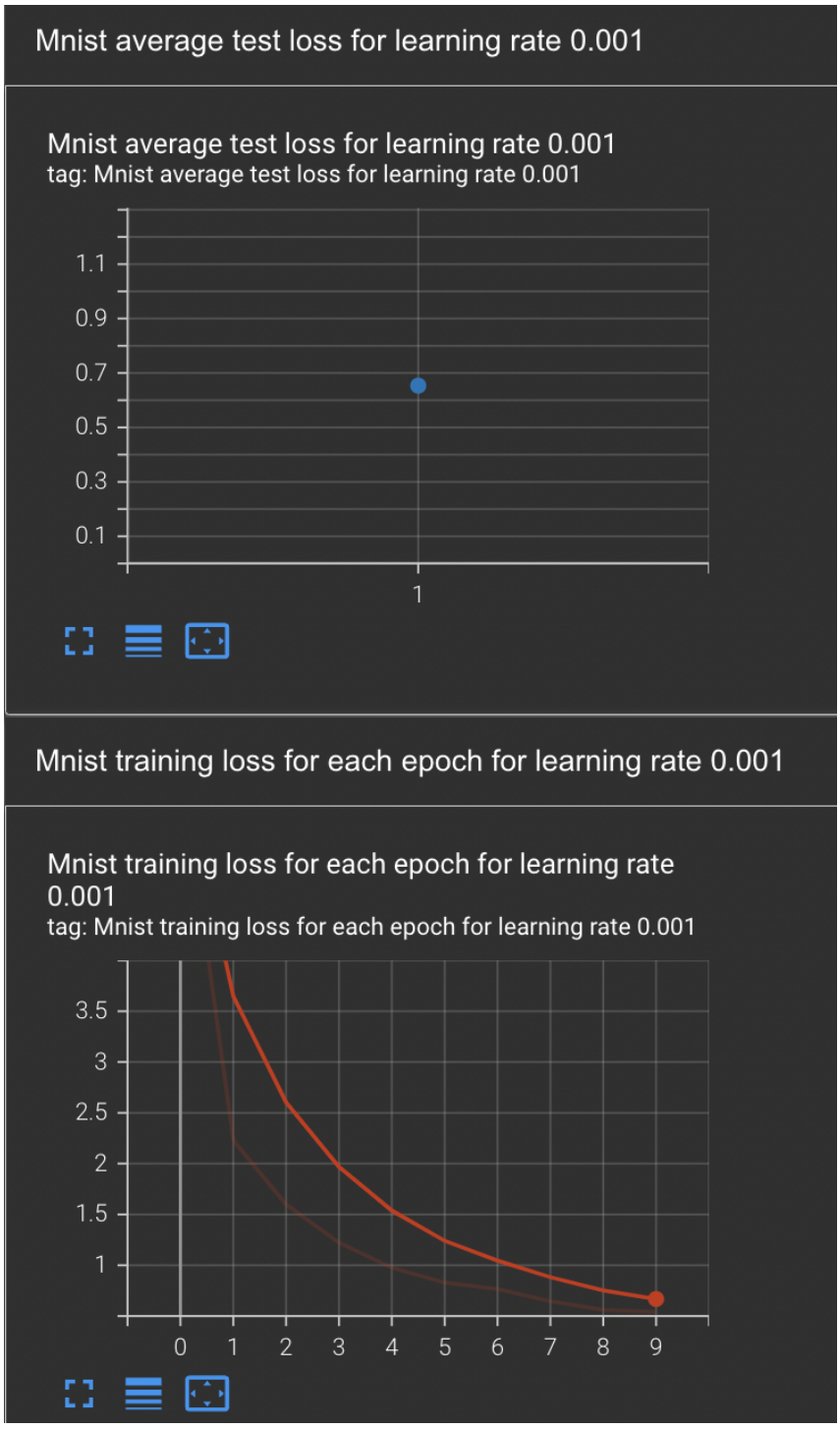
Here the error function used is RMSE. The dataset is loaded as in previous exercise using the libraries.

Since the model only accepts images of dimension (N, C, W, H), I reshaped it to (N*K,C,W,H) using the view function. Then after processing through the network, we find the sum of K consecutive images using view(-1,K) and sum(). This yields the output of the network to be the sum of the numbers in K many images.

```
def forward(self, x):
    #print("Shape of input-",x.shape)

    #Using view function as model only accepts data in (N*K,C,W,H) as given
    # in question
    x = x.view(-1,1,32,32)

    #print("Shape of input after reshaping-",x.shape)

    #Calling each layer
    out = self.layer1(x)
    out = self.layer2(out)
    #Adding the new dimension K
    x = out.view(-1,self.K,400)
    out = self.fc(x)
    out = self.relu(out)
    out = self.fc1(out)
    out = self.relu1(out)
    out = self.fc2(out)

    #Output is of size torch[10,3,1], ie 10 groups having 3 elements each as needed
    #print("Shape of out-",out.shape)

    #Finding sum of K elements i.e, each group
    x = out.view(-1,K).sum(axis=1)

    #shape of output - torch[10] - sum of each group
    #print("Shape of output-",x.shape)
    return x
```

The training is done for 10 epochs and loss is found for each epoch and it is written to tensorboard for visualising. The test error is found for the data in the test loader. The average of all losses is taken and plotted using tensor board. The code is same as done for ex 1.

Training and test loss



Output:

```
Shape of input- torch.Size([10, 3, 32, 32])
Shape of input after reshaping- torch.Size([30, 1, 32, 32])
Shape of out- torch.Size([10, 3, 1])
Shape of output- torch.Size([10])
```

Training loss outputs:

```
Epoch [1/10], Loss: 3.3309 for learning rate: 0.001
Epoch [2/10], Loss: 3.0447 for learning rate: 0.001
Epoch [3/10], Loss: 2.6780 for learning rate: 0.001
Epoch [4/10], Loss: 2.5494 for learning rate: 0.001
Epoch [5/10], Loss: 2.3752 for learning rate: 0.001
Epoch [6/10], Loss: 2.1814 for learning rate: 0.001
Epoch [7/10], Loss: 1.9443 for learning rate: 0.001
Epoch [8/10], Loss: 1.8876 for learning rate: 0.001
Epoch [9/10], Loss: 1.7949 for learning rate: 0.001
Epoch [10/10], Loss: 1.5986 for learning rate: 0.001
```

Testing loss outputs:

```
Loss: 0.6535 for learning rate 0.001
```

OBSERVATIONS:

We can see that the training losses are decreasing with the number of epochs. The testing loss is also less for this case, which shows that the model was trained correctly without overfitting. The learning rate and optimizers chosen were correct.

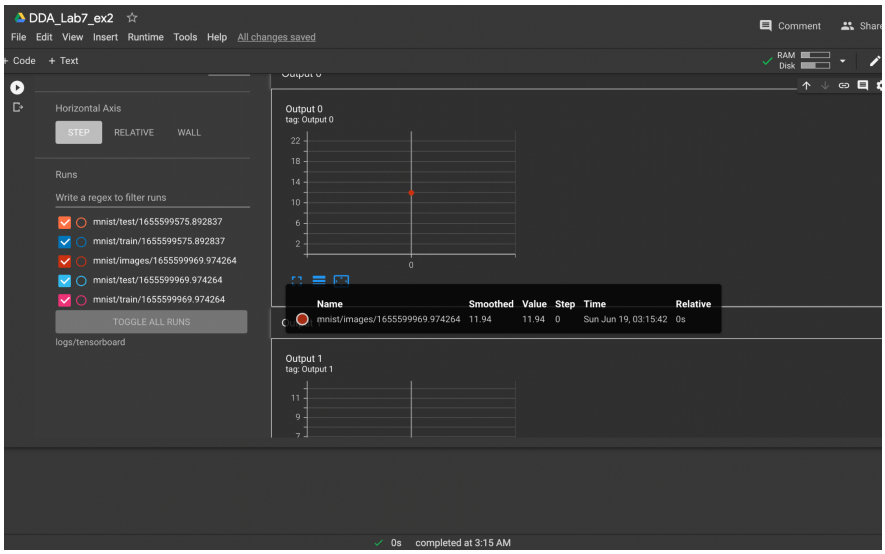
- 10 sets of test images (each set containing K many images) and the output from your trained network (final value that your model outputs)

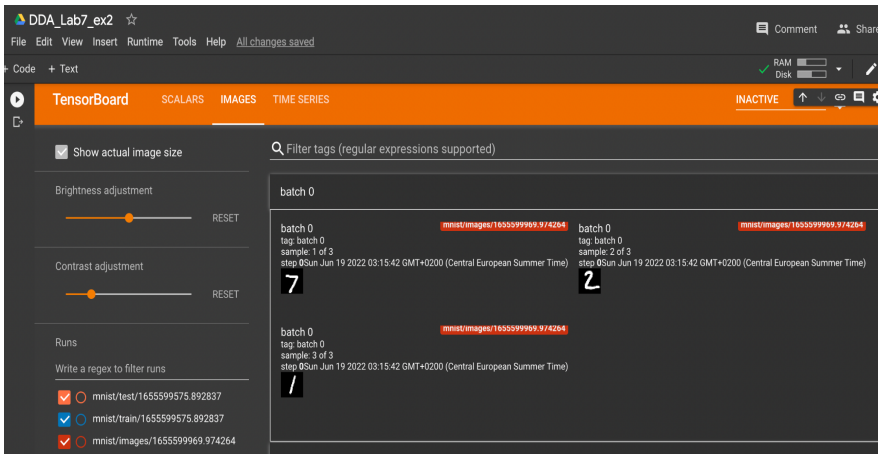
Ten sets of test images are taken from the test loader and visualised using tensorboard. The output of each set is taken after passing the image set through the network. The output values are also visualised in tensorboard.

```
#Visualising ten sets of test images, each batch having K images
#And visualising the output of the trained model on the images

images,_ = next(iter(test_loader))
images = images.to(device)
outputs = model(images)
images = images.cpu()
outputs = outputs.cpu()
outputs = outputs.detach().numpy()

for i in range(10):

  with images_summary_writer_mnist.as_default():
            tf.summary.image(f'Batch {i}', images[i].view(-1,32,32,1), step=0)
            tf.summary.scalar(f'Output {i} ', outputs[i],0)
```

Batch 0

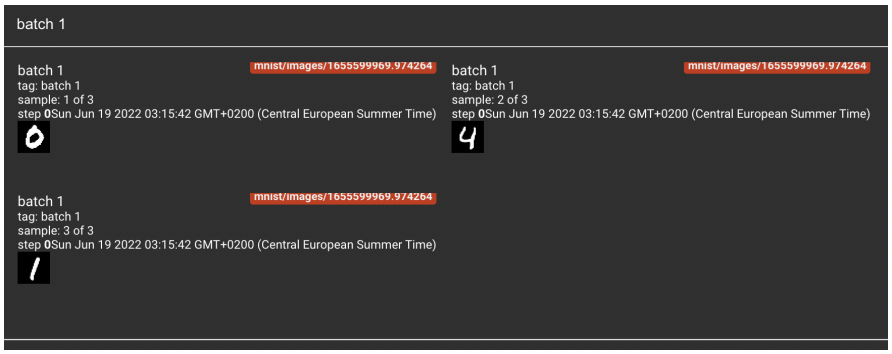
Output : 11.94 , labels of the images : 7, 2, 1

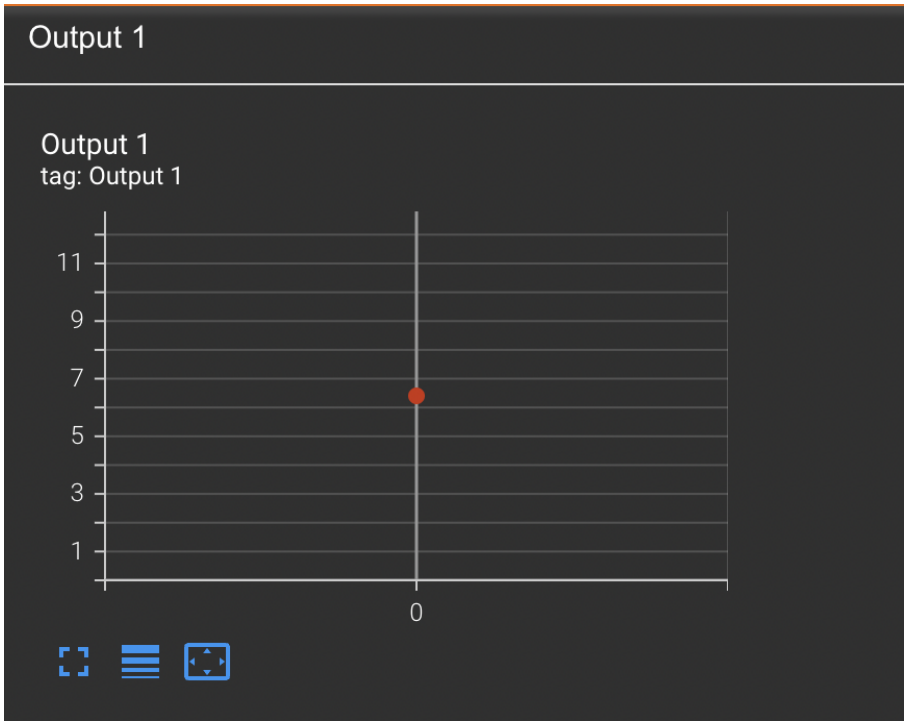




Batch 1

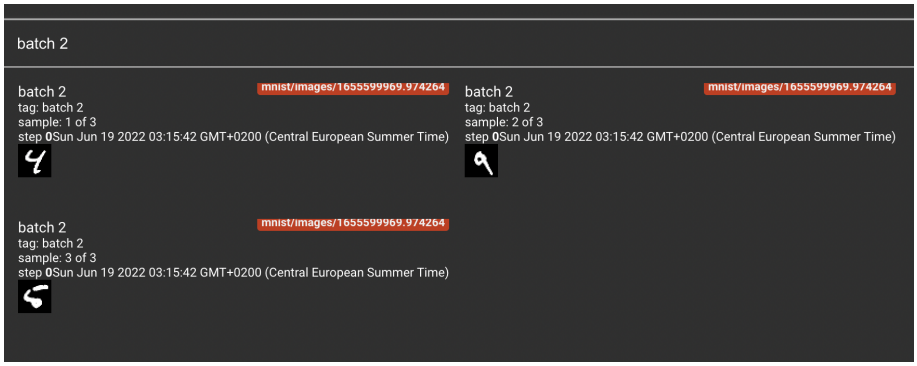
Output : 6 Labels of Images : 0, 4, 1

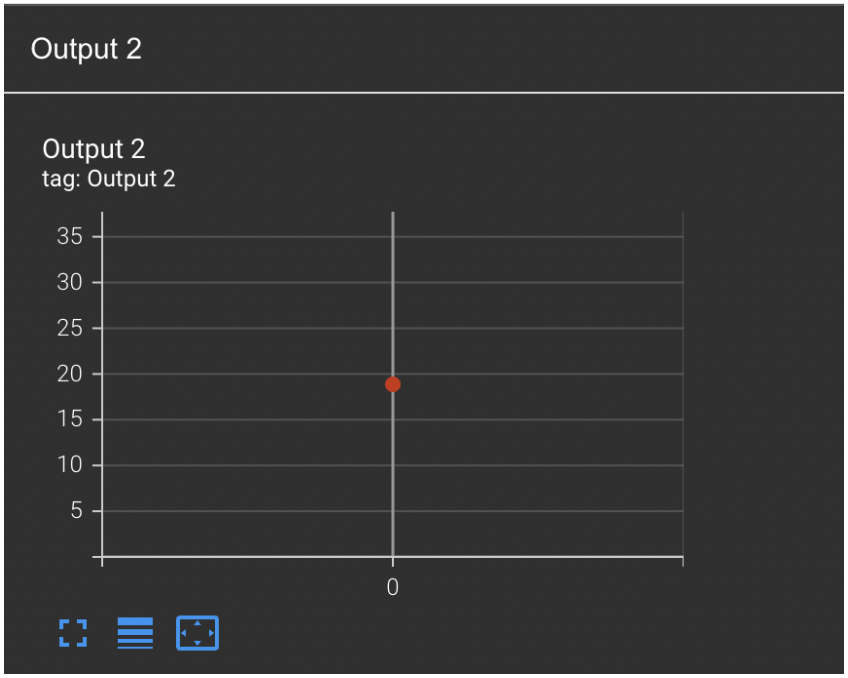




Batch 2

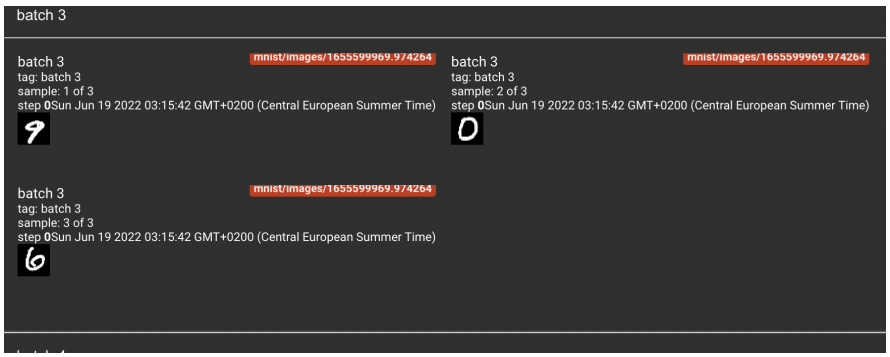
Output : 18 Labels of Images : 9, 4, 5

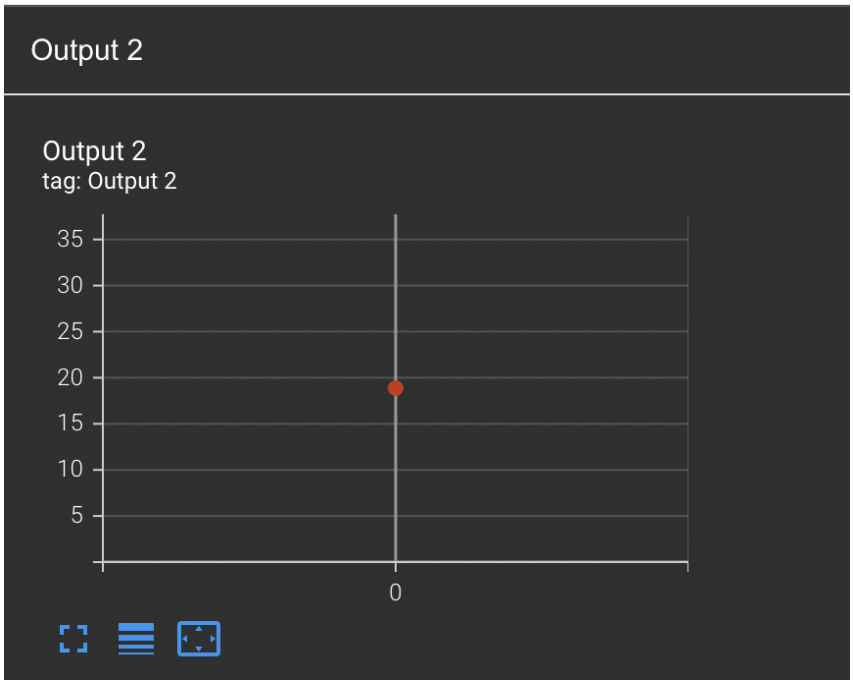




Batch 3

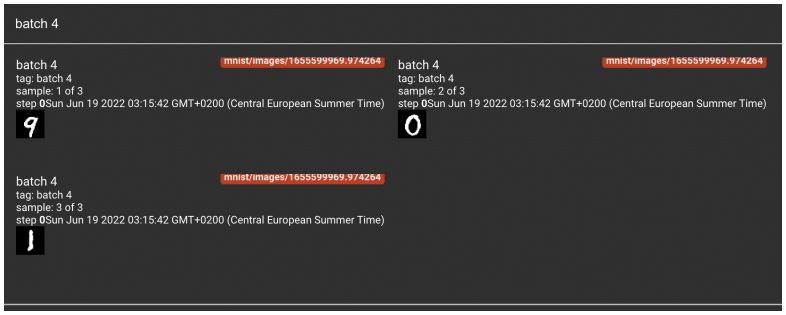
Output : 17.5 Labels of Images : 0, 6, 9

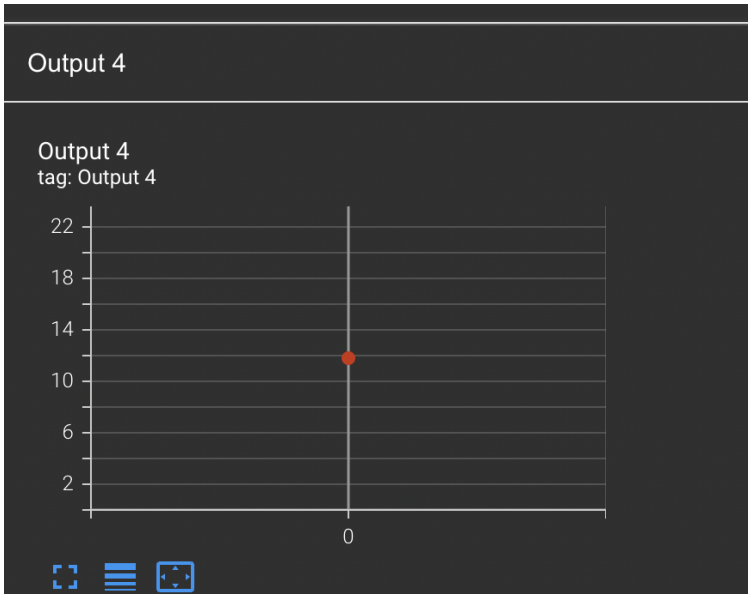




Batch 4

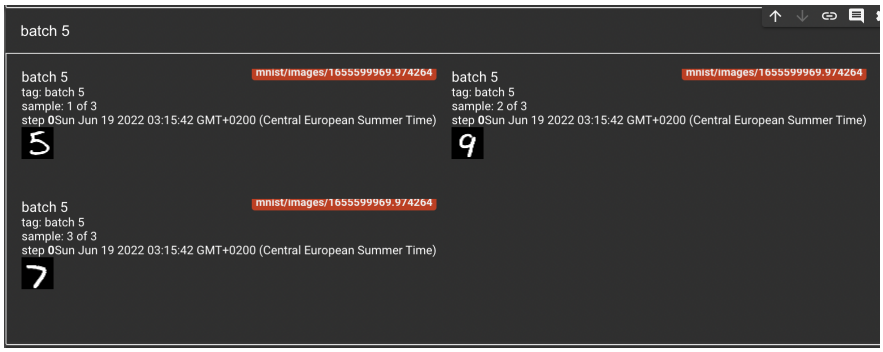
Output : 12 Labels of Images : 0, 9, 1

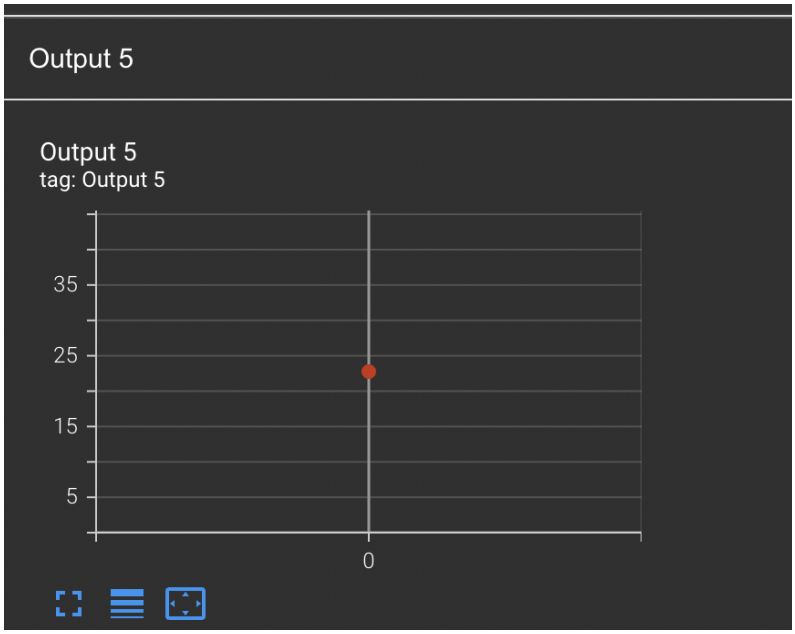




Batch 5

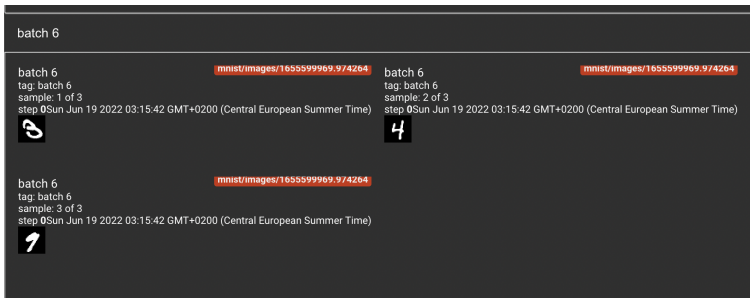
Output : 22 Labels of Images : 5, 7, 9

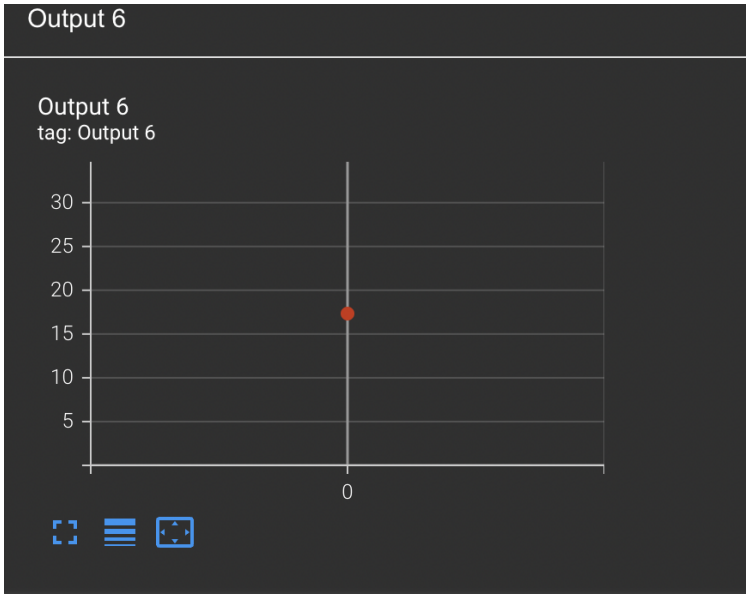




Batch 6

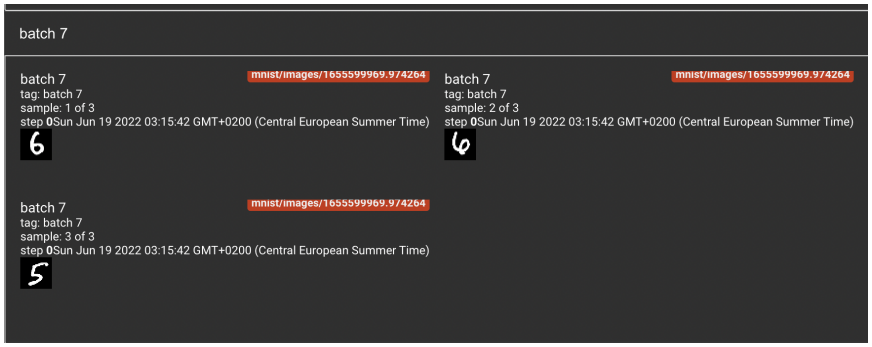
Output : 17 Labels of Images : 3, 4, 9

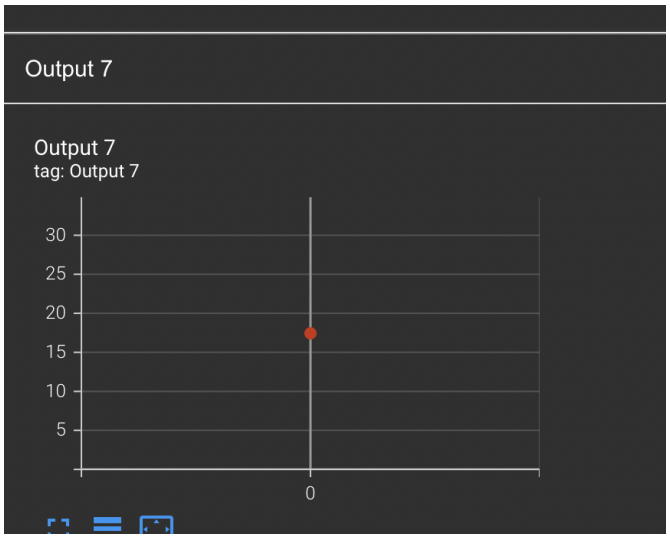




Batch 7

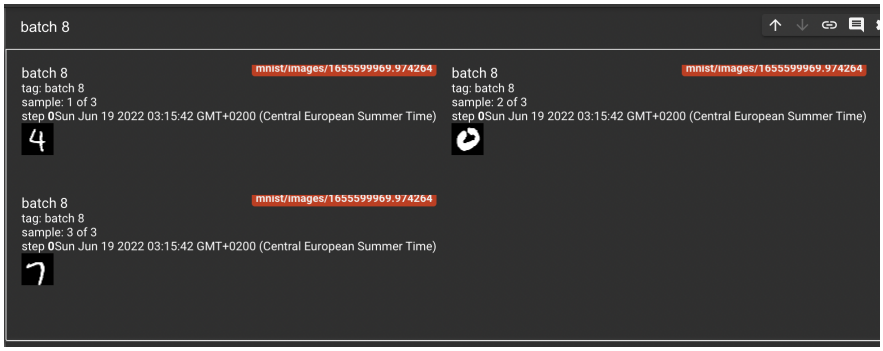
Output : 17 Labels of Images : 6, 6, 5

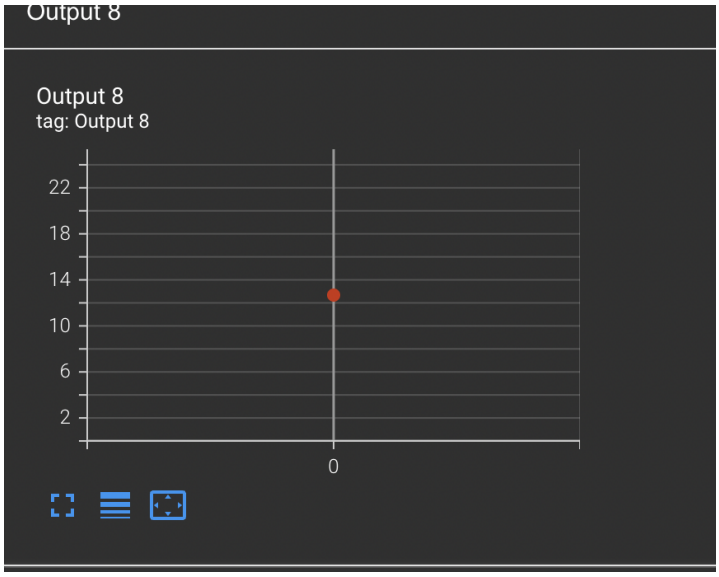




Batch 8

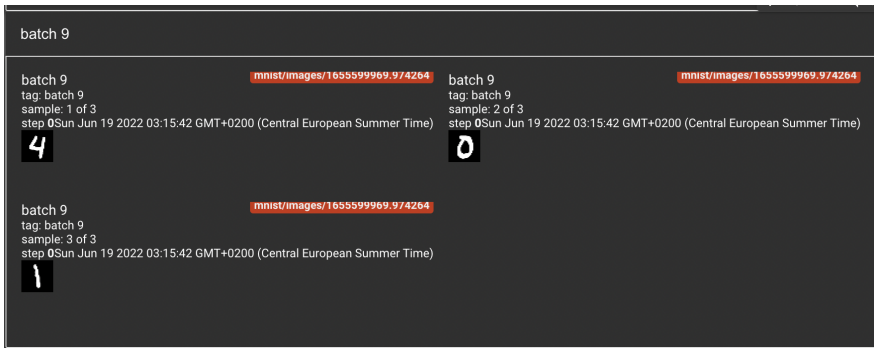
Output : 12 Labels of Images : 0, 4, 7

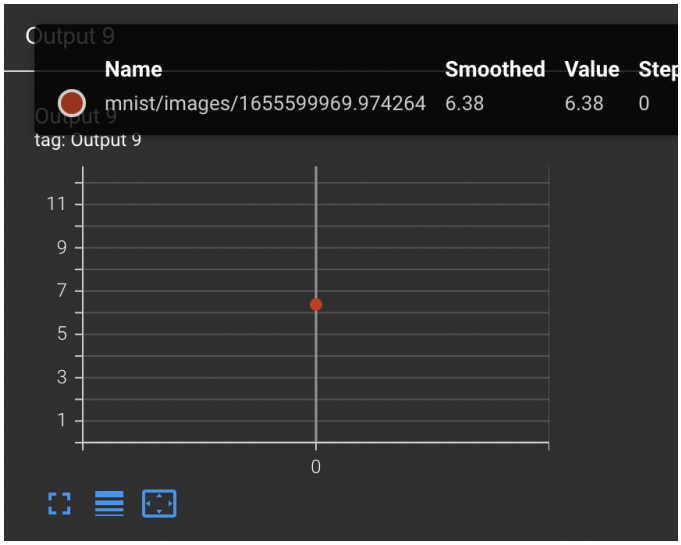




Batch 9

Output : 6.38 Labels of Images : 0, 4, 1





OBSERVATIONS:

We see that the output is almost similar to what we get on adding the numbers in each batch. There are only very slight differences. Hence we can say that the model has been trained successfully.

References:

- https://blog.paperspace.com/writing-lenet5-from-scratch-in-python/
- 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
- LeNet 1998 paper: LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.
- https://www.tensorflow.org/api_docs/python/tf/summary/SummaryWriter
- https://stackoverflow.com/questions/42479902/how-does-the-view-method-work-in-pytorch
- https://stackoverflow.com/questions/69209038/cant-call-numpy-on-tensor-that-requires-grad-use-tensor-detach-numpy-ins
- https://stackoverflow.com/questions/67248686/neural-network-learning-to-sum-two-numbers
- https://pytorch.org/docs/stable/generated/torch.Tensor.view.html