1)Cover sheet

- **Name- Nayan Shivhare**
- **Course-Advanced Intelligent Systems(CS-570)**
- **Assignment title- Torch-part1**
- **Date- 21-03-2021**

We were asked to use LeNet and Fully connected network.

Result of LeNet:-

Batch Size: 10

Learning rate : 0.001

Max Epochs: 40

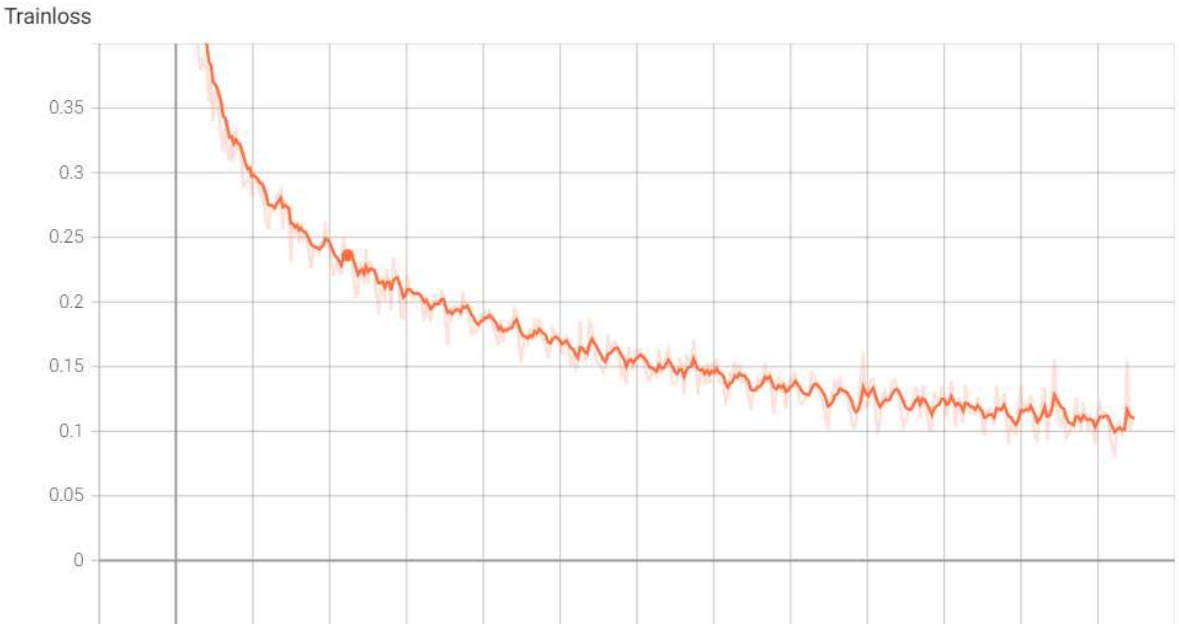
Optimizer : Adam

LeNet is lenet5, its a simple convolutional neural network proposed by Yann LeCun et al in 1998.

Training Loss: For training the in our case we were asked to used 50000 subtrain which is trained over for 40 Epochs.

I observed that the training loss of the model constantly decreasing with the increasing number of Epochs.

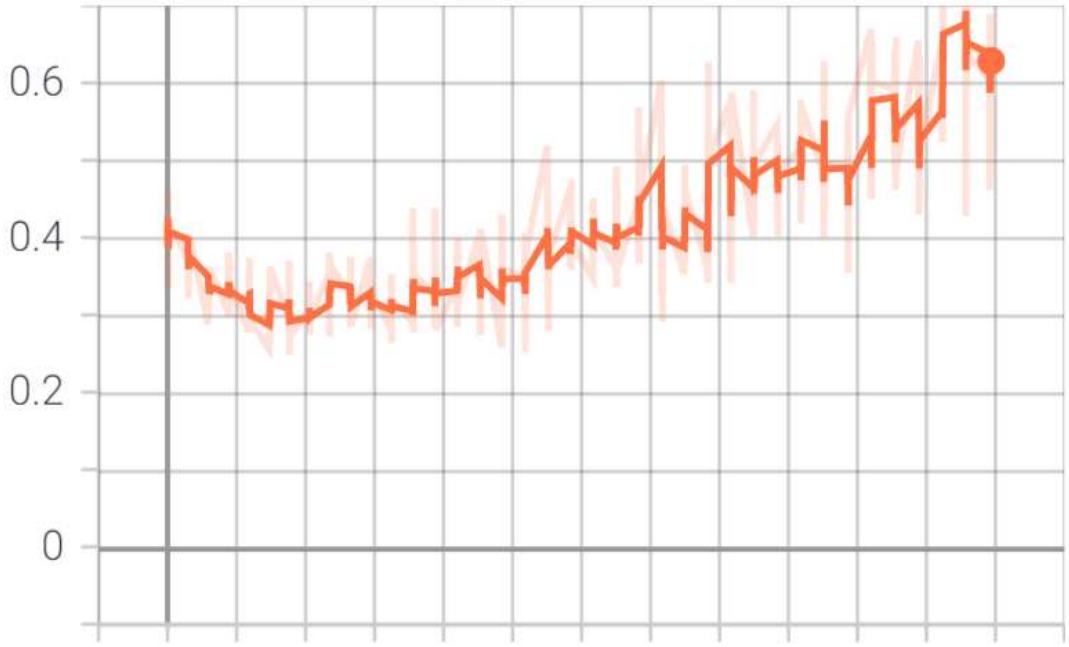
The training loss at 40 epochs is 0.1078



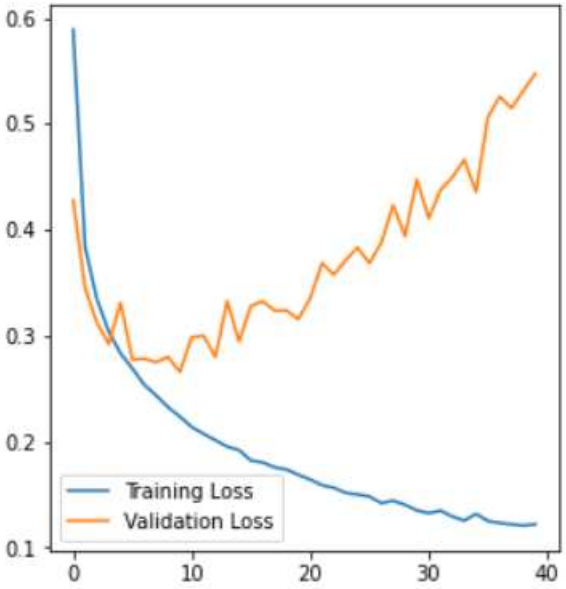
Validation Loss: I have given validation data in our case for 10000 as asked for.

As we can clearly see from the graph below that the validation loss first decreased gradually but then validation loss started increasing which shows that it is a case of overfitting. It also tells that our model is performing well on training set but in case of validation it is overfitting.

Validation loss after 40 epochs is 0.6281.



To prevent the model from overfitting we can add some dropout layers in our model. We can try different architectures and hyper parameters. We can change learning rate , optimizers , batch etc.



Accuracy of model after 40 Epochs : 89%.

Result of Fully Connected Neural Network Results.

Batch Size: 10

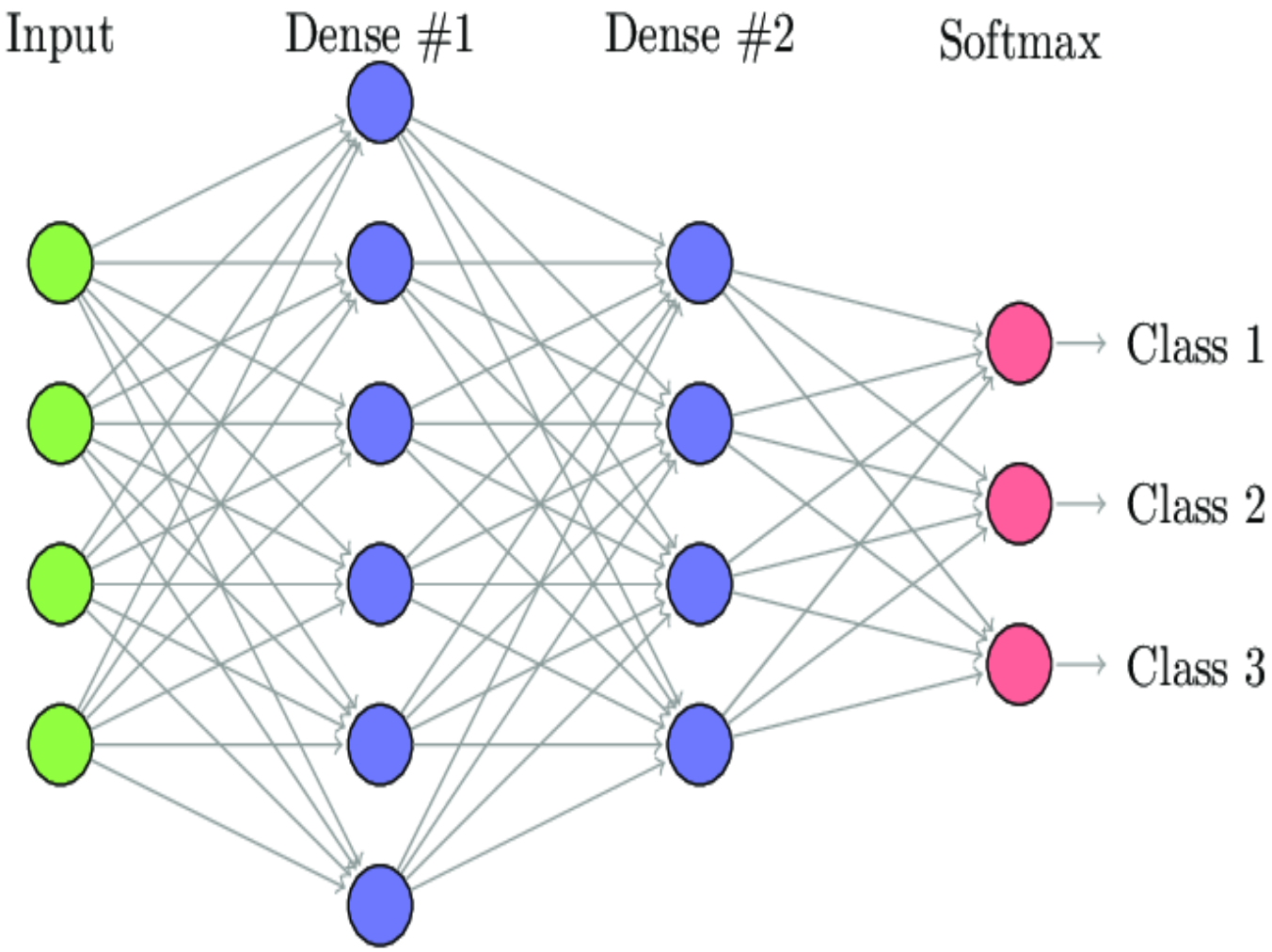
Optimizer: SGD

Learning rate: 0.001

Max Epoch: 50

Optimizer: SGD

Fully connected neural networks (FCNNs) are a type of artificial **neural network** where the architecture is such that all the nodes, or neurons, in one layer are **connected** to the neurons in the next layer.

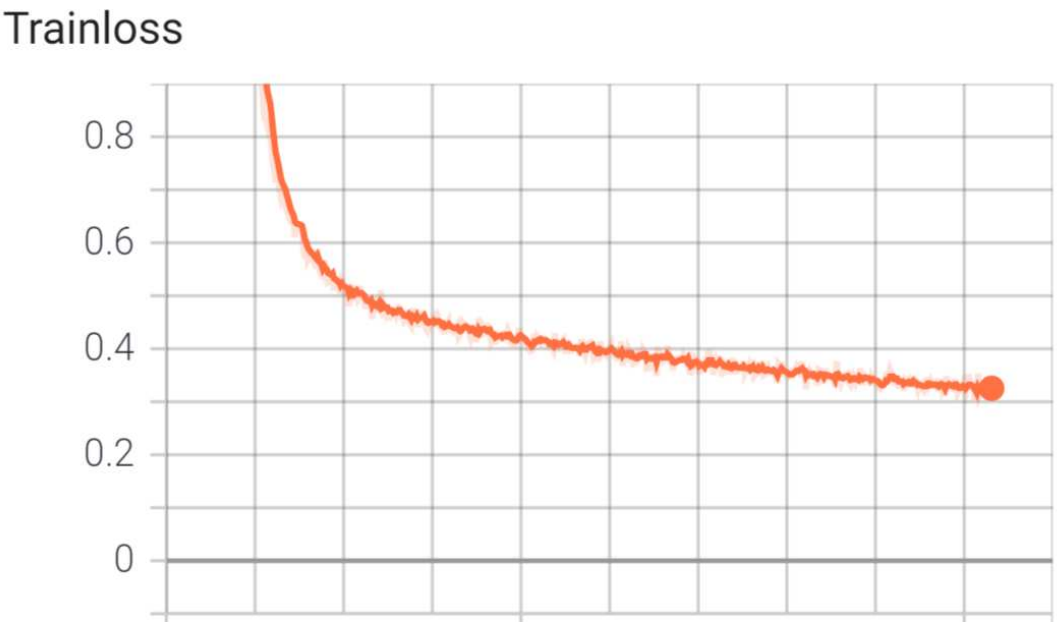


Training Loss: For training in our case we were asked to used 50000 subtrain which is trained over for 50 Epochs.

and we can see

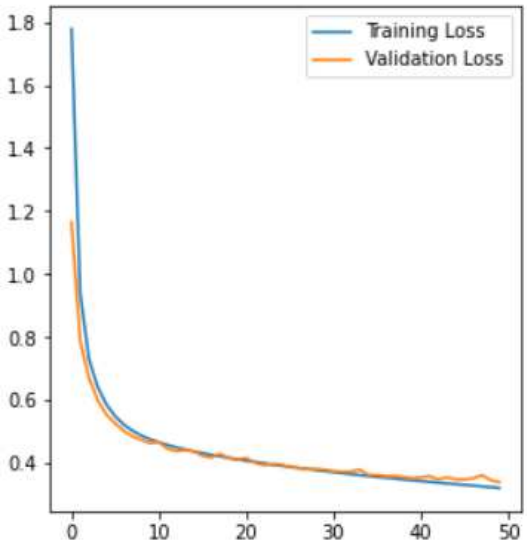
I observed that the training loss of the model constantly decreasing with the increasing number of Epochs.

The training loss at 50 epochs is 0.3202

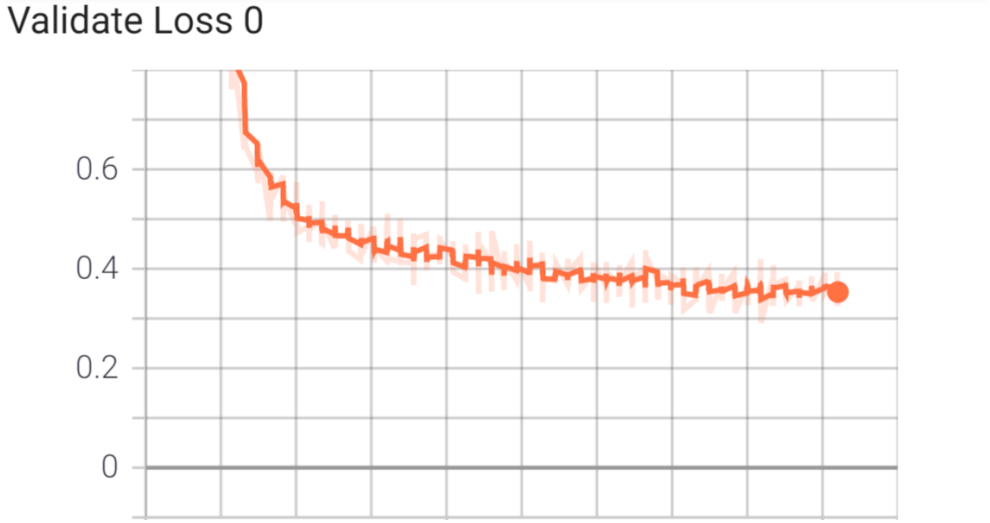


Validation Loss: I have given validation data in our case for 10000 as asked for.

As we can clearly see from the graph below validation loss is constantly decreasing which is supposed to be a good validation loss. There is no case of overfitting and Under fitting when we use fully connected neural network with SGD as optimizer.



Tensor Board Visualization:



The Accuracy of fully connected neural network after 50 epochs is 87%

CODE:

```
"""IMPORTING USEFUL LIBS"""
import torchvision
import torch
import torchvision.transforms as transforms
import numpy as np
import matplotlib.pyplot as plt
import torch.optim as optim
import time
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.tensorboard import SummaryWriter # Using tensorboard for the display of losses
```

```
from torchvision.transforms import ToTensor, Lambda
"""Importing the dataset from the inbuilt datsets from torchvision library.
it includes many dataset such as Mnist, Fakedata, Fashion mnist etc."""
#we are not coneverting the output to one hot encoded values as because CrossEntropyLoss does not support the one hot encoded values
# as target to calculatte the loss.We are transforming the dataset to pytorch tensors
ds = datasets.FashionMNIST(
    root="data",
    train=True,
    download=True,
    transform=ToTensor()) #using train=True as we wanted the train dataset only
```

```
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz to data/FashionMNIST/raw/train-images-idx3-ubyte.gz
26422272/? [00:03<00:00, 7300184.90it/s]

Extracting data/FashionMNIST/raw/train-images-idx3-ubyte.gz to data/FashionMNIST/raw
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz to data/FashionMNIST/raw/train-labels-idx1-ubyte.gz
29696/? [00:01<00:00, 23947.51it/s]

Extracting data/FashionMNIST/raw/train-labels-idx1-ubyte.gz to data/FashionMNIST/raw
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz to data/FashionMNIST/raw/t10k-images-idx3-ubyte.gz
4422656/? [00:00<00:00, 6095893.94it/s]
```

```
loss_function = torch.nn.CrossEntropyLoss() #we are using cross entropy as loss function

results = {}
from torch.utils.tensorboard import SummaryWriter #summry writer to write the logs i.e. loss accuracy to our dashboard

# create a summary writer with automatically generated folder name.
writer = SummaryWriter("resultsSGD") #Instantiated a object
```

```
"""LeNet refers to lenet-5, is a simple convolutional neural network proposed by Yann LeCun et al in 1998"""
class LeNet(nn.Module):
    def __init__(self):
      super().__init__()
      self.conv1 = nn.Conv2d(in_channels=1, out_channels= 6, kernel_size=5) #convolution layer with 1 input and 6 output with a kerner size of 5
      self.conv2 = nn.Conv2d(6, 16, 5) #convolution layer with 6 input and 16 output
      self.fc1 = nn.Linear(4*4*16, 120)#linear layer with 256 input and 120 output
      self.fc2 = nn.Linear(120, 84)
      self.output = nn.Linear(84, 10)# liner layer with 84 input and 10 output that is classes of fashion mnist
    def forward(self, x): #forward function to process the input data though the network
      x = F.relu(self.conv1(x))
      # use x.shape to check the current size
      # print (x.shape)
      x = F.max_pool2d(x, 2, 2)
      x = F.relu(self.conv2(x))
      x = F.max_pool2d(x, 2, 2)
      x = x.view(-1, 4*4*16)
      x = F.relu(self.fc1(x))
      x = F.relu(self.fc2(x))
      x = self.output(x)
      return x #returnning the output
```

```python
#Fully Connected deep neural network
class Model(nn.Module):
    def __init__(self, input_size, h1, h2, output_size):
        super().__init__()
        self.layer_1 = nn.Linear(input_size, h1)
        self.layer_2 = nn.Linear(h1, h2)
        self.layer_3 = nn.Linear(h2, output_size)
    def forward(self, x):
        x = F.relu(self.layer_1(x))
        x = F.relu(self.layer_2(x))
        x = self.layer_3(x)
        return x
```

```python
from sklearn.model_selection import KFold
#using Cross Validation with N_Fold=6
n_folds =6
kfold = KFold(n_splits=n_folds, shuffle=True)
```

Training and Validation for LeNet:

```python
loss_keeper2={'train':[],'valid':[]}
for fold, (train_ids, test_ids) in enumerate(kfold.split(ds)): #Looping though our data to train the model
    print(train_ids.shape)
    if fold==1:
      break
    train_subsampler = torch.utils.data.SubsetRandomSampler(train_ids)#sampling element randomly frm the given id that we split using kFolds
    test_subsampler = torch.utils.data.SubsetRandomSampler(test_ids)#sampling element randomly frm the given id that we split using kFolds

    trainloader = torch.utils.data.DataLoader(
                      ds,
                      batch_size=10, sampler=train_subsampler)
    testloader = torch.utils.data.DataLoader(
                      ds,
                      batch_size=10, sampler=test_subsampler) #Pytorch Data loader to load the data, given batch size 10 as hyperparametere and the random ids
    neuralnetwork = LeNet()#instantiate our LeNet Neural Network
    optimizer = torch.optim.Adam(neuralnetwork.parameters(), lr=0.001) #using ADAM as the optimizer to optimize loss
    for epoch in range(0,100): #running over 20 epochs
      train_loss=0.0
      valid_loss=0.0
      print(f'Starting epoch {epoch+1}')
      current_loss = 0.0 #set loss to 0 inititally
      current_loss_val=0.0 #validation loss to 0 initially
      for i, data in enumerate(trainloader, 0): #iterate over our 50000 Training data
        images, labels = data #getting data and label
        # print(labels)
        # print(images.shape)  #print for debugging of code
        # print(labels.shape)
```

```
  # print(labels)
  loss = loss_function(outputs, labels) #Computing Loss using the real and predicted data
  loss.backward()
  train_loss+=loss.item()
  optimizer.step()#optimize our loss
  current_loss += loss.item()
  if i % 500 == 499:#priting the statistic of loss after every 500 minibatch processed
      # print(current_loss / 500)
      # print(epoch)
      # print(f'Loss after {i + 1}:  {current_loss / 500}')
      print('Loss after %5d: %.3f' %
            (i + 1, current_loss / 500))
      writer.add_scalar(f"Trainloss",(current_loss / 500), epoch)#writing our loss to our tensorboard
      # print(current_loss / 500)
      current_loss = 0.0
      # writer.add_scalar(network._get_name()+'/loss', current_loss, epoch)

print('Validating')
correct, total = 0, 0
with torch.no_grad():
  for i, data in enumerate(testloader, 0):#loop through our validation data that is 10000 in our case
    images, labels = data #getting label and input images
  # Generate outputs
    outputs = neuralnetwork(images)#inout images to our network to get the prediction
    loss = loss_function(outputs, labels)#calculate loss using cross entropy to get validation loss
    valid_loss+=loss.item()
    current_loss_val += loss.item()
```

```
      # print(current_loss_val)
        if i % 100 == 99:#printing Stats on every 99 epochs
          print('Loss Validate %5d: %.3f' %(i + 1, current_loss_val / 99))
          writer.add_scalar(f"Validate Loss {fold}",(current_loss_val / 99), epoch)
          current_loss_val = 0.0
        _, predicted = torch.max(outputs.data, 1)#get the prediction of images
        total += labels.size(0)
        correct += (predicted == labels).sum().item()#calculated correct images to get the accuracy of model
      print(f'Accuracy {100.0 * correct / total}')
      results[fold] = 100.0 * (correct / total)#keep the result for other cross validation folds
      loss_keeper2['train'].append(train_loss)#keep track of our train and validate loss for genrerating graphs through matplotlib
      loss_keeper2['valid'].append(valid_loss)

sum = 0.0
for key, value in results.items():
    sum += value
print(f'Average: {sum/len(results.items())} %')
writer.close()
```

Training and Validation for Fully Connected

In [17]:

```
loss_keeper2={'train':[],'valid':[]}
for fold, (train_ids, test_ids) in enumerate(kfold.split(ds)): #Looping though our data to train the model
    print(train_ids.shape)
    if fold==1:
      break
    train_subsampler = torch.utils.data.SubsetRandomSampler(train_ids)#sampling element randomly frm the given id that we split
    test_subsampler = torch.utils.data.SubsetRandomSampler(test_ids)#sampling element randomly frm the given id that we split us

    trainloader = torch.utils.data.DataLoader(
                      ds,
                      batch_size=10, sampler=train_subsampler)
    testloader = torch.utils.data.DataLoader(
                      ds,
                      batch_size=10, sampler=test_subsampler) #Pytorch Data loader to load the data, given batch size 10 as hype

    """The first layer we have 784 nodes,
     because we have 28*28 = 784 pixels.
     Each pixel represents a feature. So our input_size = 784
We have 10 nodes for the last layer because we have 10 different categories from 0 to 9. So output_size = 10"""
    network = Model(784, 300, 100, 10)#instantiate our Fully Connected neural network Neural Network
    optimizer = torch.optim.SGD(network.parameters(), lr=0.001) #using SGD as the optimizer to optimize loss
    for epoch in range(0,50): #running over 20 epochs
      train_loss=0.0 #to track training loss
      valid_loss=0.0 # to track validate loss
      print(f'Starting epoch {epoch+1}')
      current_loss = 0.0 #set loss to 0 inititally
      current_loss_val=0.0 #validation loss to 0 initially
      for i, data in enumerate(trainloader, 0): #iterate over our 50000 Training data
        images, label = data #getting data and label
        images = data[0].view(data[0].shape[0], -1)
        # print(label)
        # print(images.shape)  #print for debugging of code
        # print(label.shape)
        optimizer.zero_grad()
        pred = network(images) #pass our input data to the network to get the result
```

```
        # print(images.shape)  #print for debugging of code
        # print(label.shape)
        optimizer.zero_grad()
        pred = network(images) #pass our input data to the network to get the result
        # print(label)
        # label=label.squeeze_()
        # print("Tagrets",label.shape)
        # print(label)
        loss = loss_function(pred, label) #Computing Loss using the real and predicted data
        loss.backward()
        train_loss+=loss.item()
        optimizer.step()#optimize our loss
        current_loss += loss.item()
        if i % 500 == 499:#priting the statistic of loss after every 500 minibatch processed
            # print(current_loss / 500)
            # print(epoch)
            # print(f'Loss after {i + 1}:  {current_loss / 500}')
            print('Loss after %5d: %.3f' %
                  (i + 1, current_loss / 500))
            writer.add_scalar(f"Trainloss",(current_loss / 500), epoch)#writing our loss to our tensorboard
            # print(current_loss / 500)
            current_loss = 0.0
            # writer.add_scalar(network._get_name()+'/Loss', current_loss, epoch)

    correct, total = 0, 0
    with torch.no_grad(): #with no gradients
      for i, data in enumerate(testloader, 0):#Loop through our validation data that is 10000 in our case
        images, label = data # images and label of our validate data
        images = data[0].view(data[0].shape[0], -1) #convertint he input to the shape that we given at the time of creating th
        pred = network(images)#get the predictions
        loss = loss_function(pred, label)#calcualte loss using cross entropy
        valid_loss+=loss.item()
        current_loss_val += loss.item()
        if i % 100 == 99:
          print('Loss Validate %5d: %.3f' %(i + 1, current_loss_val / 99))#print for stats
          writer.add_scalar(f"Validate Loss {fold}",(current_loss_val / 99), epoch) #add to our tensorboard
```

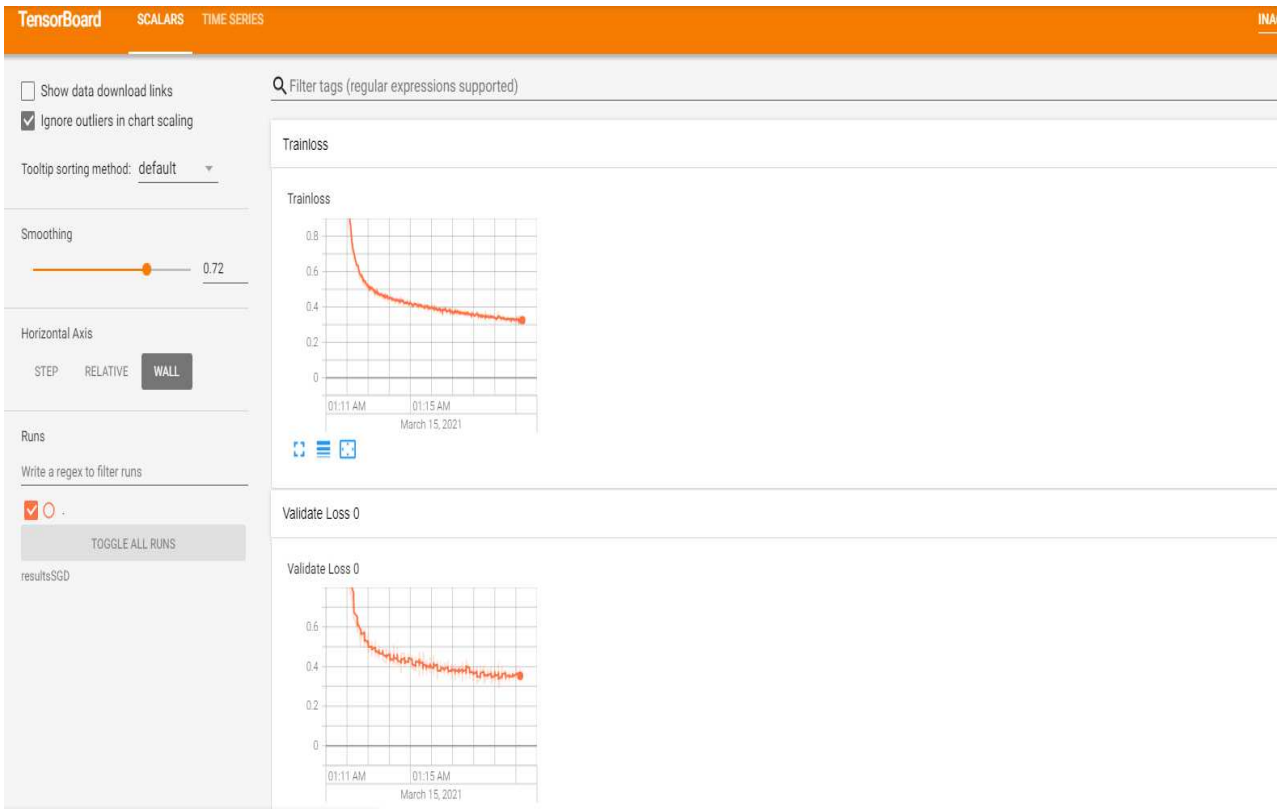
```
          print('Loss Validate %5d: %.3f' %(i + 1, current_loss_val / 99))#print for stats
          writer.add_scalar(f"Validate Loss {fold}",(current_loss_val / 99), epoch) #add to our tensorboard
          current_loss_val = 0.0
        _, predicted = torch.max(pred.data, 1)#getting the data predicted data
        total += label.size(0)
        correct += (predicted == label).sum().item()#calculating total correct data

      print(f'Accuracy {100.0 * correct / total}')
      results[fold] = 100.0 * (correct / total)
      train_loss = train_loss/len(trainloader)
      valid_loss = valid_loss/len(testloader)
      loss_keeper2['train'].append(train_loss)#track of trainloss
      loss_keeper2['valid'].append(valid_loss)#track of validtion loss for later plotting
sum = 0.0
for key, value in results.items():
    sum += value
print(f'Average: {sum/len(results.items())} %')#average of all folds
writer.close()
```

Tensorboard:-

```
%load_ext tensorboard
```

```
%tensorboard --logdir resultsSGD
```



```
fig=plt.figure(1,figsize=(10,5))
idx=1

ax=fig.add_subplot(1,2,idx)
ax.plot(loss_keeper2['train'],label="Training Loss")
ax.plot(loss_keeper2['valid'],label="Validation Loss")

idx+=1
plt.legend();
```