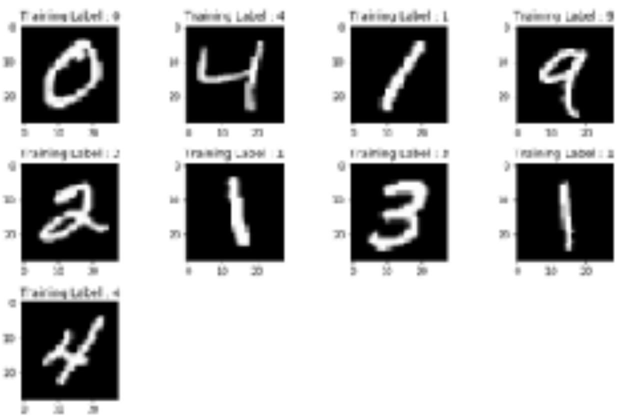
CLASSIFICATION OF MNIST AND FASHION MNIST DATA USING MUTLI-LAYER PERCEPTRON AND CONVOLUTION NEURAL NETWORK

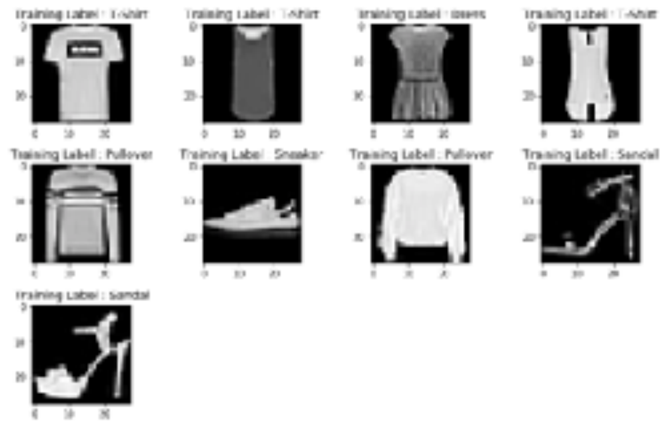
Shivangi Gupta
A25266618
CS696-Big Data Analytics

TABLE OF CONTENTS

INTRODUCTION

In my project, I have used MNIST dataset which comprises of hand-written digits and Fashion MNIST which comprises of clothing images. Both the dataset are of the same dimension and size i.e. 28X28 and 70,000 images. The images shown below depicts the sample images of MNIST and Fashion MNIST dataset.





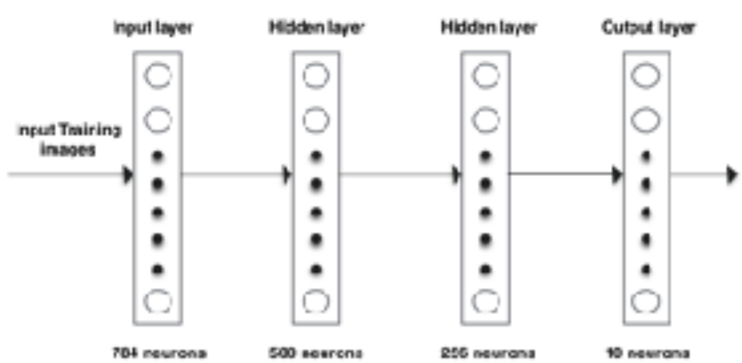
MNIST Dataset **Fashion MNIST Dataset**

To Classify the images, I have used Multi-Layer perceptron and Convolution Neural Network.

METHODS

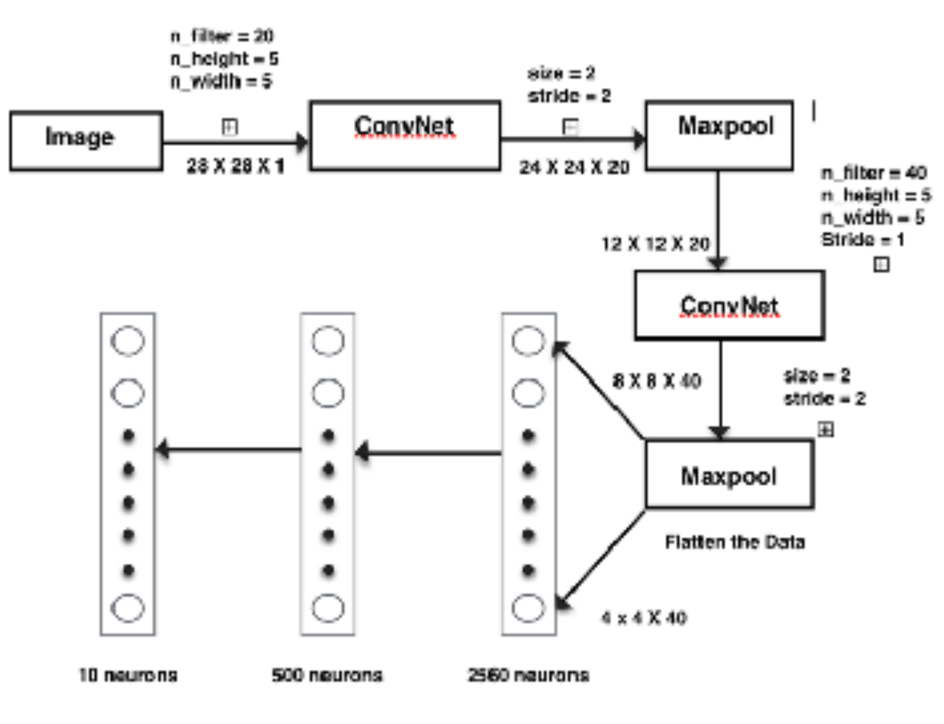
MULTI-LAYER PERCEPTRON

It is a class of Feed-forward network. It consists of at least three Hidden layers of nodes. Each node is a neuron that uses a non-linear activation function. The figure given below shows the Architecture of the Multi-Layer perceptron used for this project.



CONVOLUTIONAL NEURAL NETWORK

It is a class of deep, feed-forward network. They require relatively less pre-processing compared to other image classification algorithms. The figure given below shows the Architecture of the Convolution neural network used for this project.



At Each layer except the output Layer, I have used ReLu(Rectified Linear Unit) as the Non-linear Activation Function. Softmax is applied at the output layer and Cross Entropy Loss is used to calculate the loss at each epoch.

ReLu Function:

f(x) = max(0, x)

Softmax Function:

$$P(y = j \mid \mathbf{x}) = \frac{e^{\mathbf{x}^\mathsf{T}\mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\mathsf{T}\mathbf{w}_k}}$$

Cross Entropy Loss Function:

Loss = -(y * log(p) + (1 - y) * log(1 - p))

RESULTS AND DISCUSSIONS

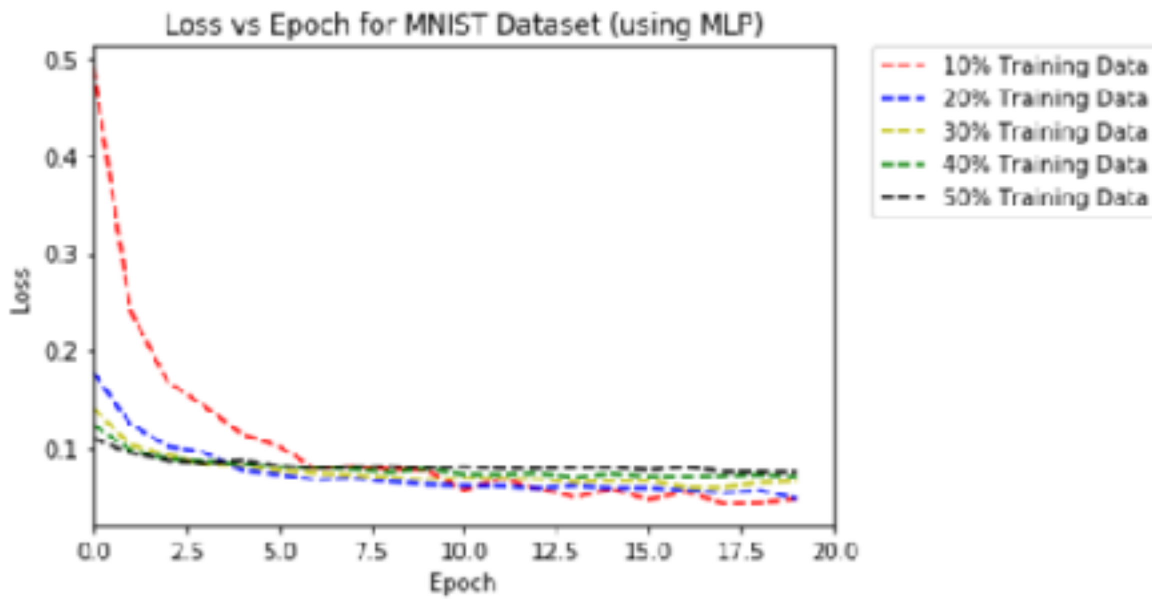


Figure 1 : The Image above shows the Loss versus Epoch graph for MNIST dataset using Multi-Layer Perceptron Model for different Training Data Sizes.

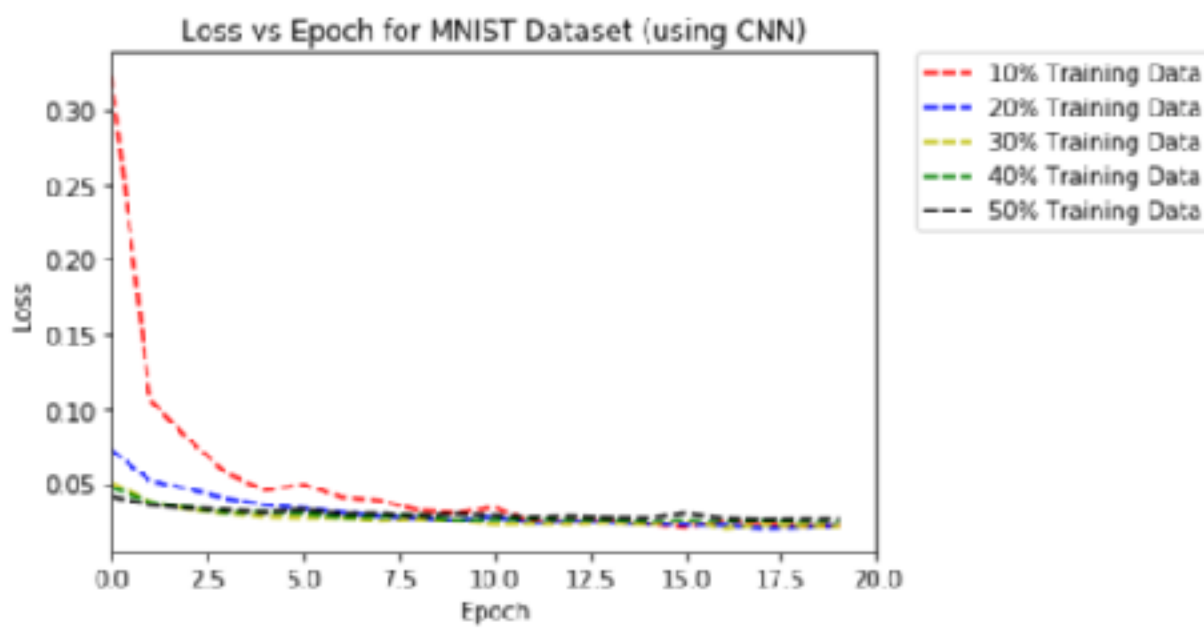


Figure 2 : The Image above shows the Loss versus Epoch graph for MNIST dataset using Convolutional Neural Network for different Training Data Sizes.

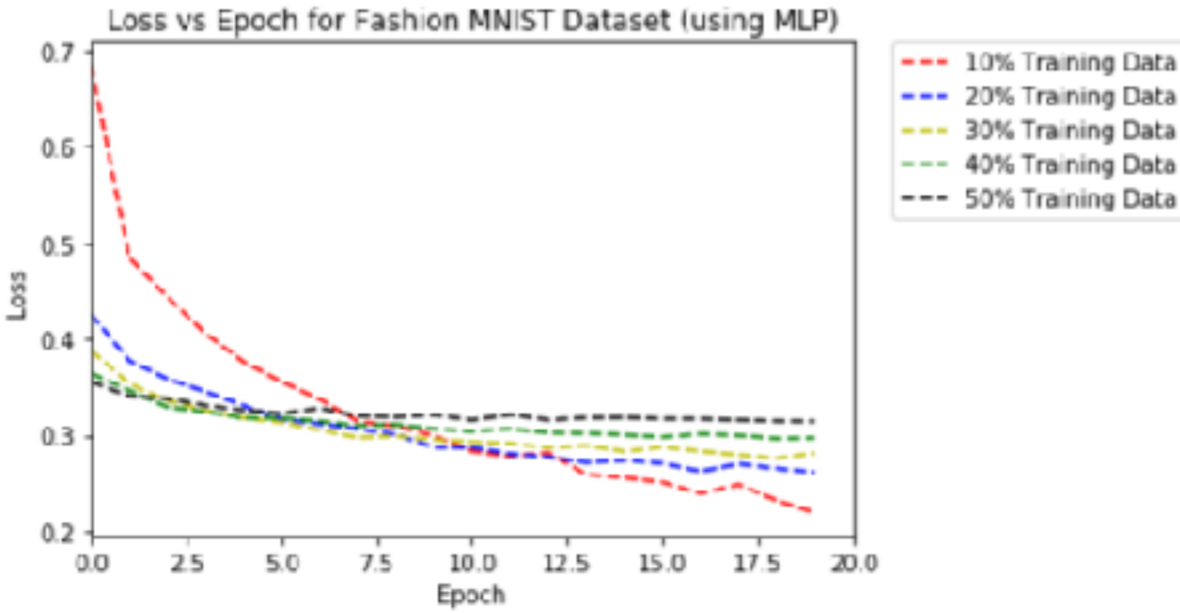


Figure 3 : The Image above shows the Loss versus Epoch graph for Fashion MNIST dataset using Multi-Layer Perceptron Model for different Training Data Sizes.

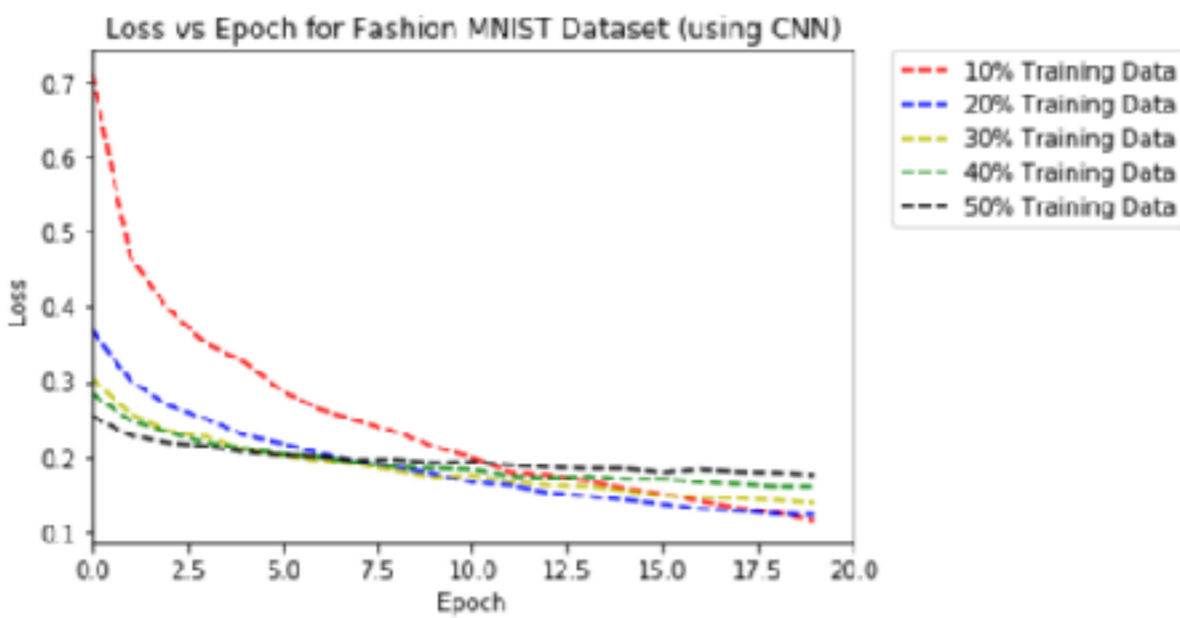


Figure 4 : The Image above shows the Loss versus Epoch graph for Fashion MNIST dataset using Convolutional Neural Network for different Training Data Sizes.

Traning Data Size	Accuracy of MNIST Dataset (using MLP)	Accuracy of Fashion MNIST Dataset (using MLP)	Accuracy of MNIST Dataset (using CNN)	Accuracy of Fashion MNIST Dataset (using CNN)
10 % Training Data	94%	83%	97%	86%
20 % Training Data	95%	84%	97%	88%
30 % Training Data	95%	86%	98%	88%
40 % Training Data	96%	86%	98%	89%
50 % Training Data	95%	87%	98%	90%

Table 1: Accuracy of MNIST and Fashion MNIST dataset using MLP and CNN on different sizes of Training Dataset

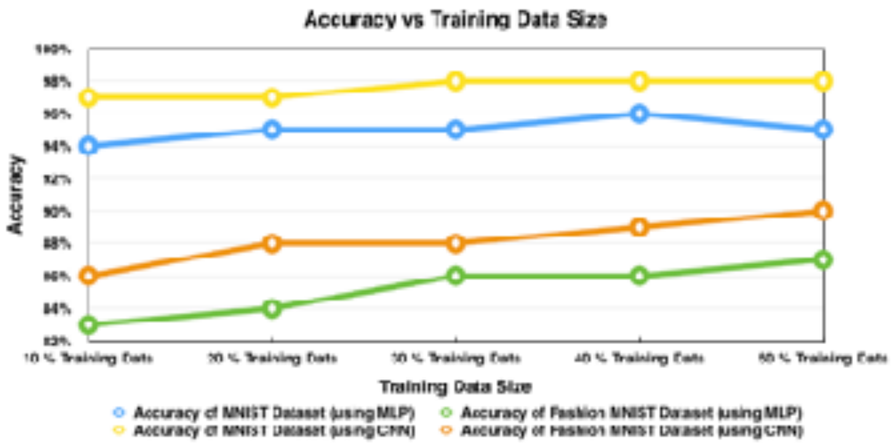


Figure 5: Accuracy of MNIST and Fashion MNIST dataset using MLP and CNN on different sizes of Training Dataset

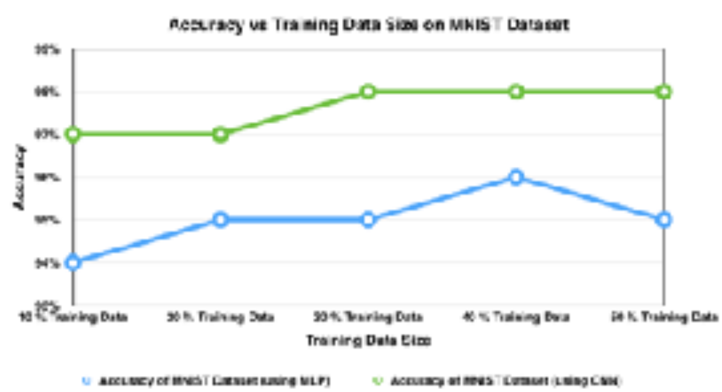


Figure 6: Accuracy of MNIST dataset using MLP and CNN on different sizes of Training Dataset

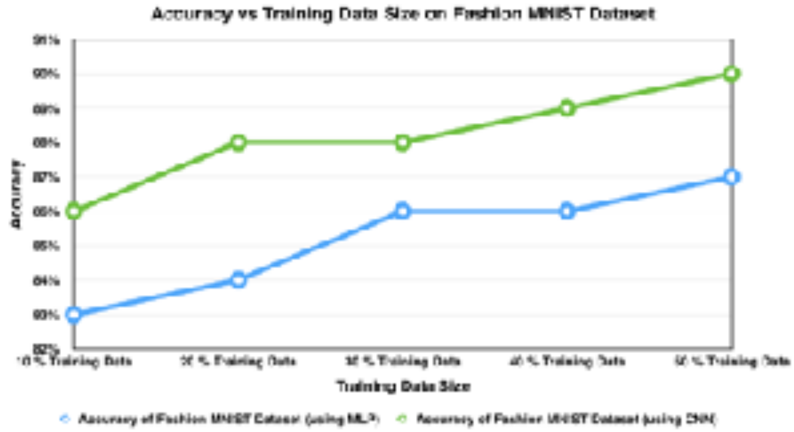


Figure 7: Accuracy of Fashion MNIST dataset using MLP and CNN on different sizes of Training Dataset

Training Data Size	Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
10% Training Data	97%	98%	93%	90%	96%	96%	95%	97%	82%	89%
20% Training Data	97%	97%	95%	95%	97%	92%	97%	96%	96%	90%
30% Training Data	98%	97%	94%	92%	96%	96%	97%	94%	97%	95%
40% Training Data	95%	97%	98%	93%	97%	93%	99%	96%	95%	93%
50% Training Data	97%	98%	97%	96%	91%	95%	93%	93%	96%	96%

Table 2: Accuracy of each class of MNIST dataset using MLP on different sizes of Training Dataset

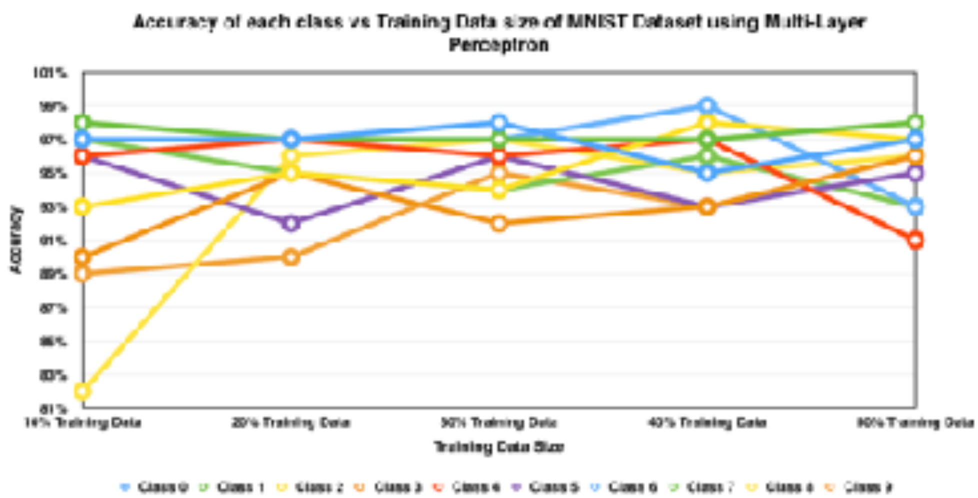


Figure 8: Accuracy of each class of MNIST dataset using MLP on different sizes of Training Dataset

Training Data Size	Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
10% Training Data	98%	98%	97%	97%	98%	97%	98%	94%	98%	97%
20% Training Data	99%	96%	98%	96%	99%	96%	98%	96%	99%	95%
30% Training Data	99%	99%	98%	98%	99%	98%	99%	99%	96%	92%
40% Training Data	99%	99%	98%	99%	98%	97%	97%	99%	97%	98%
50% Training Data	98%	97%	98%	98%	98%	97%	98%	97%	99%	98%

Table 3: Accuracy of each class of MNIST dataset using CNN on different sizes of Training Dataset

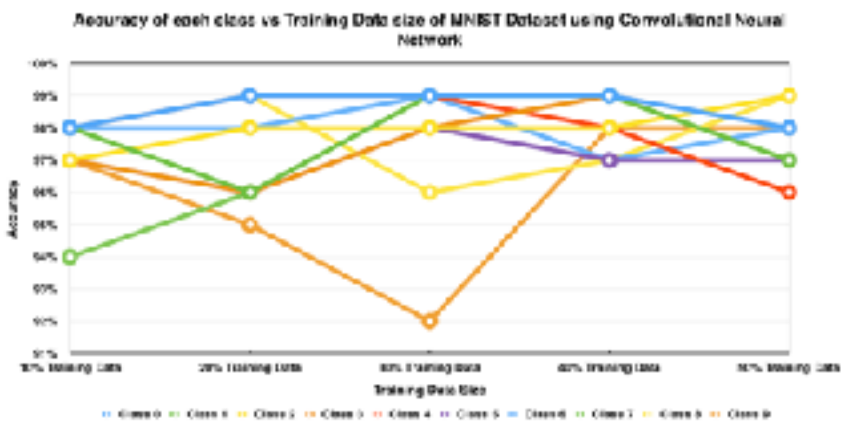


Figure 9: Accuracy of each class of MNIST dataset using CNN on different sizes of Training Dataset

Training Data Size	T-shirt	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle Boot
10% Training Data	83%	95%	71%	73%	93%	88%	46%	94%	95%	94%
20% Training Data	72%	96%	60%	88%	79%	91%	79%	97%	94%	87%
30% Training Data	82%	96%	74%	89%	81%	95%	69%	95%	97%	86%
40% Training Data	88%	97%	75%	82%	87%	91%	63%	97%	95%	89%
50% Training Data	81%	97%	77%	92%	80%	91%	67%	92%	96%	96%

Table 4: Accuracy of each class of Fashion MNIST dataset using MLP on different sizes of Training Dataset

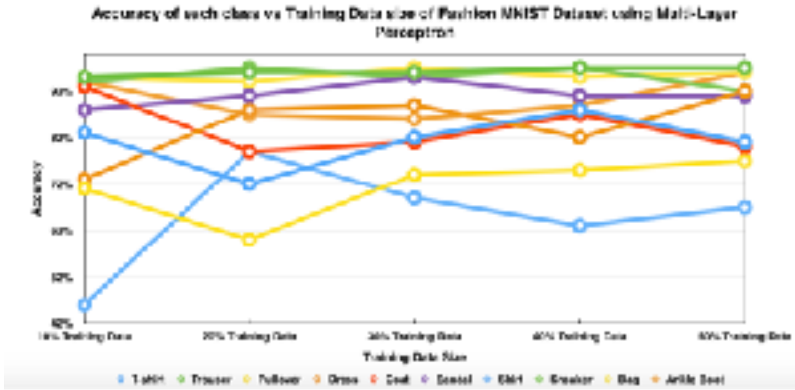


Figure 10: Accuracy of each class of Fashion MNIST dataset using MLP on different sizes of Training Dataset

Training Data Size	T-shirt	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle Boot
10% Training Data	85%	96%	70%	88%	76%	95%	66%	86%	96%	97%
20% Training Data	89%	97%	87%	88%	73%	93%	63%	90%	97%	98%
30% Training Data	77%	97%	82%	94%	85%	95%	70%	98%	96%	92%
40% Training Data	82%	97%	89%	92%	79%	94%	72%	97%	97%	96%
50% Training Data	80%	97%	87%	91%	83%	97%	79%	98%	95%	91%

Table 5: Accuracy of each class of Fashion MNIST dataset using CNN on different sizes of Training Dataset

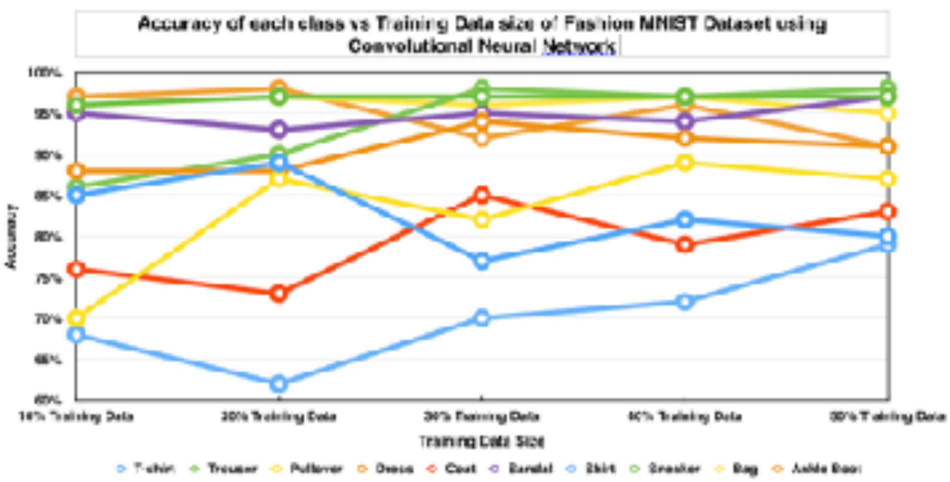


Figure 11: Accuracy of each class of Fashion MNIST dataset using CNN on different sizes of Training Dataset

CONCLUSION

From the above results, it can be observed that the Convolutional Neural Network is found to have performed better than Multi-Layer Perceptron for both the Datasets.
As, the size of the Training Data set is increased, the performance of the model on Test Data set also increases.

REFERENCES

[1] MNIST database of handwritten digits : https://pytorch.org/docs/stable/data.html

[2] Fashion MNIST database of clothes : https://pytorch.org/docs/stable/data.html

[3] Pytorch : https://pytorch.org

[4] Numpy : http://www.numpy.org

[5] Matplotlib : https://matplotlib.org

[6] Pandas : https://pandas.pydata.org

[7] Math : https://docs.python.org/2/library/math.html

[8] Random : https://docs.python.org/2/library/random.html

APPENDIX

CODE

1. Multi-Layer Perceptron on MNIST Data.

```
import torch
import torchvision
import torchvision.datasets as datasets
import torchvision.transforms as transforms
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import random
import math
from torch.utils.data.sampler import SubsetRandomSampler

#Multi-Layer Perceptron Model
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(28*28, 500)
        self.fc2 = nn.Linear(500, 256)
        self.fc3 = nn.Linear(256, 10)


    def forward(self, x):
        x = x.view(-1, 28*28)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


MLP_network = MLP()

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(MLP_network.parameters(), lr=0.001, betas=(0.9,0.99), eps=1e-08, 
weight_decay=0.001)
```

 # transforms to apply to the mnist data
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.1307,), (0.3081,))])

 # MNIST dataset
mnist_trainset = datasets.MNIST(root='./data', train=True, download=True, transform=transform)

num_train = len(mnist_trainset)
indices = list(range(num_train))
#Splitting the Dataset into 10% Training set and 90% Validation Set
split = 54000

 # Random, non-contiguous split
validation_idx = np.random.choice(indices, size=split, replace=False)
train_idx = list(set(indices) - set(validation_idx))

 # Contiguous split
 # train_idx, validation_idx = indices[split:], indices[:split]

define our samplers -- we use a SubsetRandomSampler because it will return
a random subset of the split defined by the given indices without replaf
train_sampler = SubsetRandomSampler(train_idx)
validation_sampler = SubsetRandomSampler(validation_idx)

train_load1 = torch.utils.data.DataLoader(mnist_trainset, batch_size=10, sampler=train_sampler)

validation_load1 = torch.utils.data.DataLoader(mnist_trainset, batch_size=10, sampler=validation_sampler)

```
fig = plt.figure(figsize=(10,10));
columns = 4;
rows = 5;
for i in range(1, 10):
    fig.add_subplot(rows, columns, i)
    fig.tight_layout()
    plt.imshow(mnist_trainset.train_data[i].numpy(), cmap='gray')
    plt.title('Label : %i' % mnist_trainset.train_labels[i])
plt.show()
```

#MLP on 10% of the Training Dataset
num_epochs = 20
total_step = len(train_load1)

```
Loss_1 = []
print('------------------------------------------------------')
for epoch in range(num_epochs):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(train_load1, 0):
        # get the inputs
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = MLP_network(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()

    print('Epoch {}  | Loss : {:.4f}'.format(epoch+1, running_loss / total_step))
    print('------------------------------------------------------')
    Loss_1.append(running_loss / total_step)
print('Finished Training')

#Accuracy of 90% of the Validation Test Dataset
correct = 0
total = 0
with torch.no_grad():
    for data in validation_load1:
        images, labels = data
        outputs = MLP_network(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 54000 Test images: %d %%' % (100 * correct / total))

classes = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
with torch.no_grad():
```

```
    for data in validation_load1:
        images, labels = data
        outputs = MLP_network(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(10):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1


for i in range(10):
    print('Accuracy of class %s : %2d %%' % (
        classes[i], 100 * class_correct[i] / class_total[i]))

num_train = len(mnist_trainset)
indices = list(range(num_train))
```

#Splitting the Dataset into 20% Training set and 80% Validation Set

```
split = 48000

# Random, non-contiguous split
validation_idx = np.random.choice(indices, size=split, replace=False)
train_idx = list(set(indices) - set(validation_idx))

# Contiguous split
# train_idx, validation_idx = indices[split:], indices[:split]

## define our samplers -- we use a SubsetRandomSampler because it will return
## a random subset of the split defined by the given indices without replaf
train_sampler = SubsetRandomSampler(train_idx)
validation_sampler = SubsetRandomSampler(validation_idx)

train_load2 = torch.utils.data.DataLoader(mnist_trainset, batch_size=10, sampler=train_sampler)

validation_load2 = torch.utils.data.DataLoader(mnist_trainset, batch_size=10,
sampler=validation_sampler)
```

#MLP on 20% of the Training Dataset

```
num_epochs = 20
total_step = len(train_load2)
Loss_2 = []
print('---------------------------------------------------')
```

```
for epoch in range(num_epochs):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(train_load2, 0):
        # get the inputs
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = MLP_network(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()

    print('Epoch {}  | Loss : {:.4f}'.format(epoch+1, running_loss / total_step))
    print('---------------------------------------------------')
    Loss_2.append(running_loss / total_step)
print('Finished Training’)

#Accuracy of 80% of the Validation Test Dataset
correct = 0
total = 0
with torch.no_grad():
    for data in validation_load2:
        images, labels = data
        outputs = MLP_network(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 48000 Test images: %d %%' % (100 * correct / total))

classes = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
with torch.no_grad():
    for data in validation_load2:
        images, labels = data
```

```
        outputs = MLP_network(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(10):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1


for i in range(10):
    print('Accuracy of class %s : %2d %%' % (
        classes[i], 100 * class_correct[i] / class_total[i]))

num_train = len(mnist_trainset)
indices = list(range(num_train))
#Splitting the Dataset into 30% Training set and 70% Validation Set
split = 42000

# Random, non-contiguous split
validation_idx = np.random.choice(indices, size=split, replace=False)
train_idx = list(set(indices) - set(validation_idx))

# Contiguous split
# train_idx, validation_idx = indices[split:], indices[:split]

## define our samplers -- we use a SubsetRandomSampler because it will return
## a random subset of the split defined by the given indices without replaf
train_sampler = SubsetRandomSampler(train_idx)
validation_sampler = SubsetRandomSampler(validation_idx)

train_load3 = torch.utils.data.DataLoader(mnist_trainset, batch_size=10, sampler=train_sampler)

validation_load3 = torch.utils.data.DataLoader(mnist_trainset, batch_size=10,
sampler=validation_sampler)

#MLP on 30% of the Training Dataset
num_epochs = 20
total_step = len(train_load3)
Loss_3 = []
print('--------------------------------------------------')
for epoch in range(num_epochs):  # loop over the dataset multiple times

    running_loss = 0.0
```

```
    for i, data in enumerate(train_load3, 0):
        # get the inputs
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = MLP_network(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()

    print('Epoch {}  | Loss : {:.4f}'.format(epoch+1, running_loss / total_step))
    print('-----------------------------------------------------')
    Loss_3.append(running_loss / total_step)
print('Finished Training')
```

#Accuracy of 70% of the Validation Test Dataset

```
correct = 0
total = 0
with torch.no_grad():
    for data in validation_load3:
        images, labels = data
        outputs = MLP_network(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 42000 Test images: %d %%' % (100 * correct / total))

classes = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
with torch.no_grad():
    for data in validation_load3:
        images, labels = data
        outputs = MLP_network(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
```

```
        for i in range(10):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1


for i in range(10):
    print('Accuracy of class %s : %2d %%' % (
        classes[i], 100 * class_correct[i] / class_total[i]))

num_train = len(mnist_trainset)
indices = list(range(num_train))
```
#Splitting the Dataset into 40% Training set and 60% Validation Set
```
split = 36000

# Random, non-contiguous split
validation_idx = np.random.choice(indices, size=split, replace=False)
train_idx = list(set(indices) - set(validation_idx))

# Contiguous split
# train_idx, validation_idx = indices[split:], indices[:split]

## define our samplers -- we use a SubsetRandomSampler because it will return
## a random subset of the split defined by the given indices without replaf
train_sampler = SubsetRandomSampler(train_idx)
validation_sampler = SubsetRandomSampler(validation_idx)

train_load4 = torch.utils.data.DataLoader(mnist_trainset, batch_size=10, sampler=train_sampler)

validation_load4 = torch.utils.data.DataLoader(mnist_trainset, batch_size=10,
sampler=validation_sampler)
```
#MLP on 40% of the Training Dataset
```
num_epochs = 20
total_step = len(train_load4)
Loss_4 = []
print('----------------------------------------------------')
for epoch in range(num_epochs):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(train_load4, 0):
        # get the inputs
        inputs, labels = data
```

```
        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = MLP_network(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()

    print('Epoch {}  | Loss : {:.4f}'.format(epoch+1, running_loss / total_step))
    print('-----------------------------------------------------')
    Loss_4.append(running_loss / total_step)
print('Finished Training')
```

#Accuracy of 60% of the Validation Test Dataset

```
correct = 0
total = 0
with torch.no_grad():
    for data in validation_load4:
        images, labels = data
        outputs = MLP_network(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 36000 Test images: %d %%' % (100 * correct / total))

classes = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
with torch.no_grad():
    for data in validation_load4:
        images, labels = data
        outputs = MLP_network(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(10):
            label = labels[i]
            class_correct[label] += c[i].item()
```

```
            class_total[label] += 1


for i in range(10):
    print('Accuracy of class %s : %2d %%' % (
        classes[i], 100 * class_correct[i] / class_total[i]))

num_train = len(mnist_trainset)
indices = list(range(num_train))
#Splitting the Dataset into 50% Training set and 50% Validation Set
split = 30000

# Random, non-contiguous split
validation_idx = np.random.choice(indices, size=split, replace=False)
train_idx = list(set(indices) - set(validation_idx))

# Contiguous split
# train_idx, validation_idx = indices[split:], indices[:split]

## define our samplers -- we use a SubsetRandomSampler because it will return
## a random subset of the split defined by the given indices without replaf
train_sampler = SubsetRandomSampler(train_idx)
validation_sampler = SubsetRandomSampler(validation_idx)

train_load5 = torch.utils.data.DataLoader(mnist_trainset, batch_size=10, sampler=train_sampler)

validation_load5 = torch.utils.data.DataLoader(mnist_trainset, batch_size=10,
sampler=validation_sampler)

#MLP on 50% of the Training Dataset
num_epochs = 20
total_step = len(train_load5)
Loss_5 = []
print('------------------------------------------------------')
for epoch in range(num_epochs):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(train_load5, 0):
        # get the inputs
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()
```

```
        # forward + backward + optimize
        outputs = MLP_network(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()

    print('Epoch {}  | Loss : {:.4f}'.format(epoch+1, running_loss / total_step))
    print('----------------------------------------------------')
    Loss_5.append(running_loss / total_step)
print('Finished Training')
```

Accuracy of 50% of the Validation Test Dataset

```
correct = 0
total = 0
with torch.no_grad():
    for data in validation_load5:
        images, labels = data
        outputs = MLP_network(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 30000 Test images: %d %%' % (100 * correct / total))

classes = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
with torch.no_grad():
    for data in validation_load5:
        images, labels = data
        outputs = MLP_network(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(10):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1
```

```
for i in range(10):
    print('Accuracy of class %s : %2d %%' % (
        classes[i], 100 * class_correct[i] / class_total[i]))

plt.plot(Loss_1, 'r--', label = "10% Training Data")
plt.plot(Loss_2, 'b--', label = "20% Training Data")
plt.plot(Loss_3, 'y--', label = "30% Training Data")
plt.plot(Loss_4, 'g--', label = "40% Training Data")
plt.plot(Loss_5, 'k--', label = "50% Training Data")
plt.title("Loss vs Epoch for MNIST Dataset (using MLP)")
plt.xlim([0, num_epochs])
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.show()
```

2. Multi-Layer Perceptron on Fashion MNIST Data.

```
import torch
import torchvision
import torchvision.datasets as datasets
import torchvision.transforms as transforms
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import random
import math
from torch.utils.data.sampler import SubsetRandomSampler
```

#Multi-Layer Perceptron Model

```
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(28*28, 500)
        self.fc2 = nn.Linear(500, 256)
        self.fc3 = nn.Linear(256, 10)


    def forward(self, x):
        x = x.view(-1, 28*28)
```

```
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


MLP_network = MLP()

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(MLP_network.parameters(), lr=0.001, betas=(0.9,0.99), eps=1e-08,
weight_decay=0.001)

# transforms to apply to the Fashion mnist data
normalize = transforms.Normalize(mean=[x/255.0 for x in [125.3, 123.0, 113.9]],
                                 std=[x/255.0 for x in [63.0, 62.1, 66.7]])

transform = transforms.Compose([transforms.ToTensor(),
                                transforms.Normalize((0.1307,), (0.3081,))])

# Fashion MNIST dataset
fashion_trainset = datasets.FashionMNIST(root='./fmnist/', train=True, download=True,
transform=transform)

labels_map = ('T-Shirt', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt','Sneaker', 'Bag', 'Ankle
Boot')
fig = plt.figure(figsize=(10,10));
columns = 4;
rows = 5;
for i in range(1, 10):
   fig.add_subplot(rows, columns, i)
   fig.tight_layout()
   plt.imshow(fashion_trainset.train_data[i].numpy(), cmap='gray')
   plt.title('Training Label : %s' % labels_map[fashion_trainset.train_labels[i]])
plt.show()

num_train = len(fashion_trainset)
indices = list(range(num_train))
#Splitting the Dataset into 10% Training set and 90% Validation Set
split = 54000

# Random, non-contiguous split
validation_idx = np.random.choice(indices, size=split, replace=False)
train_idx = list(set(indices) - set(validation_idx))
```

```
# Contiguous split
# train_idx, validation_idx = indices[split:], indices[:split]

## define our samplers -- we use a SubsetRandomSampler because it will return
## a random subset of the split defined by the given indices without replaf
train_sampler = SubsetRandomSampler(train_idx)
validation_sampler = SubsetRandomSampler(validation_idx)

train_loader1 = torch.utils.data.DataLoader(fashion_trainset, batch_size=10,
sampler=train_sampler)

validation_loader1 = torch.utils.data.DataLoader(fashion_trainset, batch_size=10,
sampler=validation_sampler)
```

#MLP on 10% of the Training Dataset

```
num_epochs = 20
total_step = len(train_loader1)
Loss_f1 = []
print('----------------------------------------------------')
for epoch in range(num_epochs):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(train_loader1, 0):
        # get the inputs
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = MLP_network(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()

    print('Epoch {}  | Loss : {:.4f}'.format(epoch+1, running_loss / total_step))
    print('----------------------------------------------------')
    Loss_f1.append(running_loss / total_step)
print('Finished Training')
```

```
#Accuracy of 90% of the Validation Test Dataset
correct = 0
total = 0
with torch.no_grad():
    for data in validation_loader1:
        images, labels = data
        outputs = MLP_network(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 54000 test images: %d %%' % (100 * correct / total))

class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
with torch.no_grad():
    for data in validation_loader1:
        images, labels = data
        outputs = MLP_network(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(10):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1


for i in range(10):
    print('Accuracy of class %5s : %2d %%' % (
        labels_map[i], 100 * class_correct[i] / class_total[i]))

num_train = len(fashion_trainset)
indices = list(range(num_train))
#Splitting the Dataset into 20% Training set and 80% Validation Set
split = 48000

# Random, non-contiguous split
validation_idx = np.random.choice(indices, size=split, replace=False)
train_idx = list(set(indices) - set(validation_idx))

# Contiguous split
# train_idx, validation_idx = indices[split:], indices[:split]
```

```
## define our samplers -- we use a SubsetRandomSampler because it will return
## a random subset of the split defined by the given indices without replaf
train_sampler = SubsetRandomSampler(train_idx)
validation_sampler = SubsetRandomSampler(validation_idx)

train_loader2 = torch.utils.data.DataLoader(fashion_trainset, batch_size=10,
sampler=train_sampler)

validation_loader2 = torch.utils.data.DataLoader(fashion_trainset, batch_size=10,
sampler=validation_sampler)
```

#MLP on 20% of the Training Dataset

```
num_epochs = 20
total_step = len(train_loader2)
Loss_f2 = []
print('----------------------------------------------------')
for epoch in range(num_epochs):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(train_loader2, 0):
        # get the inputs
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = MLP_network(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()

    print('Epoch {}  | Loss : {:.4f}'.format(epoch+1, running_loss / total_step))
    print('----------------------------------------------------')
    Loss_f2.append(running_loss / total_step)
print('Finished Training')
```

#Accuracy of 80% of the Validation Test Dataset

```
correct = 0
total = 0
```

```
with torch.no_grad():
    for data in validation_loader2:
        images, labels = data
        outputs = MLP_network(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 48000 test images: %d %%' % (100 * correct / total))

class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
with torch.no_grad():
    for data in validation_loader2:
        images, labels = data
        outputs = MLP_network(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(10):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1


for i in range(10):
    print('Accuracy of class %5s : %2d %%' % (
        labels_map[i], 100 * class_correct[i] / class_total[i]))

num_train = len(fashion_trainset)
indices = list(range(num_train))
#Splitting the Dataset into 30% Training set and 70% Validation Set
split = 42000

# Random, non-contiguous split
validation_idx = np.random.choice(indices, size=split, replace=False)
train_idx = list(set(indices) - set(validation_idx))

# Contiguous split
# train_idx, validation_idx = indices[split:], indices[:split]

## define our samplers -- we use a SubsetRandomSampler because it will return
## a random subset of the split defined by the given indices without replaf
train_sampler = SubsetRandomSampler(train_idx)
```

```
validation_sampler = SubsetRandomSampler(validation_idx)

train_loader3 = torch.utils.data.DataLoader(fashion_trainset, batch_size=10,
sampler=train_sampler)

validation_loader3 = torch.utils.data.DataLoader(fashion_trainset, batch_size=10,
sampler=validation_sampler)

#MLP on 30% of the Training Dataset
num_epochs = 20
total_step = len(train_loader3)
Loss_f3 = []
print('-------------------------------------------------------')
for epoch in range(num_epochs):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(train_loader3, 0):
        # get the inputs
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = MLP_network(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()

    print('Epoch {}  | Loss : {:.4f}'.format(epoch+1, running_loss / total_step))
    print('-------------------------------------------------------')
    Loss_f3.append(running_loss / total_step)
print('Finished Training’)

#Accuracy of 70% of the Validation Test Dataset
correct = 0
total = 0
with torch.no_grad():
    for data in validation_loader3:
        images, labels = data
```

```
        outputs = MLP_network(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 42000 test images: %d %%' % (100 * correct / total))

class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
with torch.no_grad():
    for data in validation_loader3:
        images, labels = data
        outputs = MLP_network(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(10):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1


for i in range(10):
    print('Accuracy of class %5s : %2d %%' % (
        labels_map[i], 100 * class_correct[i] / class_total[i]))

num_train = len(fashion_trainset)
indices = list(range(num_train))
#Splitting the Dataset into 40% Training set and 60% Validation Set
split = 36000

# Random, non-contiguous split
validation_idx = np.random.choice(indices, size=split, replace=False)
train_idx = list(set(indices) - set(validation_idx))

# Contiguous split
# train_idx, validation_idx = indices[split:], indices[:split]

## define our samplers -- we use a SubsetRandomSampler because it will return
## a random subset of the split defined by the given indices without replaf
train_sampler = SubsetRandomSampler(train_idx)
validation_sampler = SubsetRandomSampler(validation_idx)
```

```
train_loader4 = torch.utils.data.DataLoader(fashion_trainset, batch_size=10,
sampler=train_sampler)

validation_loader4 = torch.utils.data.DataLoader(fashion_trainset, batch_size=10,
sampler=validation_sampler)
```

#MLP on 40% of the Training Dataset

```
num_epochs = 20
total_step = len(train_loader4)
Loss_f4 = []
print('-------------------------------------------------')
for epoch in range(num_epochs):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(train_loader4, 0):
        # get the inputs
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = MLP_network(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()

    print('Epoch {}  | Loss : {:.4f}'.format(epoch+1, running_loss / total_step))
    print('-------------------------------------------------')
    Loss_f4.append(running_loss / total_step)
print('Finished Training')
```

#Accuracy of 60% of the Validation Test Dataset

```
correct = 0
total = 0
with torch.no_grad():
    for data in validation_loader4:
        images, labels = data
        outputs = MLP_network(images)
        _, predicted = torch.max(outputs.data, 1)
```

```
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 36000 test images: %d %%' % (100 * correct / total))

class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
with torch.no_grad():
    for data in validation_loader4:
        images, labels = data
        outputs = MLP_network(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(10):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1


for i in range(10):
    print('Accuracy of class %5s : %2d %%' % (
        labels_map[i], 100 * class_correct[i] / class_total[i]))

num_train = len(fashion_trainset)
indices = list(range(num_train))
#Splitting the Dataset into 50% Training set and 50% Validation Set
split = 30000

# Random, non-contiguous split
validation_idx = np.random.choice(indices, size=split, replace=False)
train_idx = list(set(indices) - set(validation_idx))

# Contiguous split
# train_idx, validation_idx = indices[split:], indices[:split]

## define our samplers -- we use a SubsetRandomSampler because it will return
## a random subset of the split defined by the given indices without replaf
train_sampler = SubsetRandomSampler(train_idx)
validation_sampler = SubsetRandomSampler(validation_idx)

train_loader5 = torch.utils.data.DataLoader(fashion_trainset, batch_size=10,
sampler=train_sampler)
```

```
validation_loader5 = torch.utils.data.DataLoader(fashion_trainset, batch_size=10,
sampler=validation_sampler)
```

#MLP on 50% of the Training Dataset

```
num_epochs = 20
total_step = len(train_loader5)
Loss_f5 = []
print('---------------------------------------------------')
for epoch in range(num_epochs):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(train_loader5, 0):
        # get the inputs
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = MLP_network(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()

    print('Epoch {}  | Loss : {:.4f}'.format(epoch+1, running_loss / total_step))
    print('---------------------------------------------------')
    Loss_f5.append(running_loss / total_step)
print('Finished Training')
```

#Accuracy of 50% of the Validation Test Dataset

```
correct = 0
total = 0
with torch.no_grad():
    for data in validation_loader5:
        images, labels = data
        outputs = MLP_network(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
```

```
print('Accuracy of the network on the 30000 test images: %d %%' % (100 * correct / total))

class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
with torch.no_grad():
    for data in validation_loader5:
        images, labels = data
        outputs = MLP_network(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(10):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1


for i in range(10):
    print('Accuracy of class %5s : %2d %%' % (
        labels_map[i], 100 * class_correct[i] / class_total[i]))

plt.plot(Loss_f1, 'r--', label = "10% Training Data")
plt.plot(Loss_f2, 'b--', label = "20% Training Data")
plt.plot(Loss_f3, 'y--', label = "30% Training Data")
plt.plot(Loss_f4, 'g--', label = "40% Training Data")
plt.plot(Loss_f5, 'k--', label = "50% Training Data")
plt.title("Loss vs Epoch for Fashion MNIST Dataset (using MLP)")
plt.xlim([0, num_epochs])
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.show()
```

3. Convolutional Neural Network on MNIST data.

```
import torch
import torchvision
import torchvision.datasets as datasets
import torchvision.transforms as transforms
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
```

```
import pandas as pd
import random
import math
from torch.utils.data.sampler import SubsetRandomSampler

#Convolution Neural Network
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1,20,5,1)
        self.pool = nn.MaxPool2d(kernel_size=2)
        self.conv2 = nn.Conv2d(20, 40, 5, 1)
        self.fc1 = nn.Linear(4 * 4 * 40, 500)
        self.fc2 = nn.Linear(500, 10)


    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool(x)
        x = F.relu(self.conv2(x))
        x = self.pool(x)
        x = x.view(-1, 4 * 4 * 40)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x


net = Net()

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters(), lr=0.001, betas=(0.9,0.99), eps=1e-08,
weight_decay=0.001)

# transforms to apply to the mnist data
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.1307,),
(0.3081,))])

# MNIST dataset
mnist_trainset = datasets.MNIST(root='./data', train=True, download=True,
transform=transform)

fig = plt.figure(figsize=(10,10));
columns = 4;
```

```
rows = 5;
for i in range(1, 10):
    fig.add_subplot(rows, columns, i)
    fig.tight_layout()
    plt.imshow(mnist_trainset.train_data[i].numpy(), cmap='gray')
    plt.title('Training Label : %i' % mnist_trainset.train_labels[i])
plt.show()

num_train = len(mnist_trainset)
indices = list(range(num_train))
```

#Splitting the Dataset into 10% Training set and 90% Validation Set

```
split = 54000

# Random, non-contiguous split
validation_idx = np.random.choice(indices, size=split, replace=False)
train_idx = list(set(indices) - set(validation_idx))

# Contiguous split
# train_idx, validation_idx = indices[split:], indices[:split]

## define our samplers -- we use a SubsetRandomSampler because it will return
## a random subset of the split defined by the given indices without replaf
train_sampler = SubsetRandomSampler(train_idx)
validation_sampler = SubsetRandomSampler(validation_idx)

train_load1 = torch.utils.data.DataLoader(mnist_trainset, batch_size=10, sampler=train_sampler)

validation_load1 = torch.utils.data.DataLoader(mnist_trainset, batch_size=10,
sampler=validation_sampler)
```

#CNN on 10% of the Training Dataset

```
num_epochs = 20
total_step = len(train_load1)
Loss1 = []
print('-----------------------------------------------------')
for epoch in range(num_epochs):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(train_load1, 0):
        # get the inputs
        inputs, labels = data

        # zero the parameter gradients
```

```
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()

    print('Epoch {}  | Loss : {:.4f}'.format(epoch+1, running_loss / total_step))
    print('----------------------------------------------------')
    Loss1.append(running_loss / total_step)
print('Finished Training’)
```

#Accuracy of 90% of the Validation Test Dataset

```
correct = 0
total = 0
with torch.no_grad():
    for data in validation_load1:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 54000 test images: %d %%' % (100 * correct / total)

classes = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
with torch.no_grad():
    for data in validation_load1:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(10):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1
```

```
for i in range(10):
    print('Accuracy of class %s : %2d %%' % (
        classes[i], 100 * class_correct[i] / class_total[i]))

num_train = len(mnist_trainset)
indices = list(range(num_train))
```
#Splitting the Dataset into 20% Training set and 80% Validation Set
```
split = 48000

# Random, non-contiguous split
validation_idx = np.random.choice(indices, size=split, replace=False)
train_idx = list(set(indices) - set(validation_idx))

# Contiguous split
# train_idx, validation_idx = indices[split:], indices[:split]

## define our samplers -- we use a SubsetRandomSampler because it will return
## a random subset of the split defined by the given indices without replaf
train_sampler = SubsetRandomSampler(train_idx)
validation_sampler = SubsetRandomSampler(validation_idx)

train_load2 = torch.utils.data.DataLoader(mnist_trainset, batch_size=10, sampler=train_sampler)

validation_load2 = torch.utils.data.DataLoader(mnist_trainset, batch_size=10,
sampler=validation_sampler)
```
#CNN on 20% of the Training Dataset
```
num_epochs = 20
total_step = len(train_load2)
Loss2 = []
print('--------------------------------------------------')
for epoch in range(num_epochs):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(train_load2, 0):
        # get the inputs
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
```

```
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()

    print('Epoch {}  | Loss : {:.4f}'.format(epoch+1, running_loss / total_step))
    print('---------------------------------------------------')
    Loss2.append(running_loss / total_step)
print('Finished Training')
```

#Accuracy of 80% of the Validation Test Dataset

```
correct = 0
total = 0
with torch.no_grad():
    for data in validation_load2:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 48000 test images: %d %%' % (100 * correct / total))

classes = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
with torch.no_grad():
    for data in validation_load2:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(10):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1


for i in range(10):
    print('Accuracy of class %s : %2d %%' % (
```

```
        classes[i], 100 * class_correct[i] / class_total[i]))

num_train = len(mnist_trainset)
indices = list(range(num_train))
```
#Splitting the Dataset into 30% Training set and 70% Validation Set
```
split = 42000

# Random, non-contiguous split
validation_idx = np.random.choice(indices, size=split, replace=False)
train_idx = list(set(indices) - set(validation_idx))

# Contiguous split
# train_idx, validation_idx = indices[split:], indices[:split]

## define our samplers -- we use a SubsetRandomSampler because it will return
## a random subset of the split defined by the given indices without replaf
train_sampler = SubsetRandomSampler(train_idx)
validation_sampler = SubsetRandomSampler(validation_idx)

train_load3 = torch.utils.data.DataLoader(mnist_trainset, batch_size=10, sampler=train_sampler)

validation_load3 = torch.utils.data.DataLoader(mnist_trainset, batch_size=10,
sampler=validation_sampler)
```
#CNN on 30% of the Training Dataset
```
num_epochs = 20
total_step = len(train_load3)
Loss3 = []
print('--------------------------------------------------')
for epoch in range(num_epochs):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(train_load3, 0):
        # get the inputs
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
```

```
        optimizer.step()

        # print statistics
        running_loss += loss.item()

    print('Epoch {}  | Loss : {:.4f}'.format(epoch+1, running_loss / total_step))
    print('----------------------------------------------------')
    Loss3.append(running_loss / total_step)
print('Finished Training')
```

#Accuracy of 70% of the Validation Test Dataset

```
correct = 0
total = 0
with torch.no_grad():
    for data in validation_load3:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 42000 test images: %d %%' % (100 * correct / total))

classes = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
with torch.no_grad():
    for data in validation_load3:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(10):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1


for i in range(10):
    print('Accuracy of class %s : %2d %%' % (
        classes[i], 100 * class_correct[i] / class_total[i]))

num_train = len(mnist_trainset)
```

```
indices = list(range(num_train))
```
#Splitting the Dataset into 40% Training set and 60% Validation Set
```
split = 36000

# Random, non-contiguous split
validation_idx = np.random.choice(indices, size=split, replace=False)
train_idx = list(set(indices) - set(validation_idx))

# Contiguous split
# train_idx, validation_idx = indices[split:], indices[:split]

## define our samplers -- we use a SubsetRandomSampler because it will return
## a random subset of the split defined by the given indices without replaf
train_sampler = SubsetRandomSampler(train_idx)
validation_sampler = SubsetRandomSampler(validation_idx)

train_load4 = torch.utils.data.DataLoader(mnist_trainset, batch_size=10, sampler=train_sampler)

validation_load4 = torch.utils.data.DataLoader(mnist_trainset, batch_size=10,
sampler=validation_sampler)

#CNN on 40% of the Training Dataset
num_epochs = 20
total_step = len(train_load4)
Loss4 = []
print('---------------------------------------------------')
for epoch in range(num_epochs):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(train_load4, 0):
        # get the inputs
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
```

```
        running_loss += loss.item()

    print('Epoch {}  | Loss : {:.4f}'.format(epoch+1, running_loss / total_step))
    print('-------------------------------------------------')
    Loss4.append(running_loss / total_step)
print('Finished Training')
```

#Accuracy of 60% of the Validation Test Dataset

```
correct = 0
total = 0
with torch.no_grad():
    for data in validation_load4:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 36000 test images: %d %%' % (100 * correct / total))

classes = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
with torch.no_grad():
    for data in validation_load4:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(10):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1


for i in range(10):
    print('Accuracy of class %s : %2d %%' % (
        classes[i], 100 * class_correct[i] / class_total[i]))

num_train = len(mnist_trainset)
indices = list(range(num_train))
```

#Splitting the Dataset into 50% Training set and 50% Validation Set

```
split = 30000
```

```
# Random, non-contiguous split
validation_idx = np.random.choice(indices, size=split, replace=False)
train_idx = list(set(indices) - set(validation_idx))

# Contiguous split
# train_idx, validation_idx = indices[split:], indices[:split]

## define our samplers -- we use a SubsetRandomSampler because it will return
## a random subset of the split defined by the given indices without replaf
train_sampler = SubsetRandomSampler(train_idx)
validation_sampler = SubsetRandomSampler(validation_idx)

train_load5 = torch.utils.data.DataLoader(mnist_trainset, batch_size=10, sampler=train_sampler)

validation_load5 = torch.utils.data.DataLoader(mnist_trainset, batch_size=10,
sampler=validation_sampler)
```

#CNN on 50% of the Training Dataset

```
num_epochs = 20
total_step = len(train_load5)
Loss5 = []
print('---------------------------------------------------')
for epoch in range(num_epochs):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(train_load5, 0):
        # get the inputs
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()

    print('Epoch {}  | Loss : {:.4f}'.format(epoch+1, running_loss / total_step))
```

```
    print('---------------------------------------------------')
    Loss5.append(running_loss / total_step)
print('Finished Training')
```

#Accuracy of 50% of the Validation Test Dataset

```
correct = 0
total = 0
with torch.no_grad():
    for data in validation_load5:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 30000 test images: %d %%' % (100 * correct / total))

classes = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
with torch.no_grad():
    for data in validation_load5:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(10):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1


for i in range(10):
    print('Accuracy of class %s : %2d %%' % (
        classes[i], 100 * class_correct[i] / class_total[i]))

plt.plot(Loss1, 'r--', label = "10% Training Data")
plt.plot(Loss2, 'b--', label = "20% Training Data")
plt.plot(Loss3, 'y--', label = "30% Training Data")
plt.plot(Loss4, 'g--', label = "40% Training Data")
plt.plot(Loss5, 'k--', label = "50% Training Data")
plt.title("Loss vs Epoch for MNIST Dataset (using CNN)")
plt.xlim([0, num_epochs])
```

```
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.show()
```

4. Convolutional Neural Network on Fashion MNIST Dataset

```
import torch
import torchvision
import torchvision.datasets as datasets
import torchvision.transforms as transforms
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import random
import math
from torch.utils.data.sampler import SubsetRandomSampler

#Convolution Neural Network
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1,20,5,1)
        self.pool = nn.MaxPool2d(kernel_size=2)
        self.conv2 = nn.Conv2d(20, 40, 5, 1)
        self.fc1 = nn.Linear(4 * 4 * 40, 500)
        self.fc2 = nn.Linear(500, 10)


    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool(x)
        x = F.relu(self.conv2(x))
        x = self.pool(x)
        x = x.view(-1, 4 * 4 * 40)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

```
net = Net()

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters(), lr=0.001, betas=(0.9,0.99), eps=1e-08,
weight_decay=0.001)

# transforms to apply to the Fashion mnist data
normalize = transforms.Normalize(mean=[x/255.0 for x in [125.3, 123.0, 113.9]],
                                 std=[x/255.0 for x in [63.0, 62.1, 66.7]])

transform = transforms.Compose([transforms.ToTensor(),
                              transforms.Normalize((0.1307,), (0.3081,))])

# Fashion MNIST dataset
fashion_trainset = datasets.FashionMNIST(root='./fmnist/', train=True, download=True,
transform=transform)

labels_map = ('T-Shirt', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt','Sneaker', 'Bag', 'Ankle
Boot')
fig = plt.figure(figsize=(10,10));
columns = 4;
rows = 5;
for i in range(1, 10):
   fig.add_subplot(rows, columns, i)
   fig.tight_layout()
   plt.imshow(fashion_trainset.train_data[i].numpy(), cmap='gray')
   plt.title('Training Label : %s' % labels_map[fashion_trainset.train_labels[i]])
plt.show()

num_train = len(fashion_trainset)
indices = list(range(num_train))
#Splitting the Dataset into 10% Training set and 90% Validation Set
split = 54000

# Random, non-contiguous split
validation_idx = np.random.choice(indices, size=split, replace=False)
train_idx = list(set(indices) - set(validation_idx))

# Contiguous split
# train_idx, validation_idx = indices[split:], indices[:split]

## define our samplers -- we use a SubsetRandomSampler because it will return
## a random subset of the split defined by the given indices without replaf
```

```
train_sampler = SubsetRandomSampler(train_idx)
validation_sampler = SubsetRandomSampler(validation_idx)

train_loader1 = torch.utils.data.DataLoader(fashion_trainset, batch_size=10,
sampler=train_sampler)

validation_loader1 = torch.utils.data.DataLoader(fashion_trainset, batch_size=10,
sampler=validation_sampler)
```

#CNN on 10% of the Training Dataset

```
num_epochs = 20
total_step = len(train_loader1)
Loss_f1 = []
print('----------------------------------------------------')
for epoch in range(num_epochs):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(train_loader1, 0):
        # get the inputs
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()

    print('Epoch {}  | Loss : {:.4f}'.format(epoch+1, running_loss / total_step))
    print('----------------------------------------------------')
    Loss_f1.append(running_loss / total_step)
print('Finished Training’)

#Accuracy of 90% of the Validation Test Dataset
correct = 0
total = 0
with torch.no_grad():
    for data in validation_loader1:
```

```
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 54000 test images: %d %%' % (100 * correct / total))

class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
with torch.no_grad():
    for data in validation_loader1:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(10):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1


for i in range(10):
    print('Accuracy of class %5s : %2d %%' % (
        labels_map[i], 100 * class_correct[i] / class_total[i]))

num_train = len(fashion_trainset)
indices = list(range(num_train))
#Splitting the Dataset into 20% Training set and 80% Validation Set
split = 48000

# Random, non-contiguous split
validation_idx = np.random.choice(indices, size=split, replace=False)
train_idx = list(set(indices) - set(validation_idx))

# Contiguous split
# train_idx, validation_idx = indices[split:], indices[:split]

## define our samplers -- we use a SubsetRandomSampler because it will return
## a random subset of the split defined by the given indices without replaf
train_sampler = SubsetRandomSampler(train_idx)
validation_sampler = SubsetRandomSampler(validation_idx)
```

```
train_loader2 = torch.utils.data.DataLoader(fashion_trainset, batch_size=10,
sampler=train_sampler)

validation_loader2 = torch.utils.data.DataLoader(fashion_trainset, batch_size=10,
sampler=validation_sampler)
```

#CNN on 20% of the Training Dataset

```
num_epochs = 20
total_step = len(train_loader2)
Loss_f2 = []
print('-----------------------------------------------------')
for epoch in range(num_epochs):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(train_loader2, 0):
        # get the inputs
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()

    print('Epoch {}  | Loss : {:.4f}'.format(epoch+1, running_loss / total_step))
    print('-----------------------------------------------------')
    Loss_f2.append(running_loss / total_step)
print('Finished Training')
```

#Accuracy of 80% of the Validation Test Dataset

```
correct = 0
total = 0
with torch.no_grad():
    for data in validation_loader2:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
```

```
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 48000 test images: %d %%' % (100 * correct / total))

class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
with torch.no_grad():
    for data in validation_loader2:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(10):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1


for i in range(10):
    print('Accuracy of class %5s : %2d %%' % (
        labels_map[i], 100 * class_correct[i] / class_total[i]))

num_train = len(fashion_trainset)
indices = list(range(num_train))
#Splitting the Dataset into 30% Training set and 70% Validation Set
split = 42000

# Random, non-contiguous split
validation_idx = np.random.choice(indices, size=split, replace=False)
train_idx = list(set(indices) - set(validation_idx))

# Contiguous split
# train_idx, validation_idx = indices[split:], indices[:split]

## define our samplers -- we use a SubsetRandomSampler because it will return
## a random subset of the split defined by the given indices without replaf
train_sampler = SubsetRandomSampler(train_idx)
validation_sampler = SubsetRandomSampler(validation_idx)

train_loader3 = torch.utils.data.DataLoader(fashion_trainset, batch_size=10,
sampler=train_sampler)
```

```
validation_loader3 = torch.utils.data.DataLoader(fashion_trainset, batch_size=10,
sampler=validation_sampler)

#CNN on 30% of the Training Dataset
num_epochs = 20
total_step = len(train_loader3)
Loss_f3 = []
print('--------------------------------------------------')
for epoch in range(num_epochs):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(train_loader3, 0):
        # get the inputs
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()

    print('Epoch {}  | Loss : {:.4f}'.format(epoch+1, running_loss / total_step))
    print('--------------------------------------------------')
    Loss_f3.append(running_loss / total_step)
print('Finished Training’)

#Accuracy of 70% of the Validation Test Dataset
correct = 0
total = 0
with torch.no_grad():
    for data in validation_loader3:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
```

```
print('Accuracy of the network on the 42000 test images: %d %%' % (100 * correct / total))

class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
with torch.no_grad():
    for data in validation_loader3:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(10):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1


for i in range(10):
    print('Accuracy of class %5s : %2d %%' % (
        labels_map[i], 100 * class_correct[i] / class_total[i]))

num_train = len(fashion_trainset)
indices = list(range(num_train))
```
#Splitting the Dataset into 40% Training set and 60% Validation Set
```
split = 36000

# Random, non-contiguous split
validation_idx = np.random.choice(indices, size=split, replace=False)
train_idx = list(set(indices) - set(validation_idx))

# Contiguous split
# train_idx, validation_idx = indices[split:], indices[:split]

## define our samplers -- we use a SubsetRandomSampler because it will return
## a random subset of the split defined by the given indices without replaf
train_sampler = SubsetRandomSampler(train_idx)
validation_sampler = SubsetRandomSampler(validation_idx)

train_loader4 = torch.utils.data.DataLoader(fashion_trainset, batch_size=10,
sampler=train_sampler)

validation_loader4 = torch.utils.data.DataLoader(fashion_trainset, batch_size=10,
sampler=validation_sampler)
```

```
#CNN on 40% of the Training Dataset
num_epochs = 20
total_step = len(train_loader4)
Loss_f4 = []
print('---------------------------------------------------')
for epoch in range(num_epochs):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(train_loader4, 0):
        # get the inputs
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()

    print('Epoch {}  | Loss : {:.4f}'.format(epoch+1, running_loss / total_step))
    print('---------------------------------------------------')
    Loss_f4.append(running_loss / total_step)
print('Finished Training')

#Accuracy of 60% of the Validation Test Dataset
correct = 0
total = 0
with torch.no_grad():
    for data in validation_loader4:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 36000 test images: %d %%' % (100 * correct / total))

class_correct = list(0. for i in range(10))
```

```
class_total = list(0. for i in range(10))
with torch.no_grad():
    for data in validation_loader4:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(10):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1


for i in range(10):
    print('Accuracy of class %5s : %2d %%' % (
        labels_map[i], 100 * class_correct[i] / class_total[i]))

num_train = len(fashion_trainset)
indices = list(range(num_train))
#Splitting the Dataset into 50% Training set and 50% Validation Set
split = 30000

# Random, non-contiguous split
validation_idx = np.random.choice(indices, size=split, replace=False)
train_idx = list(set(indices) - set(validation_idx))

# Contiguous split
# train_idx, validation_idx = indices[split:], indices[:split]

## define our samplers -- we use a SubsetRandomSampler because it will return
## a random subset of the split defined by the given indices without replaf
train_sampler = SubsetRandomSampler(train_idx)
validation_sampler = SubsetRandomSampler(validation_idx)

train_loader5 = torch.utils.data.DataLoader(fashion_trainset, batch_size=10,
sampler=train_sampler)

validation_loader5 = torch.utils.data.DataLoader(fashion_trainset, batch_size=10,
sampler=validation_sampler)

#CNN on 50% of the Training Dataset
num_epochs = 20
total_step = len(train_loader5)
```

```
Loss_f5 = []
print('-------------------------------------------------')
for epoch in range(num_epochs):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(train_loader5, 0):
        # get the inputs
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()

    print('Epoch {}  | Loss : {:.4f}'.format(epoch+1, running_loss / total_step))
    print('-------------------------------------------------')
    Loss_f5.append(running_loss / total_step)
print('Finished Training')
```

#Accuracy of 50% of the Validation Test Dataset

```
correct = 0
total = 0
with torch.no_grad():
    for data in validation_loader5:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 30000 test images: %d %%' % (100 * correct / total))

class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
with torch.no_grad():
    for data in validation_loader5:
```

```
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(10):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1


for i in range(10):
    print('Accuracy of class %5s : %2d %%' % (
        labels_map[i], 100 * class_correct[i] / class_total[i]))

plt.plot(Loss_f1, 'r--', label = "10% Training Data")
plt.plot(Loss_f2, 'b--', label = "20% Training Data")
plt.plot(Loss_f3, 'y--', label = "30% Training Data")
plt.plot(Loss_f4, 'g--', label = "40% Training Data")
plt.plot(Loss_f5, 'k--', label = "50% Training Data")
plt.title("Loss vs Epoch for Fashion MNIST Dataset (using CNN)")
plt.xlim([0, num_epochs])
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.show()
```