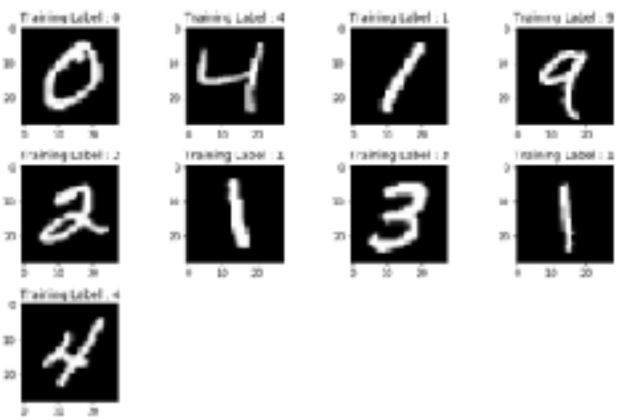
FEATURE EXTRACTION AND CLASSIFICATION OF MNIST AND FASHION MNIST DATASETS

Shivangi Gupta
A25266618
CS696-Big Data Analytics

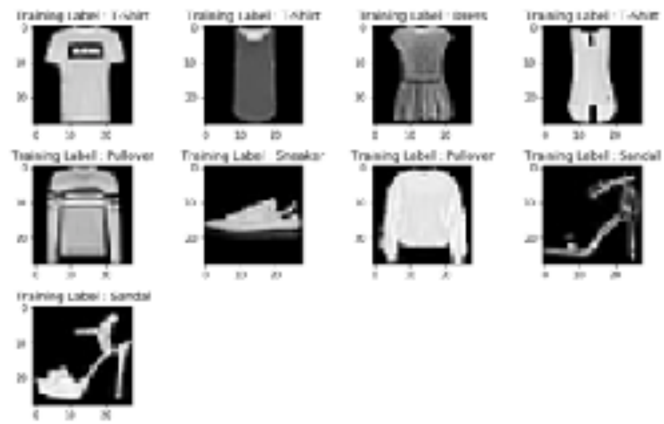
TABLE OF CONTENTS

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



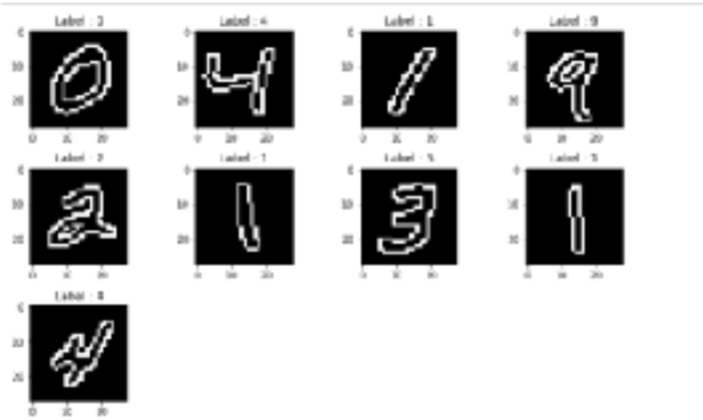
Fashion MNIST Dataset

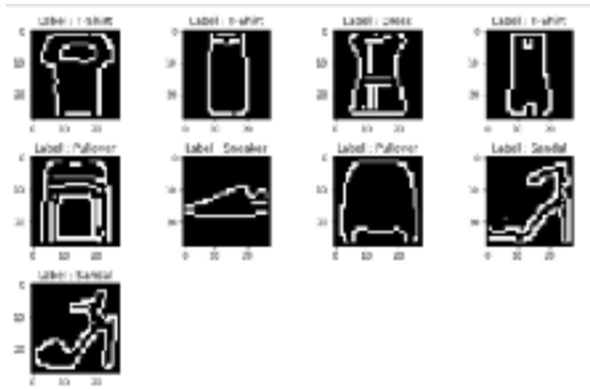
To Classify the images, I have used Multi-Layer perceptron and Convolution Neural Network.

FEATURE EXTRACTION METHODS

CANNY EDGE DETECTOR METHOD

The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. It uses a filter based on the derivative of a Gaussian in order to compute the intensity of the gradients.The Gaussian reduces the effect of noise present in the image. Then, potential edges are thinned down to 1-pixel curves by removing non-maximum pixels of the gradient magnitude. Finally, edge pixels are kept or removed using hysteresis thresholding on the gradient magnitude. The Images below depicts the processed images after applying the Canny Edge Detector Method.

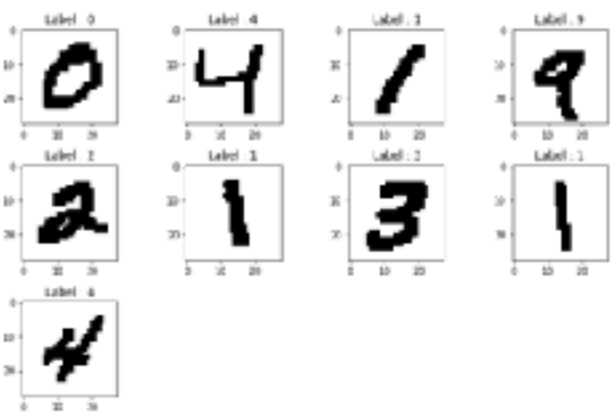


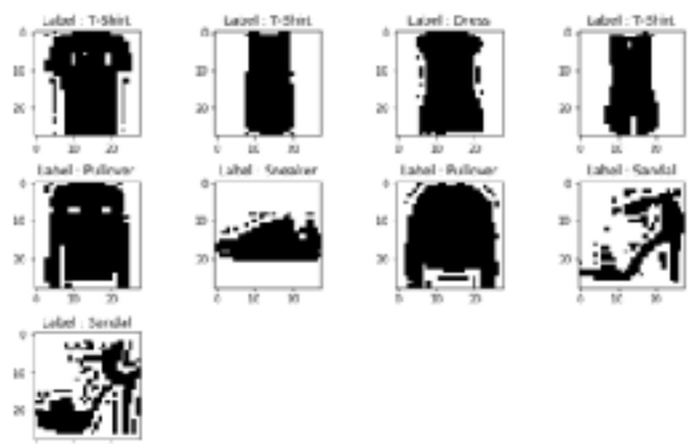


MNIST Dataset **Fashion MNIST Dataset**

LOCAL BINARY PATTERN

Local Binary Pattern (LBP) looks at points surrounding a central point and tests whether the surrounding points are greater than or less than the central point (i.e. gives a binary result). The Images below depicts the processed images after applying the LBP.

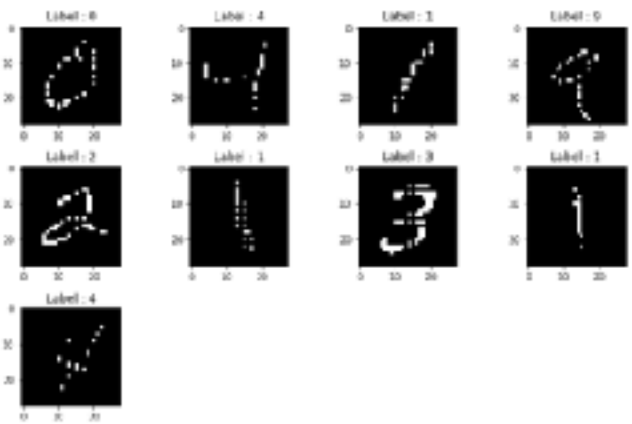




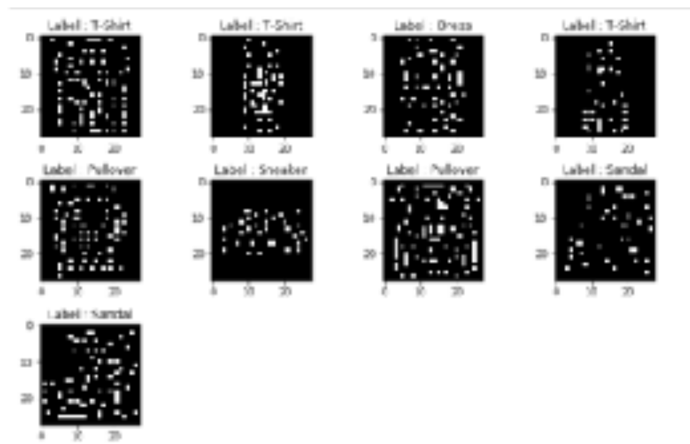
MNIST Dataset **Fashion MNIST Datas**

PEAK LOCAL MAXIMA METHOD

Find peaks in an image as coordinate list or boolean mask.
Peaks are the local maxima in a region of 2 * min_distance + 1 (i.e. peaks are separated by at least min_distance).
If peaks are flat (i.e. multiple adjacent pixels have identical intensities), the coordinates of all such pixels are returned. The Images below depicts the processed images after applying the Peak Local Max Method.



MNIST Dataset

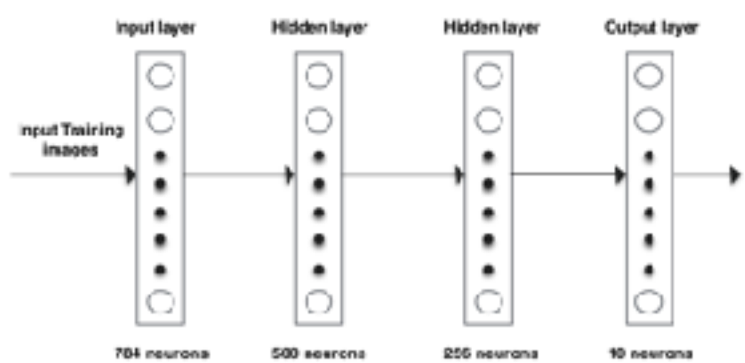


Fashion MNIST Dataset

METHOD OF CLASSIFICATION

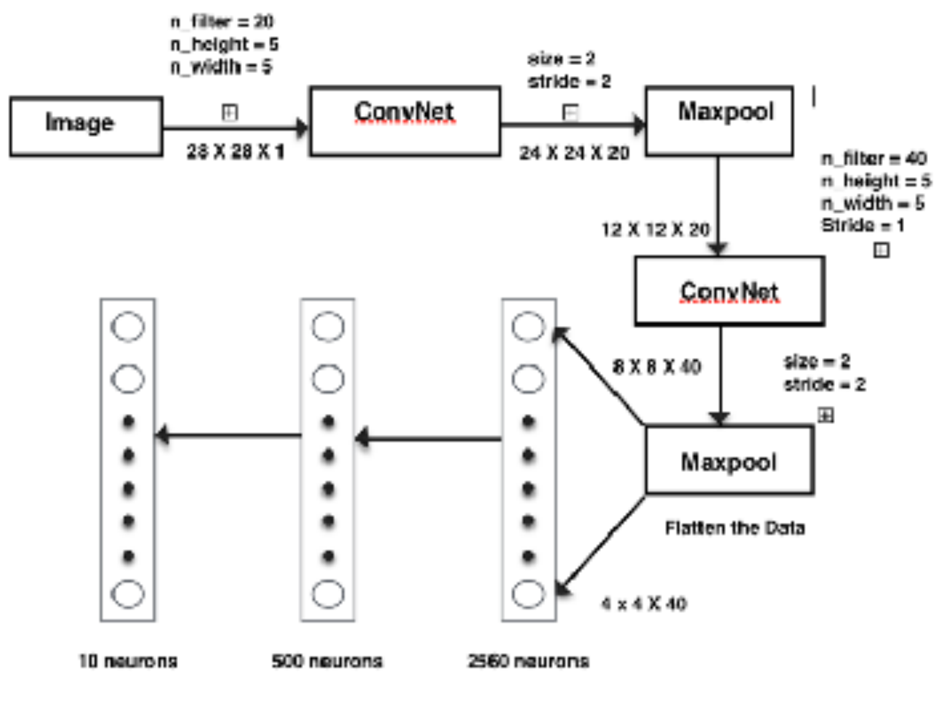
MULTI-LAYER PERCEPTRON

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

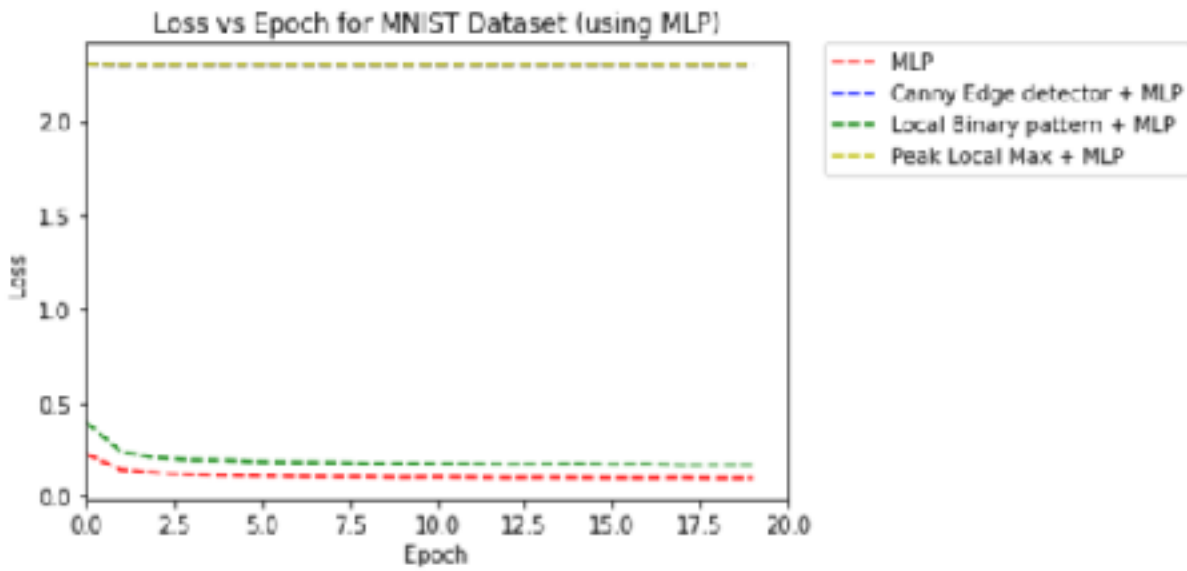


Figure 1 : The Image above shows the Loss versus Epoch graph for MNIST dataset using Different Feature Extraction Methods and MLP as Classifier

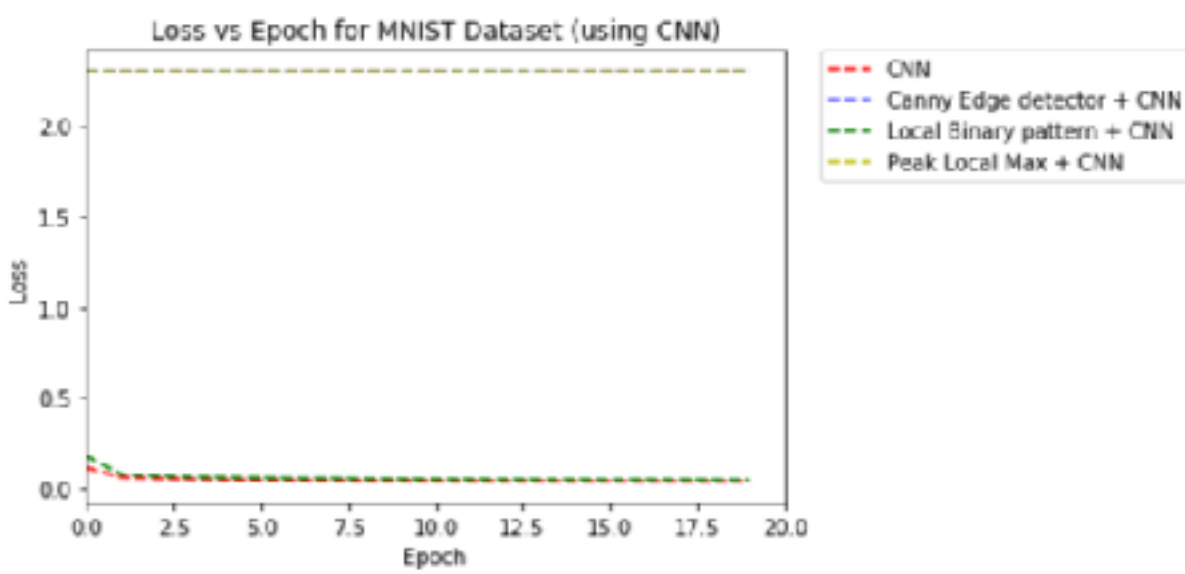


Figure 2 : The Image above shows the Loss versus Epoch graph for MNIST dataset using Different Feature Extraction Methods and CNN as Classifier

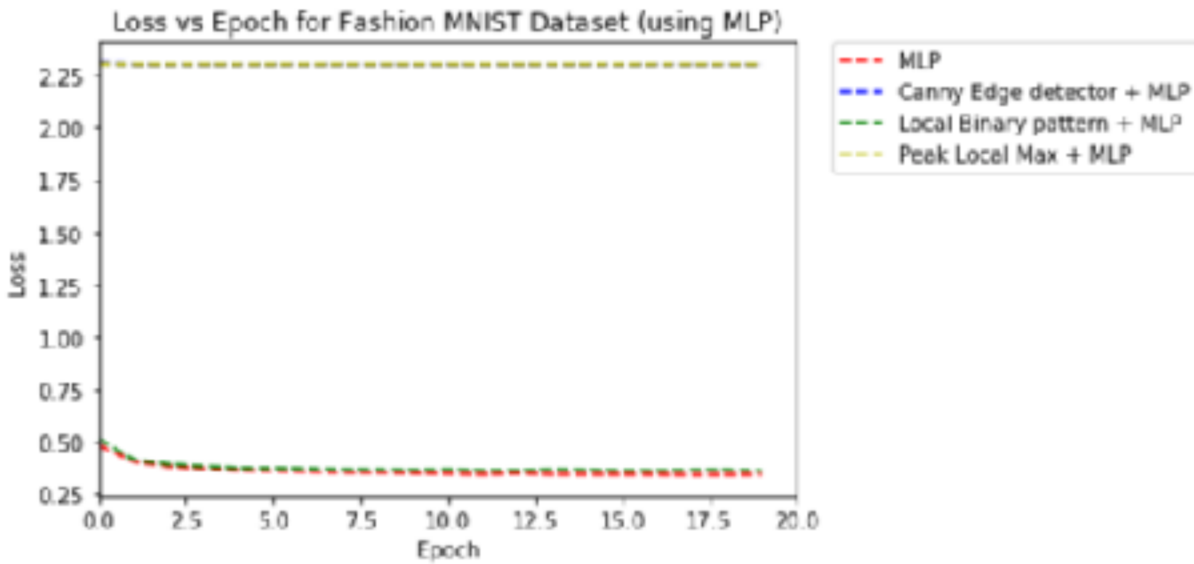


Figure 3 : The Image above shows the Loss versus Epoch graph for Fashion MNIST dataset using Different Feature Extraction Methods and MLP as Classifier

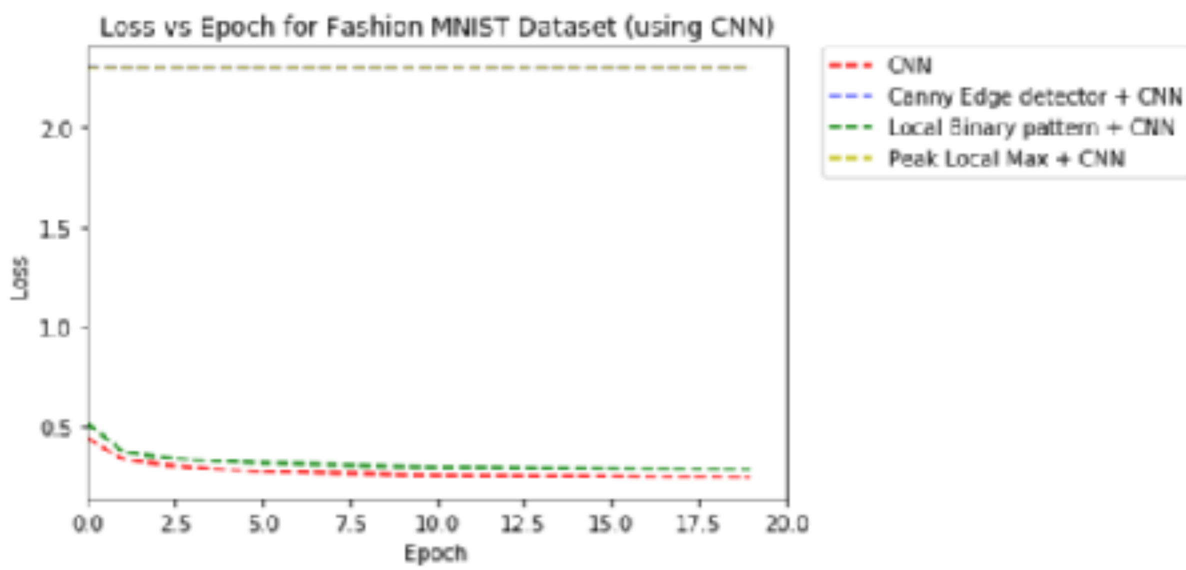


Figure 4 : The Image above shows the Loss versus Epoch graph for Fashion MNIST dataset using Different Feature Extraction Methods and CNN as Classifier

Feature Extraction Method	Multi-layer Perceptron	Convolutional Neural Network
None	94%	97%
Canny Edge Detector	11%	11%
Local Binary Pattern	96%	99%
Peak Local Max	11%	11%

Table 1: Accuracy of MNIST dataset using Different Feature Extraction Methods and MLP and CNN as Classifiers

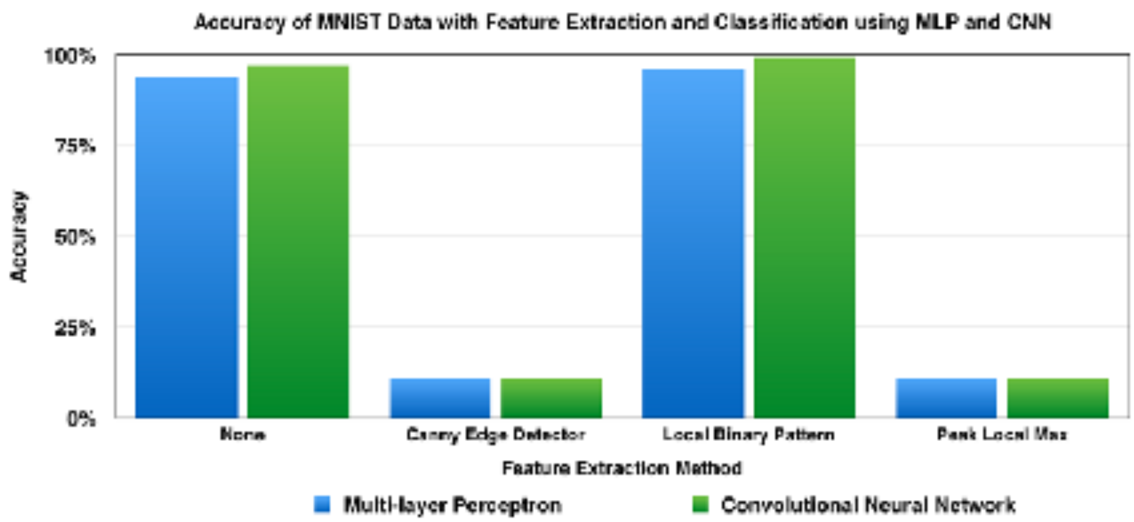


Figure 5: Accuracy of MNIST dataset using Different Feature Extraction Methods and MLP and CNN as Classifiers

Feature Extraction Method	Multi-layer Perceptron	Convolutional Neural Network
None	15	20
Canny Edge Detector	17	24
Local Binary Pattern	14	21
Peak Local Max	16	23

Table 2: Computation Time on MNIST dataset using Different Feature Extraction Methods and MLP and CNN as Classifiers

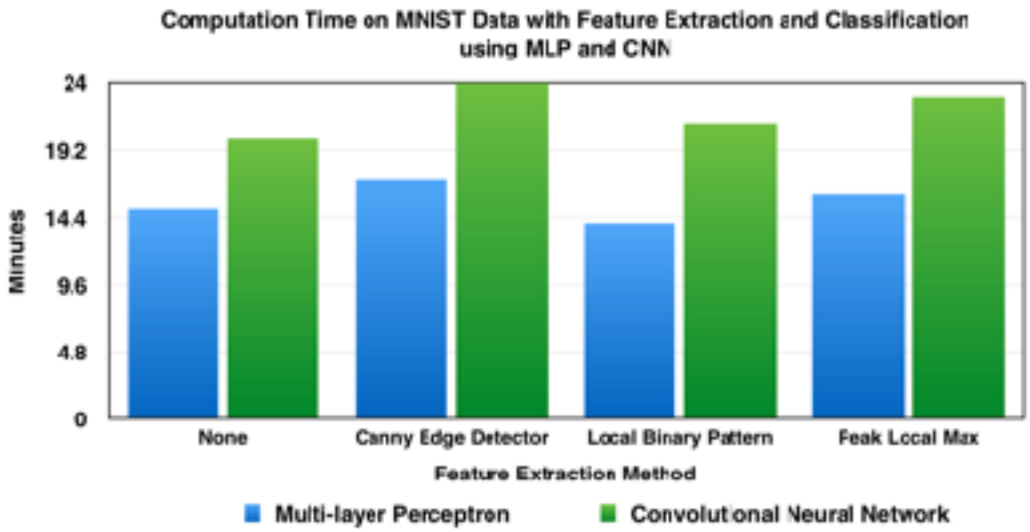


Figure 6: Computation Time on MNIST dataset using Different Feature Extraction Methods and MLP and CNN as Classifiers

Feature Extraction Method	Multi-layer Perceptron	Convolutional Neural Network
None	84%	89%
Canny Edge Detector	10%	10%
Local Binary Pattern	86%	87%
Peak Local Max	10%	10%

Table 3: Accuracy of Fashion MNIST dataset using Different Feature Extraction Methods and MLP and CNN as Classifiers

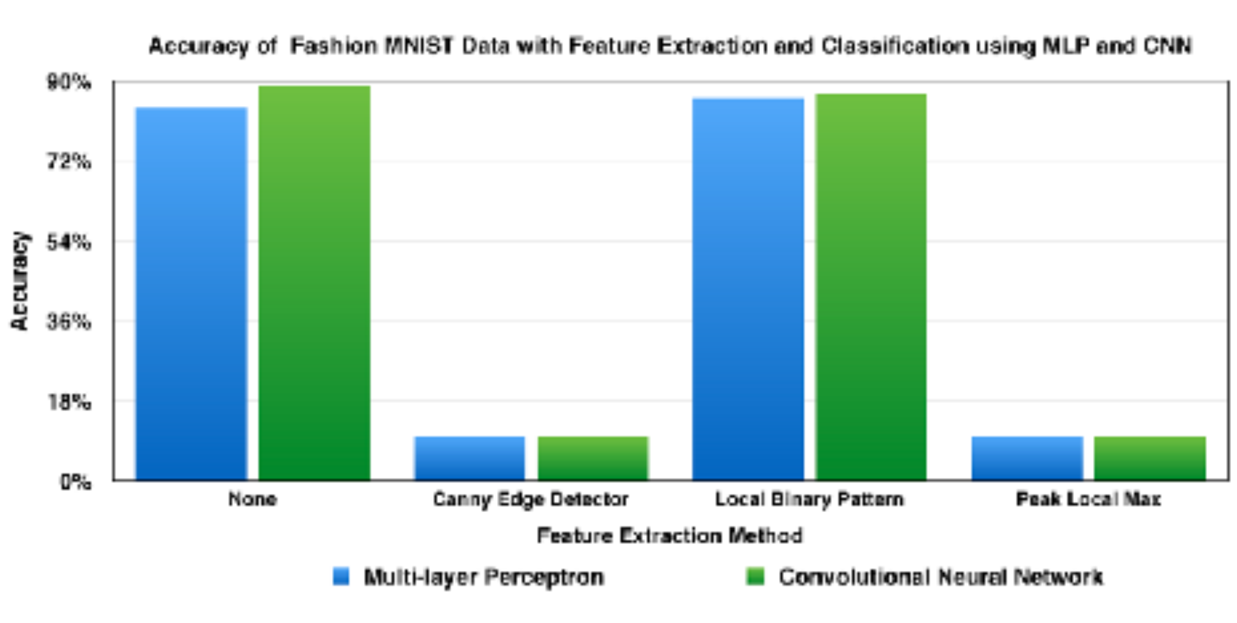


Figure 7: Accuracy of Fashion MNIST dataset using Different Feature Extraction Methods and MLP and CNN as Classifiers

Feature Extraction Method	Multi-layer Perceptron	Convolutional Neural Network
None	11	20
Canny Edge Detector	15	23
Local Binary Pattern	13	17
Peak Local Max	14	16

Table 4: Computation Time on Fashion MNIST dataset using Different Feature Extraction Methods and MLP and CNN as Classifiers

‘

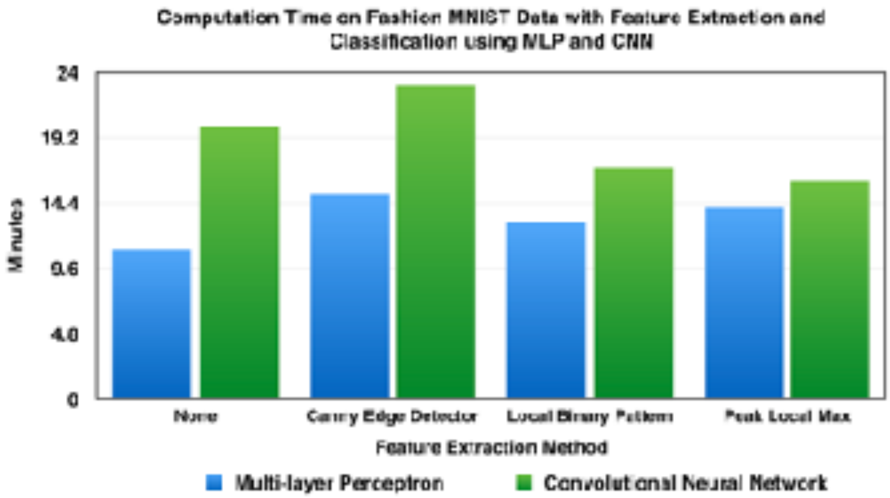


Figure 8: Computation Time on Fashion MNIST dataset using Different Feature Extraction Methods and MLP and CNN as Classifiers

Feature Extraction Method + Classifier	Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
None + MLP	98%	98%	92%	93%	82%	96%	95%	92%	97%	97%
Canny Edge Detector + MLP	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%
Local Binary Pattern + MLP	96%	97%	95%	95%	95%	94%	98%	94%	97%	90%
Peak Local Max + MLP	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%

Table 5: Accuracy of Each Class of MNIST dataset using Different Feature Extraction Methods and MLP as Classifier

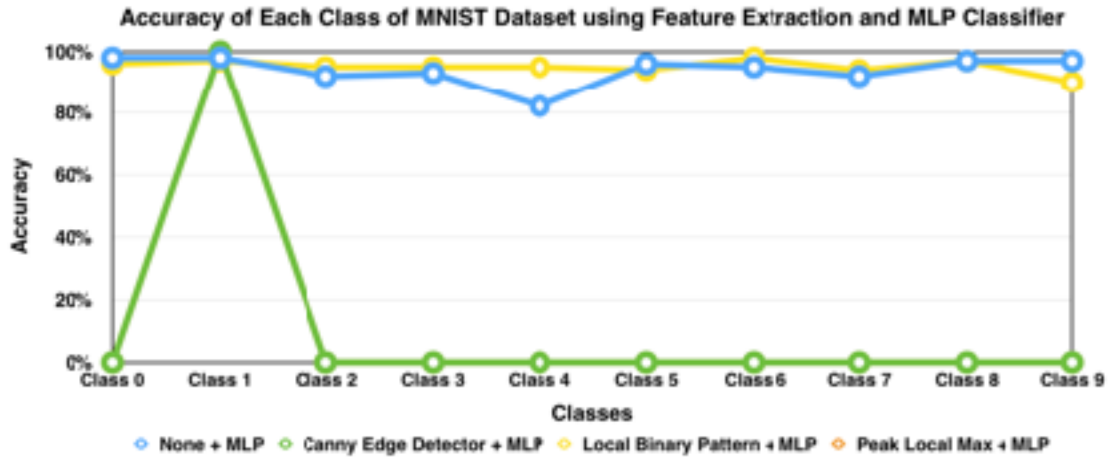


Figure 9: Accuracy of Each Class of MNIST dataset using Different Feature Extraction Methods and MLP as Classifier

Feature Extraction Method + Classifier	Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9
None + CNN	99%	98%	97%	98%	98%	98%	97%	97%	97%	95%
Canny Edge Detector + CNN	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%
Local Binary Pattern + CNN	98%	99%	98%	99%	98%	98%	98%	99%	99%	98%
Peak Local Max + CNN	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%

Table 6: Accuracy of Each Class of MNIST dataset using Different Feature Extraction Methods and CNN as Classifier

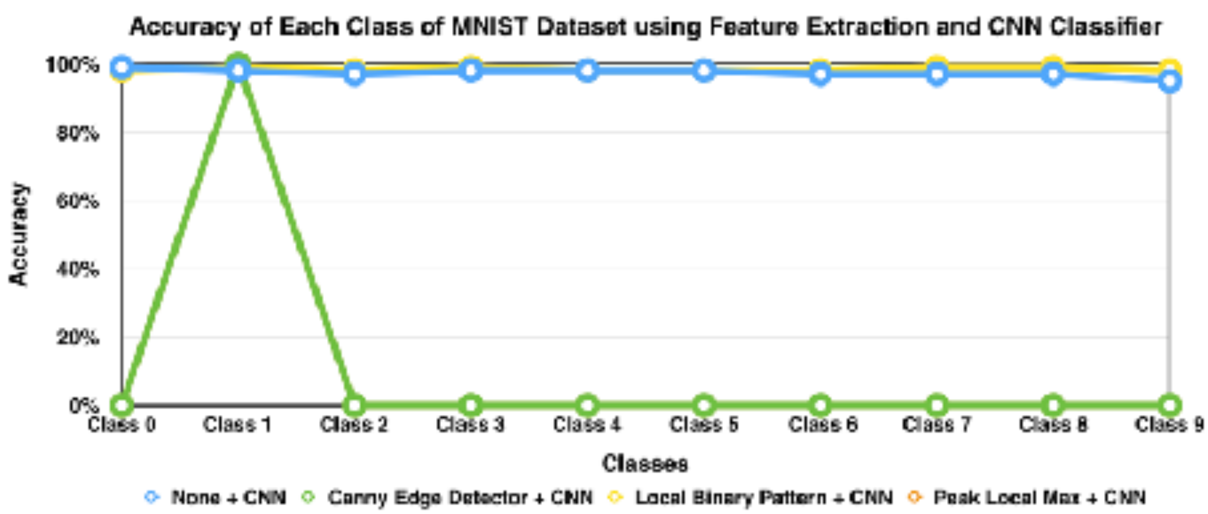


Figure 10: Accuracy of Each Class of MNIST dataset using Different Feature Extraction Methods and CNN as Classifier

Feature Extraction Method + Classifier	T-shirt	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle Boot
None + MLP	76%	96%	65%	86%	82%	92%	74%	94%	97%	95%
Canny Edge Detector + MLP	0%	0%	0%	0%	0%	0%	0%	100%	0%	0%
Local Binary Pattern + MLP	85%	93%	81%	93%	70%	96%	60%	95%	98%	95%
Peak Local Max + MLP	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%

Table 7: Accuracy of Each Class of Fashion MNIST dataset using Different Feature Extraction Methods and MLP as Classifier

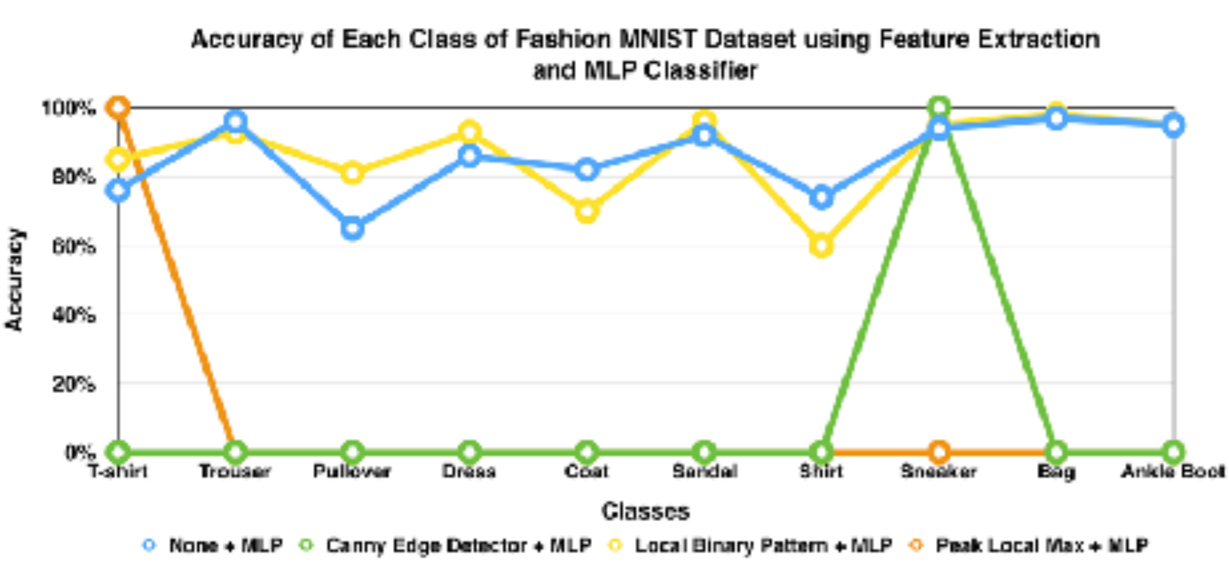


Figure 11: Accuracy of Each Class of Fashion MNIST dataset using Different Feature Extraction Methods and MLP as Classifier

Feature Extraction Method + Classifier	T-shirt	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle Boot
None + CNN	90%	96%	91%	89%	73%	96%	57%	96%	98%	96%
Canny Edge Detector + CNN	0%	0%	0%	100%	0%	0%	0%	0%	0%	0%
Local Binary Pattern + CNN	90%	97%	80%	89%	79%	96%	67%	97%	98%	96%
Peak Local Max + CNN	0%	0%	0%	0%	100%	0%	0%	0%	0%	0%

Table 8: Accuracy of Each Class of Fashion MNIST dataset using Different Feature Extraction Methods and CNN as Classifier

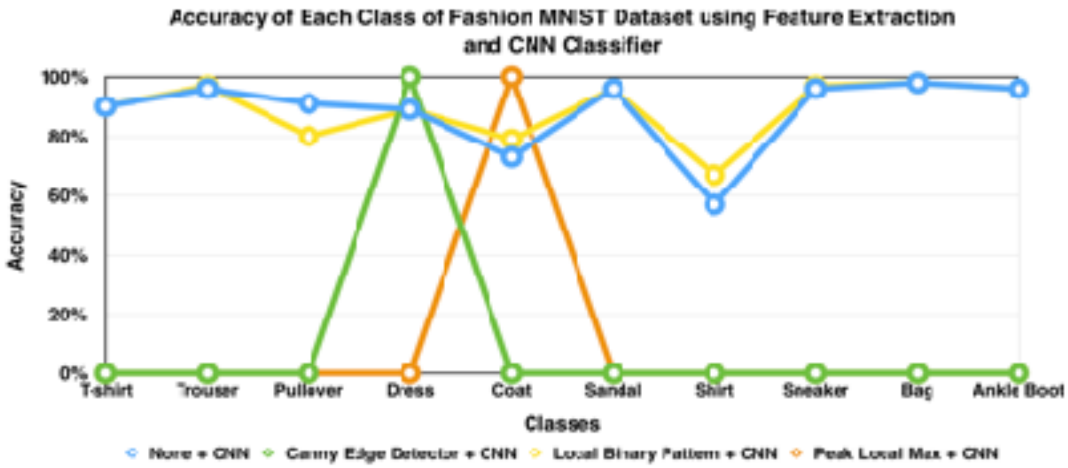


Figure 12: Accuracy of Each Class of Fashion MNIST dataset using Different Feature Extraction Methods and CNN as Classifier

CONCLUSION

From the above results, it can be observed that Classification using Local Binary Pattern Feature Extraction Method on both MNIST and Fashion MNIST dataset had an increased performance in comparison to Classification where no Feature Extraction Method was used.

The Computation time in classifying the datasets using Local Binary Pattern Feature Extraction Method was less for MNIST dataset using MLP as Classifier and for Fashion MNIST dataset using CNN as Classifier in comparison to Classification where no Feature Extraction Method was used.

The other two methods i.e. Canny Edge Detector and Peak Local Maxima performance was poor in comparison to Classification where no Feature Extraction Method was used.

The Computation time in classifying the datasets using Canny Edge Detector and Peak Local Maxima was less for Fashion MNIST dataset using CNN as Classifier in comparison to Classification where no Feature Extraction Method was used.

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
[9] Canny Edge Detector : http://scikit-image.org/docs/dev/auto_examples/edges/plot_canny.html
[10] Local Binary Pattern : http://scikit-image.org/docs/dev/auto_examples/features_detection/plot_local_binary_pattern.html
[11] Peak Local Max : http://scikit-image.org/docs/dev/api/skimage.feature.html#skimage.feature.peak_local_max

APPENDIX

CODE

1. Different Feature Extraction Methods with CNN as Classifier on 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
import numpy
from PIL import Image
from pylab import *
from skimage.feature import hog
from skimage import feature
from sklearn import datasets
import cv2
from skimage import exposure
import timeit
from skimage import transform as tf

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

# MNIST dataset
train_dataset = torchvision.datasets.MNIST(root='./data', train=True, download=True,
transform=trans)
test_dataset = torchvision.datasets.MNIST(root='./data', train=False, download=True,
transform=trans)

train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=10, shuffle=True,
num_workers=2)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_size=10, shuffle=False,
num_workers=2)

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(train_dataset.train_data[i].numpy(), cmap='gray')
    plt.title('Label : %i' % train_dataset.train_labels[i])
plt.show()

num_epochs = 20
total_step = len(train_loader)
Loss_1 = []
print('---------------------------------------------------')
start = timeit.default_timer()
```

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

    running_loss = 0.0
    for i, data in enumerate(train_loader, 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_1.append(running_loss / total_step)
print('Finished Training')
stop = timeit.default_timer()

correct = 0
total = 0
with torch.no_grad():
    for data in test_loader:
        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 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 test_loader:
        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]))

train_edge = torchvision.datasets.MNIST(root='./data', train=True, download=True,
transform=trans)
test_edge = torchvision.datasets.MNIST(root='./data', train=False, download=True,
transform=trans)

for i in range(0,len(train_edge)):
    image = feature.canny(train_edge.train_data[i].numpy()) * 1
    train_edge.train_data[i] = torch.from_numpy(image)

for i in range(0,len(test_edge)):
    image = feature.canny(test_edge.test_data[i].numpy()) * 1
    test_edge.test_data[i] = torch.from_numpy(image)

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(train_edge.train_data[i].numpy(), cmap='gray')
    plt.title('Label : %i' % train_edge.train_labels[i])
plt.show()

train_edge_loader = torch.utils.data.DataLoader(dataset=train_edge, batch_size=10,
shuffle=True, num_workers=2)
test_edge_loader = torch.utils.data.DataLoader(dataset=test_edge, batch_size=10, shuffle=False,
num_workers=2)

num_epochs = 20
total_step = len(train_edge_loader)
Loss_2 = []
```

```
print('--------------------------------------------------')
start_edge = timeit.default_timer()
for epoch in range(num_epochs):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(train_edge_loader, 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_2.append(running_loss / total_step)
print('Finished Training')
stop_edge = timeit.default_timer()

correct = 0
total = 0
with torch.no_grad():
    for data in test_edge_loader:
        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 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 test_edge_loader:
        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]))

train_lbp = torchvision.datasets.MNIST(root='./data', train=True, download=True,
transform=trans)
test_lbp = torchvision.datasets.MNIST(root='./data', train=False, download=True,
transform=trans)

for i in range(0,len(train_lbp)):
    image = feature.local_binary_pattern(train_lbp.train_data[i].numpy(), P = 100, R = 50)
    train_lbp.train_data[i] = torch.from_numpy(image)

for i in range(0,len(test_lbp)):
    image = feature.local_binary_pattern(test_lbp.test_data[i].numpy(), P = 100, R = 50)
    test_lbp.test_data[i] = torch.from_numpy(image)

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(train_lbp.train_data[i].numpy(), cmap='gray')
    plt.title('Label : %i' % train_lbp.train_labels[i])
plt.show()

train_lbp_loader = torch.utils.data.DataLoader(train_lbp, batch_size=10, shuffle=True,
num_workers=2)
test_lbp_loader = torch.utils.data.DataLoader(dataset=test_lbp, batch_size=10, shuffle=False,
num_workers=2)

num_epochs = 20
```

```
total_step = len(train_lbp_loader)
Loss_3 = []
print('--------------------------------------------------')
start_lbp = timeit.default_timer()
for epoch in range(num_epochs):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(train_lbp_loader, 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_3.append(running_loss / total_step)
print('Finished Training')
stop_lbp = timeit.default_timer()

correct = 0
total = 0
with torch.no_grad():
    for data in test_lbp_loader:
        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 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 test_lbp_loader:
        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]))

train_plm = torchvision.datasets.MNIST(root='./data', train=True, download=True,
transform=trans)
test_plm = torchvision.datasets.MNIST(root='./data', train=False, download=True,
transform=trans)

for i in range(0,len(train_plm)):
    image = feature.peak_local_max(train_plm.train_data[i].numpy(), indices = False) * 1
    train_plm.train_data[i] = torch.from_numpy(image)

for i in range(0,len(test_plm)):
    image = feature.peak_local_max(test_plm.test_data[i].numpy(), indices = False) * 1
    test_plm.test_data[i] = torch.from_numpy(image)

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(train_plm.train_data[i].numpy(), cmap='gray')
    plt.title('Label : %i' % train_lbp.train_labels[i])
plt.show()

train_plm_loader = torch.utils.data.DataLoader(train_plm, batch_size=10, shuffle=True,
num_workers=2)
test_plm_loader = torch.utils.data.DataLoader(dataset=test_plm, batch_size=10, shuffle=False,
num_workers=2)
```

```
num_epochs = 20
total_step = len(train_plm_loader)
Loss_4 = []
print('--------------------------------------------------')
start_plm = timeit.default_timer()
for epoch in range(num_epochs):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(train_plm_loader, 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_4.append(running_loss / total_step)
print('Finished Training')
stop_plm = timeit.default_timer()

correct = 0
total = 0
with torch.no_grad():
    for data in test_plm_loader:
        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 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 test_plm_loader:
        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(Loss_1, 'r--', label = "CNN")
plt.plot(Loss_2, 'b--', label = "Canny Edge detector + CNN")
plt.plot(Loss_3, 'g--', label = "Local Binary pattern + CNN")
plt.plot(Loss_4, 'y--', label = "Peak Local Max + CNN")
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()

print("Computation time for classifying the data without feature extraction : %f minutes" %
((stop-start)/60))
print("Computation time for classifying the data with Canny Edge Detector feature extraction
method : %f mintues" %((stop_edge-start_edge)/60))
print("Computation time for classifying the data with Local Binary pattern feature extraction
method : %f minutes" %((stop_lbp-start_lbp)/60))
print("Computation time for classifying the data with Peak Local Max feature extraction
method : %f minutes" %((stop_plm-start_plm)/60))
```

2. Different Feature Extraction Methods with MLP as Classifier on 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
import numpy
from PIL import Image
from pylab import *
from skimage.feature import hog
from skimage import feature
from sklearn import datasets
import cv2
from skimage import exposure
import timeit
from skimage import transform as tf

#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
```

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

# MNIST dataset
train_dataset = torchvision.datasets.MNIST(root='./data', train=True, download=True,
transform=trans)
test_dataset = torchvision.datasets.MNIST(root='./data', train=False, download=True,
transform=trans)

train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=10, shuffle=True,
num_workers=2)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_size=10, shuffle=False,
num_workers=2)

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(train_dataset.train_data[i].numpy(), cmap='gray')
    plt.title('Label : %i' % train_dataset.train_labels[i])
plt.show()

num_epochs = 20
total_step = len(train_loader)
Loss_1 = []
print('---------------------------------------------------')
start = timeit.default_timer()
for epoch in range(num_epochs):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(train_loader, 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')
stop = timeit.default_timer()

correct = 0
total = 0
with torch.no_grad():
    for data in test_loader:
        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 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 test_loader:
        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]))

train_edge = torchvision.datasets.MNIST(root='./data', train=True, download=True,
transform=trans)
```

```
test_edge = torchvision.datasets.MNIST(root='./data', train=False, download=True,
transform=trans)

for i in range(0,len(train_edge)):
    image = feature.canny(train_edge.train_data[i].numpy()) * 1
    train_edge.train_data[i] = torch.from_numpy(image)

for i in range(0,len(test_edge)):
    image = feature.canny(test_edge.test_data[i].numpy()) * 1
    test_edge.test_data[i] = torch.from_numpy(image)

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(train_edge.train_data[i].numpy(), cmap='gray')
    plt.title('Label : %i' % train_edge.train_labels[i])
plt.show()

train_edge_loader = torch.utils.data.DataLoader(dataset=train_edge, batch_size=10,
shuffle=True, num_workers=2)
test_edge_loader = torch.utils.data.DataLoader(dataset=test_edge, batch_size=10, shuffle=False,
num_workers=2)

num_epochs = 20
total_step = len(train_edge_loader)
Loss_2 = []
print('--------------------------------------------------')
start_edge = timeit.default_timer()
for epoch in range(num_epochs):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(train_edge_loader, 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')
stop_edge = timeit.default_timer()

correct = 0
total = 0
with torch.no_grad():
    for data in test_edge_loader:
        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 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 test_edge_loader:
        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]))
```

```
train_lbp = torchvision.datasets.MNIST(root='./data', train=True, download=True,
transform=trans)
test_lbp = torchvision.datasets.MNIST(root='./data', train=False, download=True,
transform=trans)

for i in range(0,len(train_lbp)):
    image = feature.local_binary_pattern(train_lbp.train_data[i].numpy(), P = 100, R = 50)
    train_lbp.train_data[i] = torch.from_numpy(image)

for i in range(0,len(test_lbp)):
    image = feature.local_binary_pattern(test_lbp.test_data[i].numpy(), P = 100, R = 50)
    test_lbp.test_data[i] = torch.from_numpy(image)

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(train_lbp.train_data[i].numpy(), cmap='gray')
    plt.title('Label : %i' % train_lbp.train_labels[i])
plt.show()

train_lbp_loader = torch.utils.data.DataLoader(train_lbp, batch_size=10, shuffle=True,
num_workers=2)
test_lbp_loader = torch.utils.data.DataLoader(dataset=test_lbp, batch_size=10, shuffle=False,
num_workers=2)

num_epochs = 20
total_step = len(train_lbp_loader)
Loss_3 = []
print('--------------------------------------------------')
start_lbp = timeit.default_timer()
for epoch in range(num_epochs):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(train_lbp_loader, 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')
stop_lbp = timeit.default_timer()

correct = 0
total = 0
with torch.no_grad():
    for data in test_lbp_loader:
        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 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 test_lbp_loader:
        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]))

train_plm = torchvision.datasets.MNIST(root='./data', train=True, download=True,
transform=trans)
test_plm = torchvision.datasets.MNIST(root='./data', train=False, download=True,
transform=trans)

for i in range(0,len(train_plm)):
    image = feature.peak_local_max(train_plm.train_data[i].numpy(), indices = False) * 1
    train_plm.train_data[i] = torch.from_numpy(image)

for i in range(0,len(test_plm)):
    image = feature.peak_local_max(test_plm.test_data[i].numpy(), indices = False) * 1
    test_plm.test_data[i] = torch.from_numpy(image)

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(train_plm.train_data[i].numpy(), cmap='gray')
    plt.title('Label : %i' % train_lbp.train_labels[i])
plt.show()

train_plm_loader = torch.utils.data.DataLoader(train_plm, batch_size=10, shuffle=True,
num_workers=2)
test_plm_loader = torch.utils.data.DataLoader(dataset=test_plm, batch_size=10, shuffle=False,
num_workers=2)

num_epochs = 20
total_step = len(train_plm_loader)
Loss_4 = []
print('----------------------------------------------------')
start_plm = timeit.default_timer()
for epoch in range(num_epochs):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(train_plm_loader, 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')
stop_plm = timeit.default_timer()

correct = 0
total = 0
with torch.no_grad():
    for data in test_plm_loader:
        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 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 test_plm_loader:
        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 = "MLP")
plt.plot(Loss_2, 'b--', label = "Canny Edge detector + MLP")
plt.plot(Loss_3, 'g--', label = "Local Binary pattern + MLP")
plt.plot(Loss_4, 'y--', label = "Peak Local Max + MLP")
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()

print("Computation time for classifying the data without feature extraction : %f minutes" %
((stop-start)/60))
print("Computation time for classifying the data with Canny Edge Detector feature extraction
method : %f mintues" %((stop_edge-start_edge)/60))
print("Computation time for classifying the data with Local Binary pattern feature extraction
method : %f minutes" %((stop_lbp-start_lbp)/60))
print("Computation time for classifying the data with Peak Local Max feature extraction
method : %f minutes" %((stop_plm-start_plm)/60))
```

3. Different Feature Extraction Methods with MLP as Classifier 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
import numpy
from PIL import Image
from pylab import *
from skimage.feature import hog
```

```
from skimage import feature
from sklearn import datasets
import cv2
from skimage import exposure
import timeit
from skimage import transform as tf

#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
train_dataset = datasets.FashionMNIST(root='./fmnist/', train=True, download=True,
transform=transform)
test_dataset = datasets.FashionMNIST(root='./fmnist/', train=False, download=True,
transform=transform)
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=10, shuffle=True,
num_workers=2)
```

```
test_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_size=10, shuffle=False,
num_workers=2)

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(train_dataset.train_data[i].numpy(), cmap='gray')
   plt.title('Label : %s' % labels_map[train_dataset.train_labels[i]])
plt.show()

num_epochs = 20
total_step = len(train_loader)
Loss_1 = []
print('--------------------------------------------------')
start = timeit.default_timer()
for epoch in range(num_epochs):  # loop over the dataset multiple times

   running_loss = 0.0
   for i, data in enumerate(train_loader, 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')
```

```
stop = timeit.default_timer()

correct = 0
total = 0
with torch.no_grad():
    for data in test_loader:
        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 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 test_loader:
        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]))

train_edge = datasets.FashionMNIST(root='./fmnist/', train=True, download=True,
transform=transform)
test_edge = datasets.FashionMNIST(root='./fmnist/', train=False, download=True,
transform=transform)

for i in range(0,len(train_edge)):
    image = feature.canny(train_edge.train_data[i].numpy()) * 1
    train_edge.train_data[i] = torch.from_numpy(image)

for i in range(0,len(test_edge)):
    image = feature.canny(test_edge.test_data[i].numpy()) * 1
    test_edge.test_data[i] = torch.from_numpy(image)
```

```
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(train_edge.train_data[i].numpy(), cmap='gray')
    plt.title('Label : %s' % labels_map[train_edge.train_labels[i]])
plt.show()

train_edge_loader = torch.utils.data.DataLoader(dataset=train_edge, batch_size=10,
shuffle=True, num_workers=2)
test_edge_loader = torch.utils.data.DataLoader(dataset=test_edge, batch_size=10, shuffle=False,
num_workers=2)

num_epochs = 20
total_step = len(train_edge_loader)
Loss_2 = []
print('--------------------------------------------------')
start_edge = timeit.default_timer()
for epoch in range(num_epochs):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(train_edge_loader, 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')
```

```
stop_edge = timeit.default_timer()

correct = 0
total = 0
with torch.no_grad():
    for data in test_edge_loader:
        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 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 test_edge_loader:
        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]))

train_lbp = datasets.FashionMNIST(root='./fmnist/', train=True, download=True,
transform=transform)
test_lbp = datasets.FashionMNIST(root='./fmnist/', train=False, download=True,
transform=transform)

for i in range(0,len(train_lbp)):
    image = feature.local_binary_pattern(train_lbp.train_data[i].numpy(), P = 100, R = 50)
    train_lbp.train_data[i] = torch.from_numpy(image)

for i in range(0,len(test_lbp)):
    image = feature.local_binary_pattern(test_lbp.test_data[i].numpy(), P = 100, R = 50)
    test_lbp.test_data[i] = torch.from_numpy(image)
```

```
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(train_lbp.train_data[i].numpy(), cmap='gray')
    plt.title('Label : %s' % labels_map[train_lbp.train_labels[i]])
plt.show()

train_lbp_loader = torch.utils.data.DataLoader(train_lbp, batch_size=10, shuffle=True,
num_workers=2)
test_lbp_loader = torch.utils.data.DataLoader(dataset=test_lbp, batch_size=10, shuffle=False,
num_workers=2)

num_epochs = 20
total_step = len(train_lbp_loader)
Loss_3 = []
print('--------------------------------------------------')
start_lbp = timeit.default_timer()
for epoch in range(num_epochs):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(train_lbp_loader, 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')
```

```
stop_lbp = timeit.default_timer()

correct = 0
total = 0
with torch.no_grad():
    for data in test_lbp_loader:
        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 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 test_lbp_loader:
        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]))

train_plm = datasets.FashionMNIST(root='./fmnist/', train=True, download=True,
transform=transform)
test_plm = datasets.FashionMNIST(root='./fmnist/', train=False, download=True,
transform=transform)

for i in range(0,len(train_plm)):
    image = feature.peak_local_max(train_plm.train_data[i].numpy(), indices = False) * 1
    train_plm.train_data[i] = torch.from_numpy(image)

for i in range(0,len(test_plm)):
    image = feature.peak_local_max(test_plm.test_data[i].numpy(), indices = False) * 1
    test_plm.test_data[i] = torch.from_numpy(image)
```

```
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(train_plm.train_data[i].numpy(), cmap='gray')
    plt.title('Label : %s' % labels_map[train_lbp.train_labels[i]])
plt.show()

train_plm_loader = torch.utils.data.DataLoader(train_plm, batch_size=10, shuffle=True,
num_workers=2)
test_plm_loader = torch.utils.data.DataLoader(dataset=test_plm, batch_size=10, shuffle=False,
num_workers=2)

num_epochs = 20
total_step = len(train_plm_loader)
Loss_4 = []
print('--------------------------------------------------')
start_plm = timeit.default_timer()
for epoch in range(num_epochs):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(train_plm_loader, 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')
```

```
stop_plm = timeit.default_timer()

correct = 0
total = 0
with torch.no_grad():
    for data in test_plm_loader:
        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 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 test_plm_loader:
        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_1, 'r--', label = "MLP")
plt.plot(Loss_2, 'b--', label = "Canny Edge detector + MLP")
plt.plot(Loss_3, 'g--', label = "Local Binary pattern + MLP")
plt.plot(Loss_4, 'y--', label = "Peak Local Max + MLP")
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()
```

```
print("Computation time for classifying the data without feature extraction : %f minutes" % ((stop-start)/60))
print("Computation time for classifying the data with Canny Edge Detector feature extraction method : %f mintues" %((stop_edge-start_edge)/60))
print("Computation time for classifying the data with Local Binary pattern feature extraction method : %f minutes" %((stop_lbp-start_lbp)/60))
print("Computation time for classifying the data with Peak Local Max feature extraction method : %f minutes" %((stop_plm-start_plm)/60))
```

4. Different Feature Extraction Methods with CNN as Classifier 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
import numpy
from PIL import Image
from pylab import *
from skimage.feature import hog
from skimage import feature
from sklearn import datasets
import cv2
from skimage import exposure
import timeit
from skimage import transform as tf

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

# 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
train_dataset = datasets.FashionMNIST(root='./fmnist/', train=True, download=True,
transform=transform)
test_dataset = datasets.FashionMNIST(root='./fmnist/', train=False, download=True,
transform=transform)
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=10, shuffle=True,
num_workers=2)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_size=10, shuffle=False,
num_workers=2)

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(train_dataset.train_data[i].numpy(), cmap='gray')
    plt.title('Label : %s' % labels_map[train_dataset.train_labels[i]])
plt.show()

num_epochs = 20
total_step = len(train_loader)
Loss_1 = []
print('--------------------------------------------------')
start = timeit.default_timer()
for epoch in range(num_epochs):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(train_loader, 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_1.append(running_loss / total_step)
print('Finished Training')
stop = timeit.default_timer()

correct = 0
total = 0
with torch.no_grad():
    for data in test_loader:
```

```
        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 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 test_loader:
        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]))

train_edge = datasets.FashionMNIST(root='./fmnist/', train=True, download=True,
transform=transform)
test_edge = datasets.FashionMNIST(root='./fmnist/', train=False, download=True,
transform=transform)

for i in range(0,len(train_edge)):
    image = feature.canny(train_edge.train_data[i].numpy()) * 1
    train_edge.train_data[i] = torch.from_numpy(image)

for i in range(0,len(test_edge)):
    image = feature.canny(test_edge.test_data[i].numpy()) * 1
    test_edge.test_data[i] = torch.from_numpy(image)

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(train_edge.train_data[i].numpy(), cmap='gray')
    plt.title('Label : %s' % labels_map[train_edge.train_labels[i]])
plt.show()

train_edge_loader = torch.utils.data.DataLoader(dataset=train_edge, batch_size=10,
shuffle=True, num_workers=2)
test_edge_loader = torch.utils.data.DataLoader(dataset=test_edge, batch_size=10, shuffle=False,
num_workers=2)

num_epochs = 20
total_step = len(train_edge_loader)
Loss_2 = []
print('---------------------------------------------------')
start_edge = timeit.default_timer()
for epoch in range(num_epochs):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(train_edge_loader, 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_2.append(running_loss / total_step)
print('Finished Training')
stop_edge = timeit.default_timer()

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

```
    for data in test_edge_loader:
        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 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 test_edge_loader:
        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]))

train_lbp = datasets.FashionMNIST(root='./fmnist/', train=True, download=True,
transform=transform)
test_lbp = datasets.FashionMNIST(root='./fmnist/', train=False, download=True,
transform=transform)

for i in range(0,len(train_lbp)):
    image = feature.local_binary_pattern(train_lbp.train_data[i].numpy(), P = 100, R = 50)
    train_lbp.train_data[i] = torch.from_numpy(image)

for i in range(0,len(test_lbp)):
    image = feature.local_binary_pattern(test_lbp.test_data[i].numpy(), P = 100, R = 50)
    test_lbp.test_data[i] = torch.from_numpy(image)

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(train_lbp.train_data[i].numpy(), cmap='gray')
    plt.title('Label : %s' % labels_map[train_lbp.train_labels[i]])
plt.show()

train_lbp_loader = torch.utils.data.DataLoader(train_lbp, batch_size=10, shuffle=True,
num_workers=2)
test_lbp_loader = torch.utils.data.DataLoader(dataset=test_lbp, batch_size=10, shuffle=False,
num_workers=2)

num_epochs = 20
total_step = len(train_lbp_loader)
Loss_3 = []
print('----------------------------------------------------')
start_lbp = timeit.default_timer()
for epoch in range(num_epochs):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(train_lbp_loader, 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_3.append(running_loss / total_step)
print('Finished Training')
stop_lbp = timeit.default_timer()

correct = 0
total = 0
```

```
with torch.no_grad():
    for data in test_lbp_loader:
        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 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 test_lbp_loader:
        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]))
train_plm = datasets.FashionMNIST(root='./fmnist/', train=True, download=True,
transform=transform)
test_plm = datasets.FashionMNIST(root='./fmnist/', train=False, download=True,
transform=transform)

for i in range(0,len(train_plm)):
    image = feature.peak_local_max(train_plm.train_data[i].numpy(), indices = False) * 1
    train_plm.train_data[i] = torch.from_numpy(image)

for i in range(0,len(test_plm)):
    image = feature.peak_local_max(test_plm.test_data[i].numpy(), indices = False) * 1
    test_plm.test_data[i] = torch.from_numpy(image)


train_plm_loader = torch.utils.data.DataLoader(train_plm, batch_size=10, shuffle=True,
num_workers=2)
```

```
test_plm_loader = torch.utils.data.DataLoader(dataset=test_plm, batch_size=10, shuffle=False,
num_workers=2)
num_epochs = 20
total_step = len(train_plm_loader)
Loss_4 = []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(train_plm.train_data[i].numpy(), cmap='gray')
    plt.title('Label : %s' % labels_map[train_lbp.train_labels[i]])
plt.show()

print('----------------------------------------------------')
start_plm = timeit.default_timer()
for epoch in range(num_epochs):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(train_plm_loader, 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_4.append(running_loss / total_step)
print('Finished Training')
stop_plm = timeit.default_timer()

correct = 0
total = 0
```

```
with torch.no_grad():
    for data in test_plm_loader:
        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 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 test_plm_loader:
        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_1, 'r--', label = "CNN")
plt.plot(Loss_2, 'b--', label = "Canny Edge detector + CNN")
plt.plot(Loss_3, 'g--', label = "Local Binary pattern + CNN")
plt.plot(Loss_4, 'y--', label = "Peak Local Max + CNN")
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()

print("Computation time for classifying the data without feature extraction : %f minutes" %
((stop-start)/60))
print("Computation time for classifying the data with Canny Edge Detector feature extraction
method : %f mintues" %((stop_edge-start_edge)/60))
```

```
print("Computation time for classifying the data with Local Binary pattern feature extraction method : %f minutes" %((stop_lbp-start_lbp)/60))
print("Computation time for classifying the data with Peak Local Max feature extraction method : %f minutes" %((stop_plm-start_plm)/60))
```