Exercise 3: : Convolutional Neural Networks on the Olivetti faces dataset

```
from sklearn.model_selection import train_test_split
from torch.utils.data import Dataset, DataLoader
from sklearn.datasets import fetch_olivetti_faces
import numpy as np
import torch
#  Downloading olivetti faces dataset
olivetti = fetch_olivetti_faces()
train = olivetti.images
label = olivetti.target
#Splitting train and test set
X = train
Y = label
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.1, stratify=Y)
np.unique(y_train, return_counts=True)
np.unique(y_test, return_counts=True)
```

```
trainX_loader = DataLoader(X_train)
trainY_loader = DataLoader(y_train)
testX_loader = DataLoader(X_test)
testY_loader = DataLoader(y_test)
```

```
downloading Olivetti faces from https://ndownloader.figshare.com/files/5976027 to /root,
```

Train set constitutes 90% of original dataset

Test set constitutes 10% of original dataset

Len of train set is 360 and len of test set is 40

Structure

Inputs image of size 64*64 is given to a convolutional layer that gives out 10 feature maps by convolving the image and 10 filters of size 5*5

Max pooling - size 2

Output from max pooling layer is flattened

Flattened output is fed in to two fully connected layers giving output of dimension 40

```
import torch.nn as nn
import torch.nn.functional as F
```

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, 5)
        self.pool = nn.MaxPool2d(2, 2)

        self.fc1 = nn.Linear(9000, 5000)
        self.fc2 = nn.Linear(5000, 40)


    def forward(self, r):
        r = torch.reshape(r,[1,1,64,64])
        r = self.pool(F.relu(self.conv1(r)))
        r = r.view(-1,9000)
        r = F.relu(self.fc1(r))
        r = self.fc2(r)

        return r

net = Net()
```

```
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

```
pip install mpi4py
```

```
Collecting mpi4py
  Downloading https://files.pythonhosted.org/packages/ec/8f/bbd8de5ba566dd77e408d8136e2b
    |████████████████████████████████| 1.4MB 2.8MB/s
Building wheels for collected packages: mpi4py
  Building wheel for mpi4py (setup.py) ... done
  Created wheel for mpi4py: filename=mpi4py-3.0.3-cp36-cp36m-linux_x86_64.whl size=2074
  Stored in directory: /root/.cache/pip/wheels/18/e0/86/2b713dd512199096012ceca61429e12b
Successfully built mpi4py
Installing collected packages: mpi4py
Successfully installed mpi4py-3.0.3
```

```
from mpi4py import MPI
print(MPI.Wtime())
```

```
10713.159903395
```

Training

Loss used: Cross entropy loss

Number of epochs : 4

```
def custom_loss_crossentropy(x,y):
    log_prob = -1.0 * F.log_softmax(x, 1)
    loss = log_prob.gather(1, y.unsqueeze(1))
    loss = loss.mean()
    return loss
def custom_loss_mse(output, target):
    loss = torch.mean((output - target)**2)
    return loss
i=0
test_accuracy = []
train_accuracy = []
start = MPI.Wtime()
for epoch in range(4):

    running_loss = 0.0

    for inputs, labels in zip(trainX_loader, trainY_loader):

        # zeroing  gradients
        optimizer.zero_grad()
        #feeding inputs to the network
        outputs = net(inputs)
        loss= criterion(outputs, labels.long())
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        if i % 2000 == 1999:

            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))

            running_loss = 0.0
        i+=1
    total = 0
    correct = 0
    for images,labels in zip(trainX_loader,trainY_loader):

        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        #         _, predicted2 = torch.max(outputs[1].data, 1)
        total += labels.size(0)
        correct += (predicted == labels ).sum().item()
    train_accuracy.append(100 * correct / total)
    print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
    total = 0
    correct = 0
    for images,labels in zip(testX_loader,testY_loader):

        outputs = net(images)
```

```
        _, predicted = torch.max(outputs.data, 1)
        #   _, predicted2 = torch.max(outputs[1].data, 1)
        total += labels.size(0)
        correct += (predicted == labels ).sum().item()
    test_accuracy.append(100 * correct / total)


print('Finished Training')
totaltime = MPI.Wtime()-start
```

```
[1,   361] loss: 0.003
[2,   721] loss: 0.001
[3,  1081] loss: 0.001
[4,  1441] loss: 0.001
Finished Training
```

```
print(train_accuracy, test_accuracy, totaltime)
```

```
[100.0, 100.0, 100.0, 100.0] [[90.  90.  90.  87.5]] 546.3072238129998
```

Time taken: 546 seconds

Train accuracy : [100.0, 100.0, 100.0, 100.0]

Test accuracy : [90. 90. 90. 87.5]

```
correct = 0
total = 0
with torch.no_grad():
    for images,labels in zip(testX_loader,testY_loader):

        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
#         _, predicted2 = torch.max(outputs[1].data, 1)
        total += labels.size(0)
        correct += (predicted == labels ).sum().item()

print('Accuracy : %d %%' % (
    100 * correct / total))
```

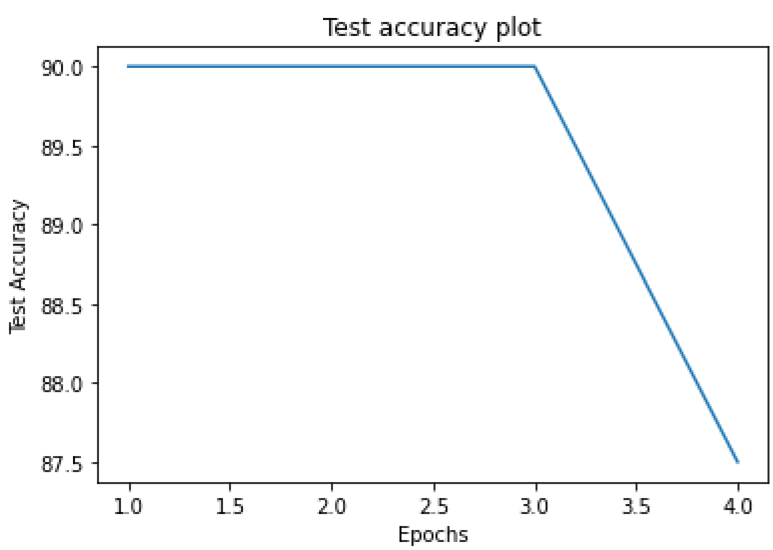
```
Accuracy : 87 %
```

```
print(test_accuracy)
print(train_accuracy)
print(totaltime)
```

```
[92.5, 90.0]
[99.44444444444444, 99.44444444444444]
265.29969947300015
```

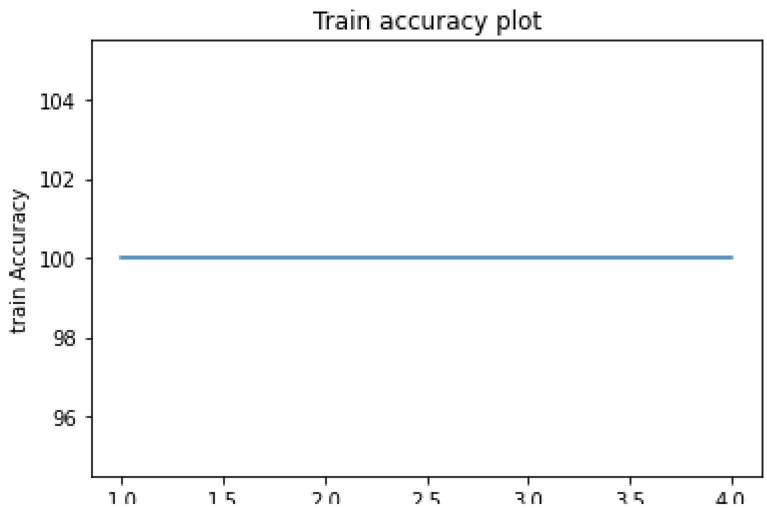
PLot for test accuracy

```
import matplotlib.pyplot as plt
epochs = [1,2,3,4]

test_acc = [90. , 90. , 90. , 87.5]
train_acc = [100.0, 100.0, 100.0, 100.0]
plt.plot(epochs,test_acc)
plt.ylabel('Test Accuracy')
plt.xlabel('Epochs')
plt.title('Test accuracy plot')
plt.show()
```



Plot for train accuracy

```
import matplotlib.pyplot as plt
epochs = [1,2,3,4]

# test_acc = [90. , 90. , 90. , 87.5]
train_acc = [100.0, 100.0, 100.0, 100.0]
plt.plot(epochs,train_acc)
plt.ylabel('train Accuracy')
plt.xlabel('Epochs')
plt.title('Train accuracy plot')
plt.show()
```



Number of trainable parameters: In the first layer the input is convolved with 10 filters each of size 5X5

Number of params in the first layer: 10*5*5

Second layer: 9000*5000(weights)+5000(bias)

Output layer: 5000*40(weights)+40(bias) Total trainable parameters = 45205290.

Time taken is higher for CNN as it takes 546 seconds

Model complexity is also higher as the number of trainable parameters is 45205290, which is 4 times greater than ex1 and 2.

But the accuracy has increased by 43.5 times. The accuracy for CNN is 87%. Because there is not information loss as we did not flatten the input imags, accuracy is considerably high.