Lab-05 (a)

Aim: To implement AlexNet architecture

```
!pip install torchmetrics

!unzip /content/Face-Images

!find . -name "*.DS_Store" -type f -delete

import torch
import torch.nn as nn
from torchvision.transforms import ToTensor, transforms
from torch.utils.data import DataLoader
from torch.utils.data import Dataset
from PIL import Image
import os

class CustomDataset(Dataset):
    def __init__(self, root_dir, transform=None):
        self.root_dir = root_dir
        self.transform = transform
        self.classes = sorted(os.listdir(root_dir))

    def __len__(self):
        return sum(len(files) for _, _, files in
os.walk(self.root_dir))

    def __getitem__(self, idx):
        class_idx = 0
        while idx >= len(os.listdir(os.path.join(self.root_dir,
self.classes[class_idx]))):
            idx -= len(os.listdir(os.path.join(self.root_dir,
self.classes[class_idx])))
            class_idx += 1
        class_dir = os.path.join(self.root_dir,
self.classes[class_idx])
        file_name = os.listdir(class_dir)[idx]
        image = Image.open(os.path.join(class_dir, file_name))
        if self.transform:
            image = self.transform(image)
        return image, class_idx


transform = transforms.Compose([
    transforms.Resize((227,227)),
    transforms.ToTensor()
])
```

```
traindataset = CustomDataset(root_dir='/content/Face Images/Final
Training Images', transform=transform)
testdataset = CustomDataset(root_dir='/content/Face Images/Final
Testing Images', transform=transform)
train = DataLoader(traindataset, batch_size=32, shuffle=True)
test = DataLoader(testdataset, batch_size=32, shuffle=True)

class AlexNet(nn.Module):
    def __init__(self, num_classes):
        super(AlexNet, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=11, stride=4, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(64, 192, kernel_size=5, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(192, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
        )
        self.avgpool = nn.AdaptiveAvgPool2d((6, 6))
        self.classifier = nn.Sequential(
            nn.Dropout(),
            nn.Linear(256 * 6 * 6, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Linear(4096, num_classes),
        )

    def forward(self, x):
        x = self.features(x)
        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        return x

import torchmetrics
from tqdm.auto import tqdm
import torch.optim as optim
from torchmetrics.classification import MulticlassAccuracy,
MulticlassConfusionMatrix
```

```python
AlexNetModel=AlexNet(16)
lossfn=nn.CrossEntropyLoss()
optimizer=optim.SGD(params=AlexNetModel.parameters(),
                    lr=0.001,
                    momentum=0.9)
accuracy=torchmetrics.classification.Accuracy(task='multiclass',num_cl
asses=16)

epochs=25

device=torch.device("cuda" if torch.cuda.is_available() else "cpu")
num_epochs = 20
for epoch in range(epochs):

    running_loss = 0.0
    for i, data in enumerate(train, 0):
        inputs, labels = data[0].to(device), data[1].to(device)

        optimizer.zero_grad()
        AlexNetModel.train()
        outputs = AlexNetModel(inputs)
        loss = lossfn(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()

        if i % 10==0:
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 100))
            running_loss = 0.0


print("finished training")
```

```
[1,     1] loss: 0.028
[2,     1] loss: 0.028
[3,     1] loss: 0.028
[4,     1] loss: 0.028
[5,     1] loss: 0.028
[6,     1] loss: 0.028
[7,     1] loss: 0.028
[8,     1] loss: 0.028
[9,     1] loss: 0.028
[10,     1] loss: 0.028
[11,     1] loss: 0.028
[12,     1] loss: 0.028
[13,     1] loss: 0.028
[14,     1] loss: 0.028
[15,     1] loss: 0.028
```

```
[16,     1] loss: 0.028
[17,     1] loss: 0.028
[18,     1] loss: 0.028
[19,     1] loss: 0.028
[20,     1] loss: 0.028
[21,     1] loss: 0.028
[22,     1] loss: 0.028
[23,     1] loss: 0.028
[24,     1] loss: 0.028
[25,     1] loss: 0.028
finished training
```

```
AlexNetModel.eval()
for data in test:
  inputs, labels = data
  outputs = AlexNetModel(inputs)
  loss = lossfn(outputs, labels)
  accuracy.update(outputs, labels)
```

```
print(f'Accuracy on test set: {accuracy.compute()*100}')
```

```
Accuracy on test set: 6.25
```

```
torch.save(AlexNetModel.state_dict(), '/content/model_weights.pth')
```