Assignment # 2

Group Name: Snipe
Group Members: l19-1196 & l19-0963
section: 6H
To: sir Mubashir Baigh

Single level perceptron

```
Processed 5000/5400
Classification starting....
                precision    recall  Accuracy time(s)

        happy      0.44       0.67       0.53    102
        sad        0.43       0.63       0.46    110.5
        angry      0.93       0.42       0.86    197
        laughing   0.90       0.39       0.45    87
        confused   0.87       0.27       0.50    132
        average Time 120 (sec)
```

Multi-level perceptron

```
Processed 5000/5400
Classification starting....
                precision    recall  Accuracy time(s)

        happy      0.63       0.93       0.83    200
        sad        0.54       0.87       0.89    201.5
        angry      0.87       0.67       0.71    245
        laughing   0.88       0.46       0.69    197
        confused   0.89       0.64       0.93    150
        average Time 202 (sec)
```

Procedure

1. Data was put in one drive and its link was used to access the data
2. correct data was filtered e.g only images which were 100 x 100
3. mean and standard deviation was calculated of each category
4. Data was transformed by ToTensor() and normalized around mean/std
5. once the model was trained with above given accuracy, an image was given to classify it.

6 10 epochs are done to train on the dataset.
7 we tested each category separately as data given above.
8 below is given some important part of the code

Implementation

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torch.utils.data as data

import torchvision.transforms as transforms
import torchvision.datasets as datasets


import matplotlib.pyplot as plt
import numpy as np

import copy
import random
import time
```

To ensure we get reproducible results we set the random seed for Python, Numpy and PyTorch.

```
SEED = 5

random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
torch.cuda.manual_seed(SEED)
torch.backends.cudnn.deterministic = True

ROOT = '.data'

train_data = datasets.MNIST(root=ROOT,
train=True,
download=True)

since values range from 0-255 (value of a pixcel

mean = train_data.data.float().mean() / 255
std = train_data.data.float().std() / 255

train_transforms = transforms.Compose([
transforms.RandomRotation(5, fill=(0,)),
transforms.RandomCrop(28, padding=2),
```

```
transforms.ToTensor(),
transforms.Normalize(mean=[mean], std=[std])
])

test_transforms = transforms.Compose([
transforms.ToTensor(),
transforms.Normalize(mean=[mean], std=[std])
])


train_data, valid_data = data.random_split(train_data,
[n_train_examples, n_valid_examples])

print(f'Number of training examples: {len(train_data)}')
print(f'Number of validation examples: {len(valid_data)}')
print(f'Number of testing examples: {len(test_data)}')

BATCH_SIZE = 64

train_iterator = data.DataLoader(train_data,
shuffle=True,
batch_size=BATCH_SIZE)

valid_iterator = data.DataLoader(valid_data,
batch_size=BATCH_SIZE)

test_iterator = data.DataLoader(test_data,
batch_size=BATCH_SIZE)

def calculate_accuracy(y_pred, y):
top_pred = y_pred.argmax(1, keepdim=True)
correct = top_pred.eq(y.view_as(top_pred)).sum()
acc = correct.float() / y.shape[0]
return acc

def train(model, iterator, optimizer, criterion, device):

epoch_loss = 0
epoch_acc = 0

model.train()

for (x, y) in tqdm(iterator, desc="Training", leave=False):

x = x.to(device)
y = y.to(device)
```

```
optimizer.zero_grad()

y_pred, _ = model(x)

loss = criterion(y_pred, y)

acc = calculate_accuracy(y_pred, y)

loss.backward()

optimizer.step()

epoch_loss += loss.item()
epoch_acc += acc.item()

return epoch_loss / len(iterator), epoch_acc / len(iterator)


def evaluate(model, iterator, criterion, device):

epoch_loss = 0
epoch_acc = 0

model.eval()

with torch.no_grad():

for (x, y) in tqdm(iterator, desc="Evaluating", leave=False):

x = x.to(device)
y = y.to(device)

y_pred, _ = model(x)

loss = criterion(y_pred, y)

acc = calculate_accuracy(y_pred, y)

epoch_loss += loss.item()
epoch_acc += acc.item()

return epoch_loss / len(iterator), epoch_acc / len(iterator)

EPOCHS = 10

best_valid_loss = float('inf')
```

```
for epoch in trange(EPOCHS):

    start_time = time.monotonic()
      time = final_time -start_time

    train_loss, train_acc = train(model, train_iterator, optimizer, criterion,
device)
    valid_loss, valid_acc = evaluate(model, valid_iterator, criterion, device)

    if valid_loss < best_valid_loss:
        best_valid_loss = valid_loss
        torch.save(model.state_dict(), 'tut1-model.pt')

    end_time = time.monotonic()

    epoch_mins, epoch_secs = epoch_time(start_time, end_time)

print(f'happy: {epoch+1:02} \t {recall } \t {accuracy } \t {time(s)}')
```