Computer Vision Coursework Submission (IN3060/INM460)

Student name, ID and cohort: Rajani Mohan Janipalli (210049506) - PG

Notebook Setup

In this section you should include all the code cells required to test your coursework submission. Specifically:

Mount Google Drive

In [1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Define Local Path

In the next cell you should assign to the variable `GOOGLE_DRIVE_PATH_AFTER_MYDRIVE` the relative path of this folder in your Google Drive.

IMPORTANT: you have to make sure that **all the files required to test your functions are loaded using this variable** (as was the case for all lab tutorials). In other words, do not use in the notebook any absolute paths. This will ensure that the markers can run your functions. Also, **do not use** the magic command `%cd` to change directory.

In [2]:
```
import os

# TODO: Fill in the Google Drive path where you uploaded the CW_folder_PG
# Example: GOOGLE_DRIVE_PATH_AFTER_MYDRIVE = 'Colab Notebooks/Computer Vision/CW_folder_PG'

GOOGLE_DRIVE_PATH_AFTER_MYDRIVE = 'Colab Notebooks/Computer Vision Coursework/CW_Folder_PG'
GOOGLE_DRIVE_PATH = os.path.join('drive', 'My Drive', GOOGLE_DRIVE_PATH_AFTER_MYDRIVE)
print(os.listdir(GOOGLE_DRIVE_PATH))
```

```
['CNN test_function.ipynb', '.DS_Store', 'Models', 'CW_Dataset', 'Code', 'Video', 'test_functions.ipynb', 'SVM & MLP test_functio
n.ipynb']
```

Load packages

In the next cell you should load all the packages required to test your functions.

Note: Please run the below cell twice to avoid error display.

In [18]:
```
!pip install opencv-python==4.5.5.64
```

```
Requirement already satisfied: opencv-python==4.5.5.64 in /usr/local/lib/python3.7/dist-packages (4.5.5.64)
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packages (from opencv-python==4.5.5.64) (1.21.6)
```

In [19]:
```
import matplotlib.pyplot as plt
import numpy as np
from joblib import dump, load
import pickle

!pip show opencv-python
import cv2
from sklearn.model_selection import train_test_split
from skimage import img_as_ubyte, io, color
from sklearn.cluster import MiniBatchKMeans
from sklearn import svm, metrics
from collections import Counter

%matplotlib inline

import pandas as pd
from torch.utils.data import Dataset
import pandas as pd
from PIL import Image
import torch
import torch.nn as nn
from torch.utils.data import DataLoader
import torchvision.transforms as transforms
from tqdm import tqdm
import torchvision
import torchvision.transforms as transforms
import torch.nn.functional as F
import torch.optim as optim
from torchvision import models
from torch.optim import lr_scheduler
from google.colab.patches import cv2_imshow
import pickle
```

```
Name: opencv-python
Version: 4.5.5.64
Summary: Wrapper package for OpenCV python bindings.
```

```
Home-page: https://github.com/skvark/opencv-python
Author: None
Author-email: None
License: MIT
Location: /usr/local/lib/python3.7/dist-packages
Requires: numpy
Required-by: imgaug, dopamine-rl, albumentations
```

In [5]:

```
# Identify path to zipped dataset
CW_zip_path = os.path.join(GOOGLE_DRIVE_PATH, 'CW_Dataset', 'CW_Dataset.zip')

# Copy it to Colab
!cp '{CW_zip_path}' .

# Unzip it
!yes|unzip -q CW_Dataset.zip

# Delete zipped version from Colab (not from Drive)
!rm CW_Dataset.zip
```

In [6]:

```
device = ("cuda" if torch.cuda.is_available() else "cpu")
```

Load models

In the next cell you should load all your trained models for easier testing of your functions. Avoid to load them within `EmotionRecognition` and `EmotionRecognitionVideo` to avoid having to reload them each time.

In [7]:

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 22 * 22, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 8)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(x.size(0), 16 * 22 * 22)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

In [8]:

```
CNNMODELPATH = os.path.join(GOOGLE_DRIVE_PATH, 'Models/CNN_FER.pt')
net = Net()
net.load_state_dict(torch.load(CNNMODELPATH))
net.eval()
```

Out[8]:

```
Net(
  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=7744, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=84, bias=True)
  (fc3): Linear(in_features=84, out_features=8, bias=True)
)
```

In [9]:

```
models.vgg16(pretrained = True)
```

```
Downloading: "https://download.pytorch.org/models/vgg16-397923af.pth" to /root/.cache/torch/hub/checkpoints/vgg16-397923af.pth
```

Out[9]:

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
```

```
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

In [10]:

```
VGG16CNN1MODELPATH = os.path.join(GOOGLE_DRIVE_PATH, 'Models/VGG16CNN1_FER.pt')

device = torch.device('cpu')
vgg16cnn = models.vgg16(pretrained = True)
vgg16cnn.load_state_dict(torch.load(VGG16CNN1MODELPATH, map_location=device))
```

Out[10]:

```
<All keys matched successfully>
```

Test EmotionRecognition

This section should allow a quick test of the `EmotionRecognition` function. First, add cells with the code needed to load the necessary subroutines to make `EmotionRecognition` work.

In [11]:

```
path_to_testset = os.path.join(GOOGLE_DRIVE_PATH, 'CW_Dataset')
```

In [12]:

```
def EmotionRecognition(path_to_testset, model_name):

  def create_imagename_dataframe(path):
      """Create a dataframe of images and labels from selected directories"""
      data_df = pd.DataFrame(columns=["imgname", "label"])
      data_df["imgname"] = [file for file in sorted(os.listdir(os.path.join(path))) if file.endswith('.jpg')]
      label_set = np.loadtxt(os.path.join('labels', 'list_label_{}.txt'.format(path)), dtype='str')
      label_nums = [] # create an empty list to append label numbers.
      for i in range(len(label_set)): # execute a for loop to extract the exact labels from data and append them to a list.
        label_nums.append(label_set[i][1])

      data_df["label"] = label_nums

      data_df.to_csv (r'{}_df_csv.csv'.format(path), index = False, header=True)

      print('Create datafame with file name {}_df_csv.csv'.format(path))

  create_imagename_dataframe('test')

  def imagedisplay(img):
    img = img / 2 + 0.5     # Unnormalize: back to range [0, 1] just for showing the images
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))     # Reshape: C, H, W -> H, W, C
    plt.show()

  class CreateDataset(Dataset):
    def __init__(self, root_dir, annotation_file, transform=None):
        self.root_dir = root_dir
        self.annotations = pd.read_csv(annotation_file)
        self.transform = transform

    def __len__(self):
        return len(self.annotations)

    def __getitem__(self, idx):
        if torch.is_tensor(idx):
            idx = idx.tolist()

        img_id = self.annotations.iloc[idx, 0]
        img = Image.open(os.path.join(self.root_dir, img_id))
        y_label = self.annotations.iloc[idx, 1]

        if self.transform is not None:
            img = self.transform(img)

        return (img, y_label)

  transform = transforms.Compose(
```

```
        [
            transforms.ToTensor(),
            transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
        ]
    )
test_dataset = CreateDataset("test","test_df_csv.csv",transform=transform)
test_loader = DataLoader(dataset=test_dataset, shuffle=False, batch_size=16,num_workers=1)

if model_name == 'CNN':

  correct = 0
  total = 0
  with torch.no_grad():             # Avoid backprop at test
      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(f"Accuracy of the CNN on the 3068 test images: {100 * correct / total}%")
  imagedisplay(torchvision.utils.make_grid(images))
  print('Ground-truth:', labels)
  print('Ground-truth:', predicted)
  print('Label explanation: 1-Surprise, 2-Fear, 3-Disgust, 4-Happiness, 5-Sadness, 6-Anger, 7-Neutral')

elif model_name == 'VGG16CNN1':

  correct = 0
  total = 0
  with torch.no_grad():             # Avoid backprop at test
    for data in test_loader:
      vgg16cnn.cuda()
      images, labels = data
      outputs = vgg16cnn(images.cuda())
      _, predicted = torch.max(outputs.data, 1)
      total += labels.size(0)
      correct += (predicted == labels.cuda()).sum().item()

  print(f"Accuracy of the network on the 3068 test images: {100 * correct / total}%")
  imagedisplay(torchvision.utils.make_grid(images))
  print('Ground-truth:', labels)
  print('Ground-truth:', predicted)
  print('Label explanation: 1-Surprise, 2-Fear, 3-Disgust, 4-Happiness, 5-Sadness, 6-Anger, 7-Neutral')
```

Then, make a call to the `EmotionRecognition` function to see what results it produces. You must also indicate the syntax needed to test your different models.

In [13]:

```
EmotionRecognition(path_to_testset, 'CNN')
```

```
Create datafame with file name test_df_csv.csv
Accuracy of the CNN on the 3068 test images: 71.90352020860496%
```

```
Ground-truth: tensor([7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7])
Ground-truth: tensor([7, 5, 7, 7, 4, 7, 5, 7, 3, 3, 7, 5])
Label explanation: 1-Surprise, 2-Fear, 3-Disgust, 4-Happiness, 5-Sadness, 6-Anger, 7-Neutral
```

In [14]:

```
EmotionRecognition(path_to_testset, 'VGG16CNN1')
```

```
Create datafame with file name test_df_csv.csv
Accuracy of the network on the 3068 test images: 80.3129074315515%
```

```
Ground-truth: tensor([7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7])
Ground-truth: tensor([7, 7, 5, 7, 4, 7, 7, 1, 7, 7, 7, 7], device='cuda:0')
Label explanation: 1-Surprise, 2-Fear, 3-Disgust, 4-Happiness, 5-Sadness, 6-Anger, 7-Neutral
```

Test EmotionRecognitionVideo

This section should allow a quick test of the `EmotionRecognitionVideo` function. First, add cells with the code needed to load the necessary subroutines to make `EmotionRecognitionVideo` work.

In [15]:

```
video_input3 = os.path.join(GOOGLE_DRIVE_PATH, 'Video/EmotionalInterviewRDJr3.mp4')
```

In [16]:
```
def EmotionRecognitionVideo1(model, video_input):

  video = cv2.VideoCapture(video_input)

  transform = transforms.Compose(
            [
                transforms.ToTensor(),
                transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
            ]
        )

  # if (video.isOpened() == False):
  #     print("An error occured while trying to read video")
  # frame_width = int(video.get(3))
  # frame_height = int(video.get(4))

  while(video.isOpened()):
      ret, frame = video.read()
      if ret == True:
        model.eval()
        with torch.no_grad():

          haar_cas = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
          faces_detected = haar_cas.detectMultiScale(frame, scaleFactor = 1.2, minNeighbors=5)

          for (x, y, w, h) in faces_detected:
            cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0),2)

            converted_image = Image.fromarray(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
            trans_frame = transform(converted_image)
            trans_frame.cuda()
            model.cuda()

            outputs = model(trans_frame.unsqueeze(0).cuda())
            _, preds = torch.max(outputs.data, 1)

            labels = {1: "Surprise", 2: "Fear", 3: "Disgust", 4: "Happiness", 5: "Sadness", 6: "Anger", 7: "Neutral"}
            cv2.putText(frame, labels[preds.item()], (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255, 0, 0),2)
            cv2_imshow(frame)
        # out.write(frame)
        if cv2.waitKey(1) & 0xFF == ord("q"):
          break
      else:
        break

  video.release()
  cv2.destroyAllWindows()


  ## CODING REFERENCES:
  ## https://github.com/Conero007/Emotion-Detection-Pytorch/blob/master/video_capture.py
  ## https://github.com/vjgpt/Face-and-Emotion-Recognition/blob/master/face-rec-emotion.py
```

Then, make a call to the `EmotionRecognitionVideo` function to see what results it produces.

In [17]:
```
EmotionRecognitionVideo1(vgg16cnn, video_input3)
```

Neutral
ROBERT DOESN'T KNOW
WHAT A CAR IS

Neutral
ROBERT DOESN'T KNOW
WHAT A CAR IS

Neutral
ROBERT DOESN'T KNOW
WHAT A CAR IS

Neutral
ROBERT DOESN'T KNOW
WHAT A CAR IS

Neutral
ROBERT DOESN'T KNOW
WHAT A CAR IS

Neutral
ROBERT DOESN'T KNOW
WHAT A CAR IS

Neutral
ROBERT DOESN'T KNOW
WHAT A CAR IS

Neutral
ROBERT DOESN'T KNOW
WHAT A CAR IS

Neutral
ROBERT DOESN'T KNOW
WHAT A CAR IS

Neutral
ROBERT DOESN'T KNOW
WHAT A CAR IS