Computer Vision Coursework Submission (INM460)

Student name, ID and cohort: John Doe (200056789) - PG

Notebook Setup

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

Mount Google Drive

```
from google.colab import drive
drive.mount('/content/drive',force_remount=True)
```

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

Install Facenet

```
!pip install facenet_pytorch
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting facenet_pytorch
  Downloading facenet_pytorch-2.5.3-py3-none-any.whl (1.9 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.9/1.9 MB 23.0 MB/s eta 0:00:00
Requirement already satisfied: pillow in /usr/local/lib/python3.9/dist-packages (from facenet_pytorch) (8.4.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.9/dist-packages (from facenet_pytorch) (1.22.4)
Requirement already satisfied: torchvision in /usr/local/lib/python3.9/dist-packages (from facenet_pytorch) (0.15.1+cu11
Requirement already satisfied: requests in /usr/local/lib/python3.9/dist-packages (from facenet_pytorch) (2.27.1)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.9/dist-packages (from requests->facenet_pyto
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.9/dist-packages (from requests->facenet_p
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.9/dist-packages (from requests->facene
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/dist-packages (from requests->facenet_pytorch) (
Requirement already satisfied: torch==2.0.0 in /usr/local/lib/python3.9/dist-packages (from torchvision->facenet_pytorch)
Requirement already satisfied: networkx in /usr/local/lib/python3.9/dist-packages (from torch==2.0.0->torchvision->facene
Requirement already satisfied: triton==2.0.0 in /usr/local/lib/python3.9/dist-packages (from torch==2.0.0->torchvision->
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.9/dist-packages (from torch==2.0.0->torchvisio
Requirement already satisfied: sympy in /usr/local/lib/python3.9/dist-packages (from torch==2.0.0->torchvision->facenet_
Requirement already satisfied: jinja2 in /usr/local/lib/python3.9/dist-packages (from torch==2.0.0->torchvision->facenet_
Requirement already satisfied: filelock in /usr/local/lib/python3.9/dist-packages (from torch==2.0.0->torchvision->facene
Requirement already satisfied: lit in /usr/local/lib/python3.9/dist-packages (from triton==2.0.0->torch==2.0.0->torchvisi
Requirement already satisfied: cmake in /usr/local/lib/python3.9/dist-packages (from triton==2.0.0->torch==2.0.0->torchvi
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.9/dist-packages (from jinja2->torch==2.0.0->torc
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.9/dist-packages (from sympy->torch==2.0.0->torchvis
Installing collected packages: facenet_pytorch
Successfully installed facenet_pytorch-2.5.3
```

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.

```
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'

dirCVCourseWork = '/content/drive/MyDrive/ComputerVision/CVCourseWork_Mohsin/CW_Folder_PG/'
GOOGLE_DRIVE_PATH = os.path.join(dirCVCourseWork)
print(os.listdir(GOOGLE_DRIVE_PATH))
```

```
['CV2023_CW_Dataset', '.DS_Store', 'Video', 'Models', 'Code', 'test_functions.ipynb']
```

Load packages

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

```
import random
import cv2
import joblib
import matplotlib
import matplotlib.animation as animation
import matplotlib.pyplot as plt
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision.transforms as transforms
from facenet_pytorch import MTCNN
from joblib import dump, load
from matplotlib import patches, rc
from PIL import Image
from skimage import color, img_as_float, img_as_ubyte, io
from skimage.measure import label, regionprops
from sklearn import metrics
from torch.utils.data import DataLoader, Dataset, TensorDataset
```

Unzip Dataset

```
zip_path = os.path.join(GOOGLE_DRIVE_PATH, 'CV2023_CW_Dataset/CV2023_CW_Dataset.zip')

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

# Unzip it
!yes|unzip -q CV2023_CW_Dataset.zip

# Delete zipped version from Colab (not from Drive)
!rm CV2023_CW_Dataset.zip
```

Load models

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

```
class CNNFaceMaskModel(nn.Module):
  def __init__(self):
        super(CNNFaceMaskModel, 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*12*12, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 3)

  def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x =x.view(-1, 16*12*12)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x




CFMM = CNNFaceMaskModel()
```

```
def loadCNNModel():
  pathModelCFMM='Models/BestCFMM_MODEL.pth'
  CFMM.load_state_dict(torch.load(os.path.join(GOOGLE_DRIVE_PATH,pathModelCFMM)))
  return CFMM

def loadSVMModel():
  SVMBestModel = load(os.path.join(GOOGLE_DRIVE_PATH,'Models/bestModelSVM.joblib'))
  return SVMBestModel

def loadMLPModel():
  SVMBestModel = load(os.path.join(GOOGLE_DRIVE_PATH,'Models/bestSVMMaskModelFinal.joblib'))
  return SVMBestModel
```

Test MaskDetection

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

Function use Image from PIL to convert jpeg files to arrays in list of arrays and .and read text files using readline() and write the labels in list

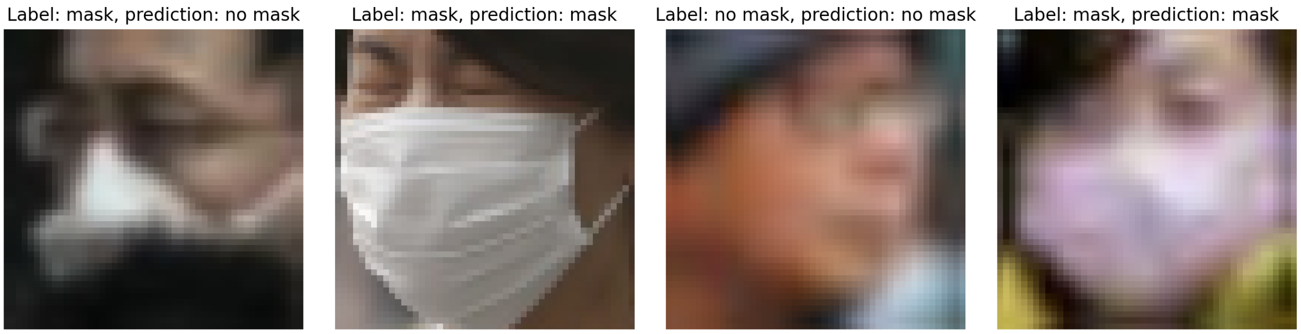
```
def dataAccesingAndConverting(path, label_list=None):
    """Load images and labels from selected directories"""
    images = []
    labels = []
    imageSize=[]

    if label_list is None:
        folder_names = [folder for folder in sorted(os.listdir(path)) if not folder.startswith('.')]
    else:
        folder_names = [folder for folder in sorted(os.listdir(path)) if folder in label_list]

    for folder in folder_names:
        if folder=='images':
          file_names = [file for file in sorted(os.listdir(os.path.join(path, folder))) if file.endswith('.jpeg')]
          for file in file_names:
            images.append(io.imread(os.path.join(path, folder, file)))
        if folder=='labels':
          label_file_names = [file for file in sorted(os.listdir(os.path.join(path, folder))) if file.endswith('.txt')]
          for file in label_file_names:
              with open(os.path.join(path, folder, file), 'r') as f:
                  label = f.readline().strip()
                  label= int(label)
                  labels.append(label)

    return images,labels
```

```
class faceMaskDataset(Dataset):
    def __init__(self, XFeatures, yLabel, transform=None):
        self.XFeatures = XFeatures
        self.yLabel = yLabel
        self.transform = transform

    def __len__(self):

        return len(self.XFeatures)

    def __getitem__(self, index):
        x =Image.fromarray(self.XFeatures[index])
        y = self.yLabel[index]
        if self.transform:
            x = self.transform(x)

        return x, y
    def numOfSamples(self):
        return len(self)
```

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

```
# Syntax for the next function is the following:
#
def MaskDetection(pathTotestset, modelType):
  if modelType == 'CNN':
    print()
    print('######### This Model is CNN ###########')
    print()
    modelClassifier = loadCNNModel()
  elif modelType == 'MLP':
    print()
    print('######### This Model is MLP ###########')
    print()
    modelClassifier=loadMLPModel()
  elif modelType == 'SVM':
    print()
    print('######### This Model is SVM ###########')
    print()
    modelClassifier=loadSVMModel()

  y_true = []
  y_pred = []
  randomImages=[]
  allImagesAndPrediction={}
  classLabelTestCFMM=['0','1','2']
  # Accessing XTest and YTest from the above by calling the function

  XTest,yTest= dataAccesingAndConverting(pathTotestset)

  # Transformation on the test set as also applied on the training set
  transform = transforms.Compose(
    [transforms.ToTensor(), transforms.Resize((60, 60),antialias=True),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
  testSetCFMM= faceMaskDataset(XTest,yTest, transform)


  testLoaderCFMM = torch.utils.data.DataLoader(testSetCFMM, batch_size=32, shuffle=True, num_workers=2)



  randomBatch = next(iter(testLoaderCFMM))
  imagesOfRandomBatch = randomBatch[0][:4]
  labelsOfRandomBatch=randomBatch[1][:4]
  newRandomDataset = TensorDataset(imagesOfRandomBatch,labelsOfRandomBatch)

  # create a DataLoader object with batch size 1 for the 4 images
  NewDataLoaderFromDatset = DataLoader(newRandomDataset, batch_size=1)
  fig, axes = plt.subplots(1, 4, figsize=(10, 20), sharex=True, sharey=True)
  ax = axes.ravel()
  for i, data in enumerate(NewDataLoaderFromDatset, 0):
      images, labels = data
      print(images.shape)
      randomImages.append(images)
      outputs = modelClassifier(images)
      _, predicted = torch.max(outputs, 1)
      y_true.extend(labels.tolist())
      y_pred.extend(predicted.tolist())
      print(y_pred)
      # Check if the image has already been added to the dictionary
      x_np = images.numpy()

      # Transpose to (H, W, C) format
      x_np = np.transpose(x_np,  (0, 2, 3, 1))

      #Clipped because the image values were between -1 and 1
      Image = np.clip(x_np, 0, 1)

      ax[i].imshow(Image[0])
      ax[i].set_title(f'Label: {y_true[i]} \n Prediction: {y_pred[i]}')
      ax[i].set_axis_off()


  fig.tight_layout()
  plt.show()
  # Print the classification report for all labels



pathToTestSet = os.path.join('test')
modelType = input("Enter a model type CNN, MLP, SVM: ")
MaskDetection(pathToTestSet, modelType)
```

```
Enter a model type CNN, MLP, SVM: CNN

######### This Model is CNN ###########
```



Label: mask, prediction: no mask

Label: mask, prediction: mask

Label: no mask, prediction: no mask

Label: mask, prediction: mask

Test MaskDetectionVideo

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

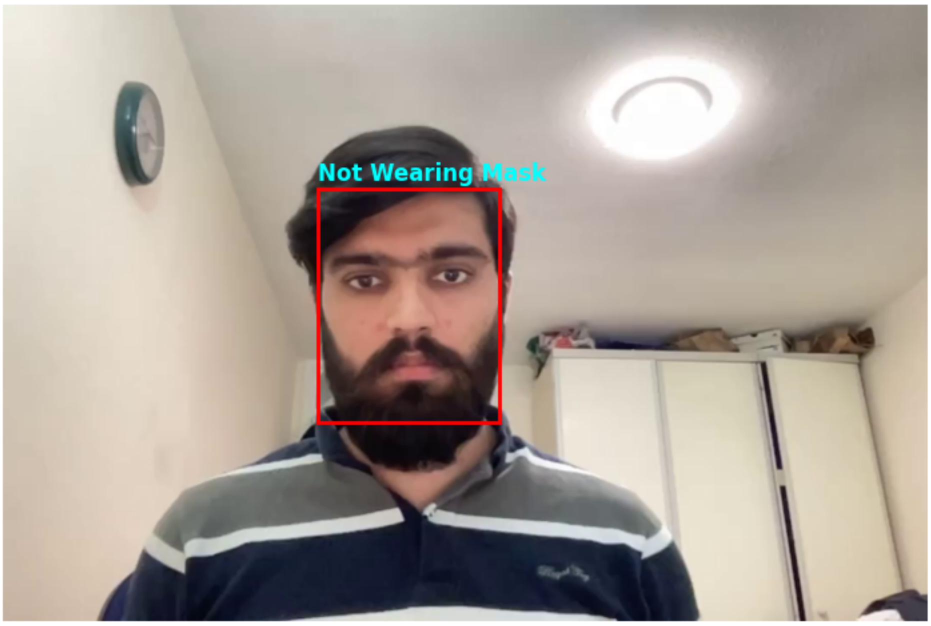
```
def imageTranformationVideoFrame(imageToTranform):
   tranformationMethodsPreprocessing = transforms.Compose([transforms.ToTensor(),
                                   transforms.Resize((60, 60),antialias=True),
                                   transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
                                   ])

   tranformedImage=tranformationMethodsPreprocessing(imageToTranform)

   return tranformedImage
```

This particular code was taken inspiration from lab 5 and lab 9 collectively

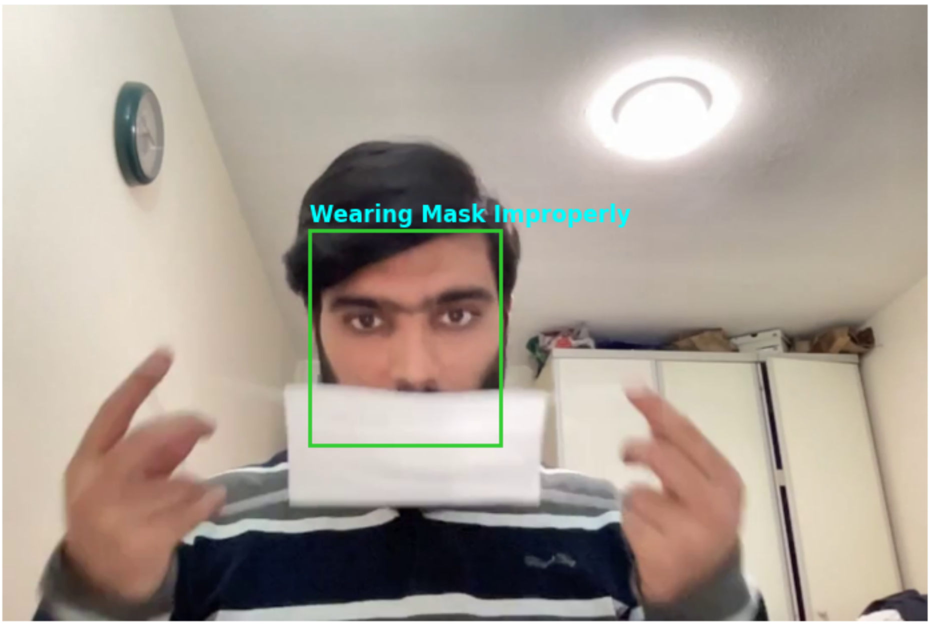
```

def MaskDetectionVideo(pathToVideo):
    %matplotlib inline

    cap = cv2.VideoCapture(os.path.join(GOOGLE_DRIVE_PATH, pathToVideo))
    frameCount = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    frameWidth = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    frameHeight = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

    video = np.empty((frameCount, frameHeight, frameWidth, 3), np.dtype('uint8'))

    fc = 0
    ret = True

    while fc < frameCount and ret:
        ret, video[fc] = cap.read()
        video[fc] = cv2.cvtColor(video[fc], cv2.COLOR_BGR2RGB)
        fc += 1

    cap.release()
    mtcnn = MTCNN(keep_all=True)

    # Iterate over every 20th frame
    for i in range(0, frameCount, 20):
        faces_MTCNN, _ = mtcnn.detect(video[i, :, :, :], landmarks=False)

        # Create a subplot for the frame
        fig, ax = plt.subplots(figsize=(9, 6))

        # Display the frame
        ax.imshow(video[i, :, :, :])
        ax.set_axis_off()
        ax.set_title('Frame %d' % (i+1))

        if faces_MTCNN is not None:
          for face in faces_MTCNN:
              # Get the face coordinates and convert to tensor
              x1, y1, x2, y2 = face.astype(int)
              face_region = (video[i, y1:y2, x1:x2])

              # # Convert the face region to a PIL Image object
              face_pil = Image.fromarray(face_region)

              faceTransformed = imageTranformationVideoFrame(face_pil)



              with torch.no_grad():
                 output = modelClassifierCNN(faceTransformed)
                 _, predicted = torch.max(output, 1)


                 y_pred.extend(predicted.tolist())


              # Draw a rectangle around the face and label it as "with mask" or "without mask"
                 if predicted[0].item() == 1:
                    labelAssignedMask = "Wearing a Mask"
                    colorMask='y'
                 elif predicted[0].item() == 2:
                    labelAssignedMask ="Wearing Mask Improperly"
                    colorMask='limegreen'
                 else:
                    labelAssignedMask ="Not Wearing Mask"
                    colorMask='r'

                  # color = 'p' if prob > mask_threshold else 'r'

                 ax.add_patch(patches.Rectangle(xy=(x1, y1), width=x2-x1, height=y2-y1,
                                                        fill=False, color=colorMask, linewidth=2))

                 ax.text(x1, y1-10, labelAssignedMask , fontsize=12, color='cyan', fontweight='bold')
        plt.show()

    return video
```

```
videoMaskWild=MaskDetectionVideo('Video/MaskVideo.mp4')
```

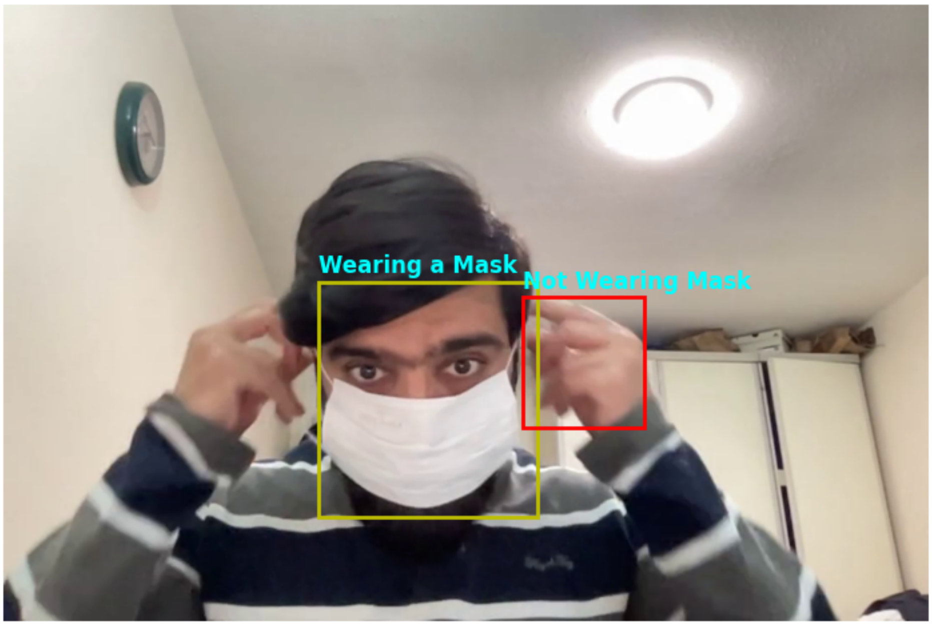
Frame 1



Frame 21



Frame 41



Frame 61