Double-click (or enter) to edit

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')

     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.

```
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/CW_Folder_PG/CW_Folder_PG'
GOOGLE_DRIVE_PATH = os.path.join('drive', 'My Drive', GOOGLE_DRIVE_PATH_AFTER_MYDRIVE)
print(os.listdir(GOOGLE_DRIVE_PATH))
```

```
['.DS_Store', 'Models', 'Code', 'CW_Dataset', 'Video', 'Copy of Copy of Copy of test_functions.ipynb', 'test_functions.ipynb']
```

Load packages

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

```
!pip install opencv-python==4.5.5.64
```

```
Collecting opencv-python==4.5.5.64
  Downloading opencv_python-4.5.5.64-cp36-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (60.5 MB)
     |████████████████████████████████| 60.5 MB 81 kB/s
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)
Installing collected packages: opencv-python
  Attempting uninstall: opencv-python
    Found existing installation: opencv-python 4.1.2.30
    Uninstalling opencv-python-4.1.2.30:
      Successfully uninstalled opencv-python-4.1.2.30
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is th
albumentations 0.1.12 requires imgaug<0.2.7,>=0.2.5, but you have imgaug 0.2.9 which is incompatible.
Successfully installed opencv-python-4.5.5.64
```

```
import matplotlib.pyplot as plt
import numpy as np
from joblib import dump, load
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
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import classification_report
from sklearn.metrics import plot_confusion_matrix
import pickle
from skimage.feature import hog
from skimage import data, exposure
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import make_classification
```

```
%matplotlib inline
```

```
import torch                                        # root package
from torch.utils.data import Dataset, DataLoader, SubsetRandomSampler    # dataset representation and loading
import torchvision
from torchvision import models              # construct a model with random weights by calling its constructor
import torch.optim as optim
from torch.optim import lr_scheduler
import torchvision.transforms as transforms # composable transforms
import torch.autograd as autograd           # computation graph
from torch import Tensor                    # tensor node in the computation graph
import torch.nn as nn                       # neural networks
import torch.nn.functional as F             # layers, activations and more
import torch.optim as optim                 # optimizers e.g. gradient descent, ADAM, etc.
from torch.jit import script, trace         # hybrid frontend decorator and tracing jit
```

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision
from torchvision import models
import torchvision.transforms as transforms
from torch.utils.data import Dataset, DataLoader, SubsetRandomSampler
import os
from PIL import Image
import torch.optim as optim
```

```
from torch.optim import lr_scheduler
import cv2
from google.colab.patches import cv2_imshow
import pickle
import time
import random
```

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.

SIFT+SVM

```
with open ("/content/drive/My Drive/Colab Notebooks/Computer Vision/CW_Folder_PG/CW_Folder_PG/Models/Model_SIFT_SVM.pickle", 'rb') as
    model_name = pickle.load(f)
```

HOG+MLP

```
with open ("/content/drive/My Drive/Colab Notebooks/Computer Vision/CW_Folder_PG/CW_Folder_PG/Models/Model_HOG_MLP.pickle", 'rb') as
    model_name = pickle.load(f)
```

CNN - VGG16

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

```
Downloading: "https://download.pytorch.org/models/vgg16-397923af.pth" to /root/.cache/torch/hub/checkpo
100%                                              528M/528M [00:06<00:00, 90.8MB/s]

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)
```

```
with open("/content/drive/My Drive/Colab Notebooks/Computer Vision/CW_Folder_PG/CW_Folder_PG/Models/VGG_model.pickle", "rb") as f:
    VGG16_model = pickle.load(f)
```

```
      )
```

TEST FUNCTION FOR SVM AND MLP

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

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

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

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

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.

```
# Below is the test function for SIFT+SVM, HOG+MLP, HOG+SVM

def EmotionRecognition(path_to_testset, model):
  with open (path_to_testset, 'rb') as f:
    model_name = pickle.load(f)
  #print("1")
  def import_dataset(content_path, path):
    """Load images and labels from selected directories"""
    images = []
```

```
      labels = []

      label_file = open(content_path+"labels/list_label_" + path + ".txt", "r") # reading the text files from the path
      labels_num = [name.split(' ')[1][0] for name in label_file] # Slicing off the class number to create a list of all the classes

      for num in range(len(labels_num)):
        if labels_num[num] == '1':
          labels.append('Suprise')
        elif labels_num[num] == '2':
          labels.append('Fear')
        elif labels_num[num] == '3':
          labels.append('Disgust')
        elif labels_num[num] == '4':
          labels.append('Happiness')
        elif labels_num[num] == '5':
          labels.append('Sadness')
        elif labels_num[num] == '6':
          labels.append('Anger')
        else:
          labels.append('Neutral')
  # for images
      img_file = [file for file in sorted(os.listdir(os.path.join(path))) if file.endswith('.jpg')] # locate the .jpg files in the path
      for file in img_file:
          images.append(io.imread(os.path.join(path, file))) # append the images in the list to create a list of arrays

      return images, labels
      #print("2")
    X_train, y_train = import_dataset('/content/','train')
    X_test, y_test = import_dataset('/content/','test')
    #print(y_test[0])

    if model == "SIFT+SVM":
      #print("####")
      sift = cv2.SIFT_create()
      des_list = []
      y_train_list = []
```

```
        for i in range(len(X_train)):
            # Identify keypoints and extract descriptors with SIFT
            img = img_as_ubyte(color.rgb2gray(X_train[i]))
            kp, des = sift.detectAndCompute(img, None)
            # Append list of descriptors and label to respective lists
            if des is not None:
                des_list.append(des)
                y_train_list.append(y_train[i])
        des_array = np.vstack(des_list)
        hist_list = []
        k = len(np.unique(y_train)) * 10
        batch_size = des_array.shape[0] // 4
        kmeans = MiniBatchKMeans(n_clusters=k, batch_size=batch_size).fit(des_array)
        for i in range(len(X_test)):
            img = img_as_ubyte(color.rgb2gray(X_test[i]))
            kp, des = sift.detectAndCompute(img, None)

            if des is not None:
                hist = np.zeros(k)

                idx = kmeans.predict(des)

                for j in idx:
                    hist[j] = hist[j] + (1 / len(des))

                # hist = scale.transform(hist.reshape(1, -1))
                hist_list.append(hist)

            else:
                hist_list.append(None)

        # Remove potential cases of images with no descriptors
        idx_not_empty = [i for i, x in enumerate(hist_list) if x is not None]
        hist_list = [hist_list[i] for i in idx_not_empty]
        y_test_sift = [y_test[i] for i in idx_not_empty]
        hist_array = np.vstack(hist_list)
        y_Pred_SS = model_name.predict(hist_array).tolist()
```

```
    print(classification_report(y_test_sift, y_Pred_SS))
    fig, axes = plt.subplots(1, 4, figsize=(14, 7), sharex=True, sharey=True)
    ax = axes.ravel()

    for i in range(4):
        r = random.randint(0,len(X_test))
        ax[i].imshow(X_test[r])
        ax[i].set_title(f'Label: {y_test[r]} \n Prediction: {y_Pred_SS[r]}')
        ax[i].set_axis_off()
    fig.tight_layout()
    plt.show()
  if model == "HOG+SVM" or model == "HOG+MLP":
    hog_images = []
    hog_features = []
    ppc = 16

    for image in range(len(X_test)):

      img = img_as_ubyte(color.rgb2gray(X_test[image]))
      fd, hog_image = hog( img, orientations=8, pixels_per_cell=(ppc, ppc),  cells_per_block=(1, 1), visualize=True)

      hog_images.append(hog_image)
      hog_features.append(fd)
    y_Pred = model_name.predict(hog_features).tolist()
    print(classification_report(y_test, y_Pred))
    fig, axes = plt.subplots(1, 4, figsize=(14, 7), sharex=True, sharey=True)
    ax = axes.ravel()

    for i in range(4):
        r = random.randint(0,len(X_test))
        ax[i].imshow(X_test[r])
        ax[i].set_title(f'Label: {y_test[r]} \n Prediction: {y_Pred[r]}')
        ax[i].set_axis_off()
    fig.tight_layout()
    plt.show()

# SIFT+SVM = os.path.join(GOOGLE_DRIVE_PATH, 'Models','Model_SIFT_SVM.pickle')
# HOG+MLP = os.path.join(GOOGLE_DRIVE_PATH, 'Models','Model_HOG_MLP.pickle')
# HOG+SVM = os.path.join(GOOGLE_DRIVE_PATH, 'Models','Model_HOG_SVM.pickle')
```

```
#hog-svm = load(os.path.join(GOOGLE_DRIVE_PATH, 'Models','hog-svm.joblib'))
```

TEST FUNCTION FOR CONVOLUTIONAL NEURAL NETWORK

PLEASE CHANGE THE RUNTIME TO GPU TO EXECUTE THE FUNCTIONS RELATED TO CNN***

It was taking a lot of time to create the model in CPU. So, GPU was used to create and save the model. So, in this test function the model doesn't load if the runtime is not GPU.

VGG16 MODEL

```
import os
import pandas as pd
from PIL import Image
import torch                                        # root package
from torch.utils.data import Dataset, DataLoader, SubsetRandomSampler    # dataset representation and loading
import torchvision
from torchvision import models              # construct a model with random weights by calling its constructor
import torch.optim as optim
from torch.optim import lr_scheduler
import torchvision.transforms as transforms # composable transforms
import torch.autograd as autograd           # computation graph
from torch import Tensor                    # tensor node in the computation graph
import torch.nn as nn                       # neural networks
import torch.nn.functional as F             # layers, activations and more
import torch.optim as optim                 # optimizers e.g. gradient descent, ADAM, etc.
from torch.jit import script, trace         # hybrid frontend decorator and tracing jit
```

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

```
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)
  )
)
```

```
with open("/content/drive/My Drive/Colab Notebooks/Computer Vision/CW_Folder_PG/CW_Folder_PG/Models/VGG_model.pickle", "rb") as f:
    VGG16_model = pickle.load(f)
```

```
def EmotionRecognition_cnn(path_to_testset, model_name):

  if model_name == 'VGG16':

    def loading_data(path):
      label_path = path
      column_names = ["Image_filename", "Labels"]
      label_df = pd.read_csv(label_path, names = column_names, delim_whitespace = True)
      label_df["Image_filename"] = label_df["Image_filename"].apply(lambda i: i[:-4])
      label_df["Image_filename"] = label_df["Image_filename"] + "_aligned.jpg"
      # print(label_df.head())
      return label_df
    train_df = loading_data("/content/labels/list_label_train.txt")
    test_df = loading_data("/content/labels/list_label_test.txt")

    class CustomImageDataset(Dataset):
        def __init__(self, annotations_file, img_dir, transform=None, target_transform=None):
            self.img_labels = annotations_file
            self.img_dir = img_dir
            self.transform = transform
            #self.target_transform = target_transform # commenting out as target doesn't require any transformation

        def __len__(self):
            return len(self.img_labels)
        def __getitem__(self, idx):
            if torch.is_tensor(idx): # https://discuss.pytorch.org/t/custom-dataset-weird-probelm/59278
              idx = idx.tolist()

            img_path = os.path.join(self.img_dir, self.img_labels.iloc[idx, 0])
            image = Image.open(img_path) # opens and identifies the given image. Reference - https://www.geeksforgeeks.org/python-pil
            label = self.img_labels.iloc[idx, 1]
            if self.transform:
                image = self.transform(image)
            # if self.target_transform:
            #     label = self.target_transform(label)
            return image, label

    transform = transforms.Compose(
        [transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
    test_image_path = "/content/test"
    test_data = CustomImageDataset(annotations_file = test_df,img_dir = test_image_path, transform = transform)
    testloader = torch.utils.data.DataLoader(test_data, batch_size=16,shuffle=False, num_workers=2)

    correct = 0
    total = 0
    with torch.no_grad():
      for i, (images, labels) in enumerate(testloader):

          outputs = VGG16_model(images.cuda())
          _, predicted = torch.max(outputs.data, 1)
          total += len(labels)
          correct += (predicted == labels.cuda()).sum()
    vgg_accuracy = 100 * correct / float(total)
    print("Accuracy of the test images: {} % ".format(vgg_accuracy))

    def imshow(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()
```

```
    dataiter = iter(testloader)
    images1, labels1 = dataiter.next()

    # show images and print labels
    imshow(torchvision.utils.make_grid(images1))
    # first_labels = [labels1[j] for j in range(4)]
    # print('Ground-truth:', first_labels)
    first_labels = [label for label in predicted]
    print('Predicted Labels:', first_labels)
    # print(f"Accuracy of the test images: {100 * correct / total}%")

   # emotions = {0: "Surprise", 1: "Fear", 2: "Disgust", 3: "Happiness", 4: "Sadness", 5: "Anger", 6: "Neutral"}
```

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

Due to limitations in Pytorch, it wasn't possible to convert the labels to string.

```
pass
```

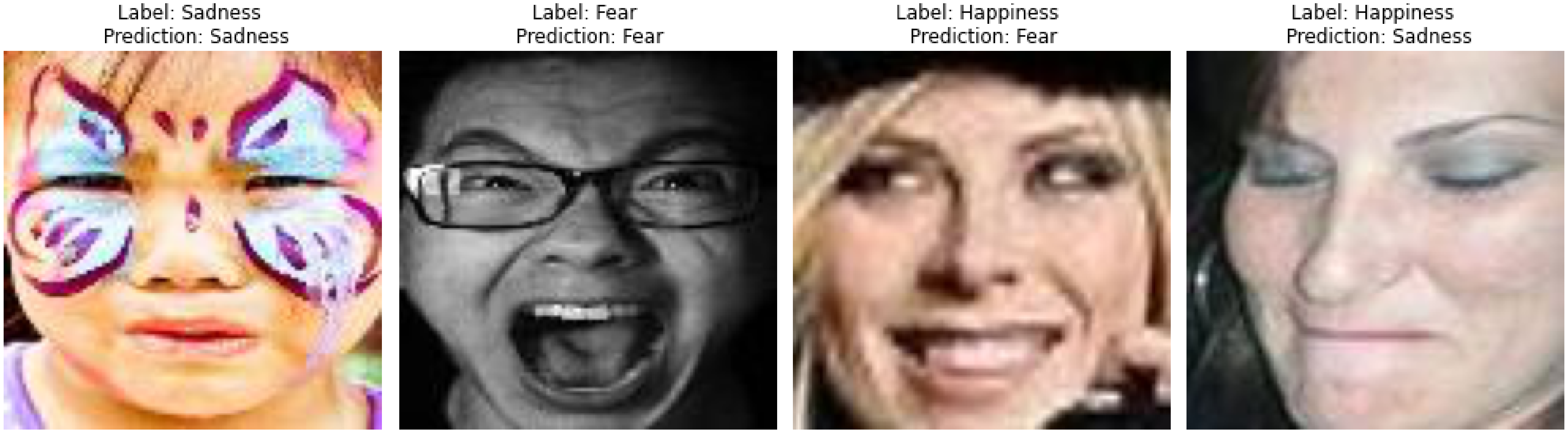
Then, make a call to the `EmotionRecognition` function to see what results it produces. You must also indicate the syntax needed to test your different models.

```
EmotionRecognition_cnn(path_to_testset, 'VGG16')
```

```
Accuracy of the test images: 81.61668395996094 %
```

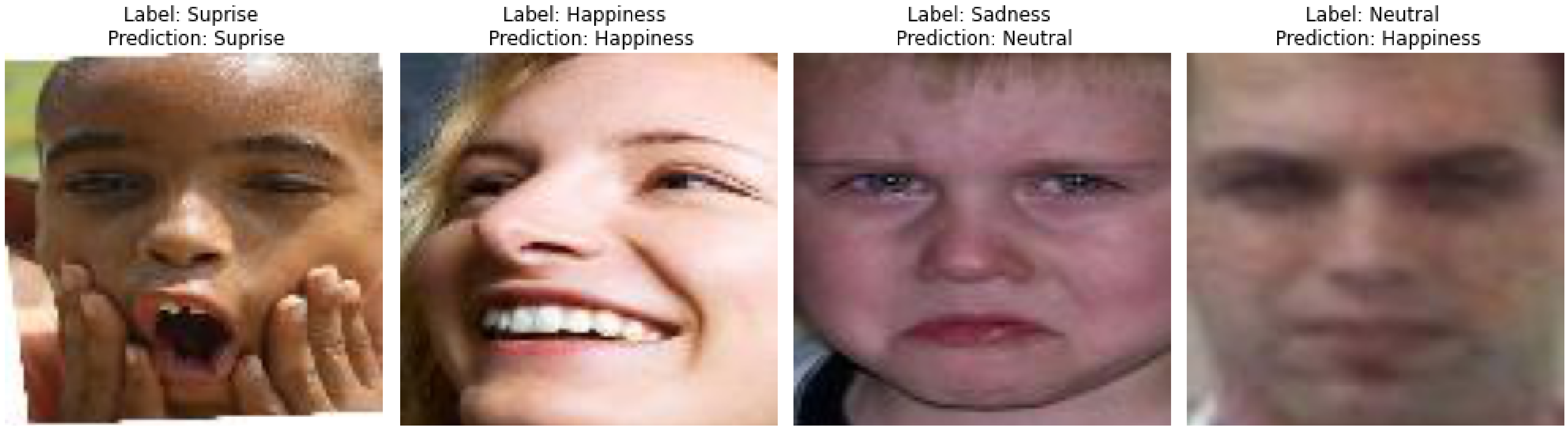
```
EmotionRecognition(os.path.join(GOOGLE_DRIVE_PATH, 'Models','Model_HOG_SVM.pickle'), "HOG+SVM")
```

```
              precision    recall  f1-score   support

       Anger       0.38      0.55      0.45       162
     Disgust       0.21      0.29      0.24       160
        Fear       0.37      0.34      0.35        74
   Happiness       0.84      0.73      0.78      1185
     Neutral       0.61      0.59      0.60       680
     Sadness       0.51      0.51      0.51       478
     Suprise       0.52      0.61      0.56       329

    accuracy                           0.61      3068
   macro avg       0.49      0.52      0.50      3068
weighted avg       0.64      0.61      0.62      3068
```



```
EmotionRecognition(os.path.join(GOOGLE_DRIVE_PATH, 'Models','Model_HOG_MLP.pickle'), "HOG+MLP")
```

```
              precision    recall  f1-score   support

       Anger       0.52      0.37      0.43       162
     Disgust       0.23      0.07      0.11       160
        Fear       0.56      0.27      0.36        74
   Happiness       0.75      0.83      0.79      1185
     Neutral       0.53      0.66      0.59       680
     Sadness       0.46      0.53      0.49       478
     Suprise       0.70      0.34      0.46       329

    accuracy                           0.62      3068
   macro avg       0.54      0.44      0.46      3068
weighted avg       0.61      0.62      0.60      3068
```



Label: Suprise
Prediction: Suprise

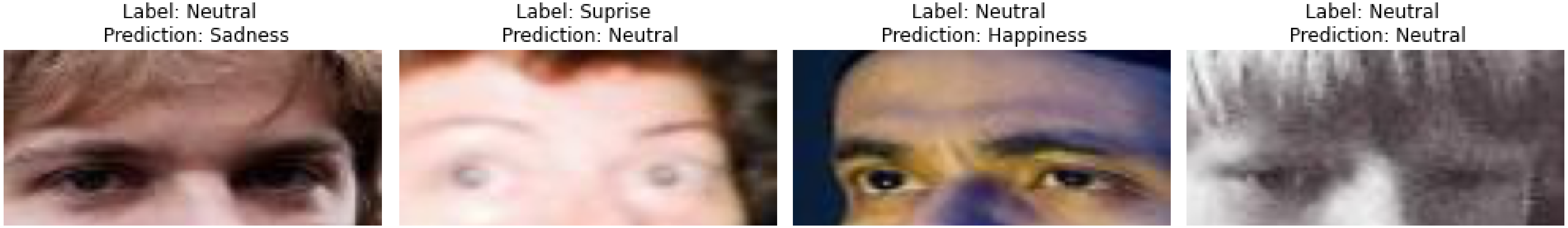
Label: Happiness
Prediction: Happiness

Label: Sadness
Prediction: Neutral

Label: Neutral
Prediction: Happiness

```
EmotionRecognition(os.path.join(GOOGLE_DRIVE_PATH, 'Models','Model_SIFT_SVM.pickle'), "SIFT+SVM")
```

```
Sadness
              precision    recall  f1-score   support

       Anger       0.10      0.19      0.13       162
     Disgust       0.06      0.14      0.08       160
        Fear       0.06      0.07      0.06        74
   Happiness       0.41      0.26      0.32      1184
     Neutral       0.28      0.37      0.32       679
     Sadness       0.15      0.10      0.12       478
     Suprise       0.11      0.12      0.11       329

    accuracy                           0.23      3066
   macro avg       0.17      0.18      0.16      3066
weighted avg       0.26      0.23      0.23      3066
```



```
# Syntax for the next function is the following:
#
# EmotionRecognition(path_to_testset, model_type)
#
# where model_type can be one of
#   - hog-svm
#   - hog-mlp
#   - cnn

#path_to_testset = os.path.join(GOOGLE_DRIVE_PATH, 'CW_Dataset/test')
#EmotionRecognition(path_to_testset, 'hog-svm')
```

Test EmotionRecognitionVideo

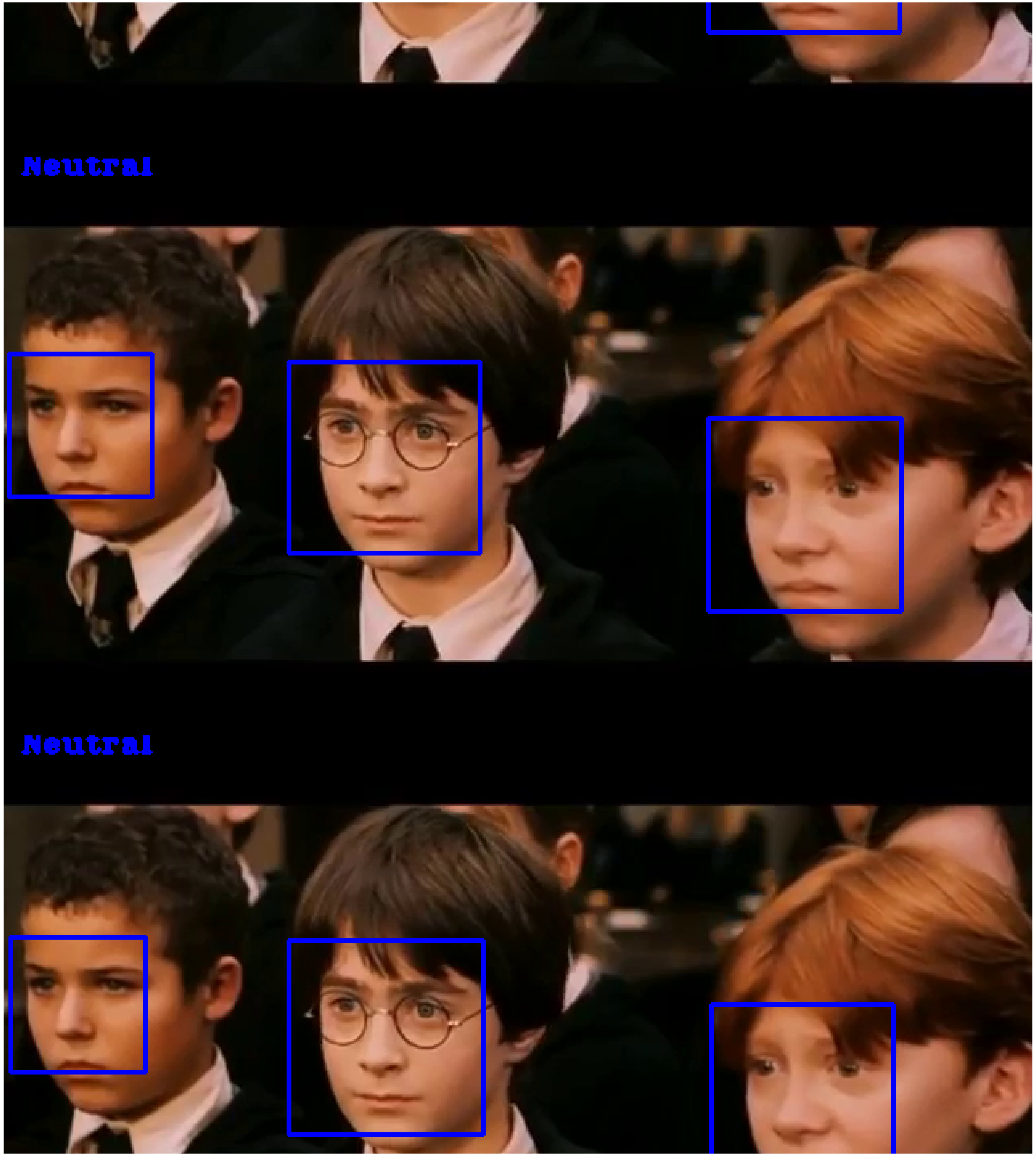
This section should allow a quick test of the `EmotionRecognitionVideo` function. First, add cells with the code needed to load the necessary subroutines to make `EmotionRecognitionVideo` work.

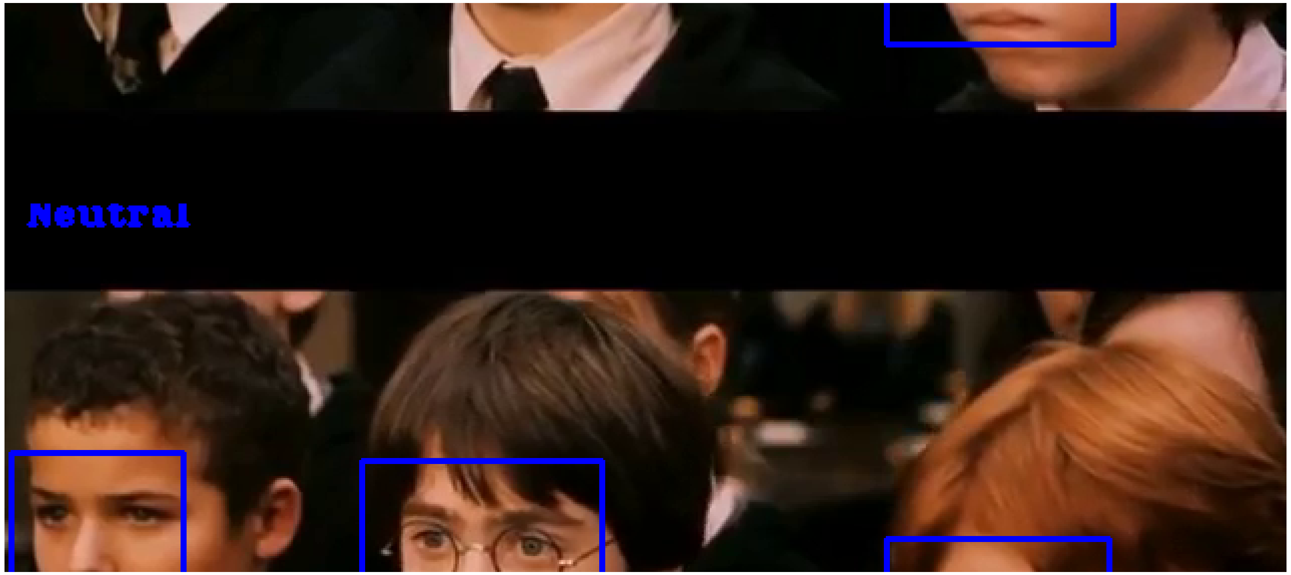
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision
from torchvision import models
import torchvision.transforms as transforms
from torch.utils.data import Dataset, DataLoader, SubsetRandomSampler
import os
from PIL import Image
import torch.optim as optim
from torch.optim import lr_scheduler
import cv2
from google.colab.patches import cv2_imshow
import pickle
import time
```

```
# This code was prepared using several references. Hence, the references have been provided step by step
def EmotionRecognitionVideo(model, video_path):

  cap = cv2.VideoCapture(video_path)

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

  if not cap.isOpened(): # https://docs.opencv.org/4.x/dd/d43/tutorial_py_video_display.html
    print("Cannot open the video")
    exit()

  while True:
    ret, frame = cap.read()
    if ret == True:
      #start_time = time.time()
      model.eval() # https://stackoverflow.com/questions/68846681/how-do-i-stream-a-video-with-opencv-into-my-pytorch-neural-network
      with torch.no_grad():

        face_cas = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml') # https://pythonprogramming.n
        faces_detected = face_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))
          tensor_frame = transform(converted_image) # converting images to tensor # https://stackoverflow.com/questions/68824648/why-
          tensor_frame.cuda()
          model.cuda()

        outputs = model(tensor_frame.unsqueeze(0).cuda()) # putting the images into batch form for the model to accept
        _, preds = torch.max(outputs.data, 1)

      facial_emotions = {1: "Surprise", 2: "Fear", 3: "Disgust", 4: "Happiness", 5: "Sadness", 6: "Anger", 7: "Neutral"}
      cv2.putText(frame, facial_emotions[preds.item()], (10, 10), cv2.FONT_HERSHEY_COMPLEX_SMALL, 0.9, (255, 0, 0),2)

      cv2_imshow(frame)
      if cv2.waitKey(1) & 0xFF == ord("q"): # https://stackoverflow.com/questions/29317262/opencv-video-saving-in-python
        break
    else:
      break

  cap.release()
  cv2.destroyAllWindows()

with open("/content/drive/My Drive/Colab Notebooks/Computer Vision/CW_Folder_PG/CW_Folder_PG/Models/VGG_model.pickle", "rb") as f:
    VGG16_model = pickle.load(f)


Harry_Potter = "/content/drive/My Drive/Colab Notebooks/Computer Vision/CW_Folder_PG/CW_Folder_PG/Video/Facial expression - Ron.mp4"
```

*** Note: Sometimes the output of the below function saves in colab

```
EmotionRecognitionVideo(VGG16_model, Harry_Potter)
```


Neutral
Neutral


Neutral