In [ ]:
```
#Load Packages
import numpy as np
import pandas as pd
import os
import matplotlib
import seaborn as sns
import matplotlib.pyplot as plt
from tqdm import tqdm_notebook
%matplotlib inline
import cv2 as cv
```

In [2]:
```
#Load Data
DATA_FOLDER = 'data'
TRAIN_SAMPLE_FOLDER = 'train_sample_videos'
TEST_FOLDER = 'test_videos'

print(f"Train samples: {len(os.listdir(os.path.join(DATA_FOLDER, TRAIN_SAMPLE_FOLDER)))}")
print(f"Test samples: {len(os.listdir(os.path.join(DATA_FOLDER, TEST_FOLDER)))}")
```

```
Train samples: 2237
Test samples: 1700
```

In [ ]:
```
FACE_DETECTION_FOLDER = 'haar-cascades-for-face-detection'
print(f"Face detection resources: {os.listdir(FACE_DETECTION_FOLDER)}")
```

In [ ]:
```
#Check file type.Here we check the train data files extensions. Most of the files looks to have `mp4` extension, let's check if there is other extension as well.
train_list = list(os.listdir(os.path.join(DATA_FOLDER, TRAIN_SAMPLE_FOLDER)))
ext_dict = []
for file in train_list:
    file_ext = file.split('.')[1]
    if (file_ext not in ext_dict):
        ext_dict.append(file_ext)
print(f"Extensions: {ext_dict}")
```

In [ ]:
```
#Let's count how many files with each extensions there are.
for file_ext in ext_dict:
    print(f"Files with extension `{file_ext}`: {len([file for file in train_list if  file.endswith(file_ext)])}")
```

In [ ]:
```
#Let's repeat the same process for test videos folder.
test_list = list(os.listdir(os.path.join(DATA_FOLDER, TEST_FOLDER)))
ext_dict = []
for file in test_list:
    file_ext = file.split('.')[1]
    if (file_ext not in ext_dict):
        ext_dict.append(file_ext)
print(f"Extensions: {ext_dict}")
for file_ext in ext_dict:
    print(f"Files with extension `{file_ext}`: {len([file for file in train_list if  file.endswith(file_ext)])}")
```

In [ ]:
```
#Let's check the `json` file first.
json_file = [file for file in train_list if  file.endswith('json')][0]
print(f"JSON file: {json_file}")
```

In [ ]:
```
#Aparently here is a metadata file. Let's explore this JSON file.
def get_meta_from_json(path):
    df = pd.read_json(os.path.join(DATA_FOLDER, path, json_file))
    df = df.T
    return df

meta_train_df = get_meta_from_json(TRAIN_SAMPLE_FOLDER)
meta_train_df.head()
```

In [ ]:
```
#Check for missing data
def missing_data(data):
    total = data.isnull().sum()
    percent = (data.isnull().sum()/data.isnull().count()*100)
    tt = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
    types = []
    for col in data.columns:
        dtype = str(data[col].dtype)
        types.append(dtype)
    tt['Types'] = types
    return(np.transpose(tt))
```

In [ ]:
```
missing_data(meta_train_df)
```

In [ ]:
```
##There are missing data 6.35% of the samples (or 108).
#We suspect that actually the real data has missing original (if we generalize from the data we glimpsed). Let's check this hypothesis.
```

In [ ]:
```
missing_data(meta_train_df.loc[meta_train_df.label=='REAL'])
```

In [ ]:
```
#Indeed, all missing original data are the one associated with REAL label.
```

In [ ]:
```
#Unique values
#Let's check into more details the unique values.
```

In [ ]:
```
def unique_values(data):
    total = data.count()
    tt = pd.DataFrame(total)
    tt.columns = ['Total']
    uniques = []
    for col in data.columns:
        unique = data[col].nunique()
        uniques.append(unique)
    tt['Uniques'] = uniques
    return(np.transpose(tt))
```

In [ ]:
```
unique_values(meta_train_df)
```

In [ ]:
```
#We observe that original label has the same pattern for uniques values.
#We know that we have 108 missing data (that's why total is only 1591) and we observe that we do have 108 unique examples.
```

In [ ]:
```
#Most frequent originals
#Let's look now to the most frequent originals uniques in train sample data.
```

In [ ]:
```
def most_frequent_values(data):
    total = data.count()
    tt = pd.DataFrame(total)
    tt.columns = ['Total']
    items = []
    vals = []
    for col in data.columns:
        itm = data[col].value_counts().index[0]
        val = data[col].value_counts().values[0]
        items.append(itm)
        vals.append(val)
    tt['Most frequent item'] = items
    tt['Frequence'] = vals
    tt['Percent from total'] = np.round(vals / total * 100, 3)
    return(np.transpose(tt))
```

In [ ]:
```
most_frequent_values(meta_train_df)
```

In [ ]:
```
#We see that most frequent label is FAKE (93.64%), fneqiqpqvs.mp4 is the most frequent original (35 samples).
```

In [ ]:
```
#Video data exploration
#In the following we will explore some of the video data.

#Missing video (or meta) data
#We check first if the list of files in the meta info and the list from the folder are the same.
```

In [ ]:
```
meta = np.array(list(meta_train_df.index))
storage = np.array([file for file in train_list if  file.endswith('mp4')])
print(f"Metadata: {meta.shape[0]}, Folder: {storage.shape[0]}")
print(f"Files in metadata and not in folder: {np.setdiff1d(meta,storage,assume_unique=False).shape[0]}")
print(f"Files in folder and not in metadata: {np.setdiff1d(storage,meta,assume_unique=False).shape[0]}")
```

In [ ]:
```
fake_train_sample_video = list(meta_train_df.loc[meta_train_df.label=='FAKE'].sample(3).index)
fake_train_sample_video
```

In [ ]:
```
def display_image_from_video(video_path):
    '''
    input: video_path - path for video
    process:
    1. perform a video capture from the video
    2. read the image
    3. display the image
    '''
    capture_image = cv.VideoCapture(video_path)
    ret, frame = capture_image.read()
    fig = plt.figure(figsize=(10,10))
    ax = fig.add_subplot(111)
    frame = cv.cvtColor(frame, cv.COLOR_BGR2RGB)
    ax.imshow(frame)
```

In [ ]:
```
for video_file in fake_train_sample_video:
    display_image_from_video(os.path.join(DATA_FOLDER, TRAIN_SAMPLE_FOLDER, video_file))
```

In [ ]:
```
#Few Real videos
real_train_sample_video = list(meta_train_df.loc[meta_train_df.label=='REAL'].sample(9).index)
real_train_sample_video
```

In [ ]:
```
for video_file in real_train_sample_video:
    display_image_from_video(os.path.join(DATA_FOLDER, TRAIN_SAMPLE_FOLDER, video_file))
```

In [ ]:
```
#Videos with same original
#Let's look now to set of samples with the same original.
```

In [ ]:
```
meta_train_df['original'].value_counts()[0:9]
```

In [ ]:
```
def display_image_from_video_list(video_path_list, video_folder=TRAIN_SAMPLE_FOLDER):
    '''
    input: video_path_list - path for video
    process:
    0. for each video in the video path list
        1. perform a video capture from the video
        2. read the image
        3. display the image
    '''
    plt.figure()
    fig, ax = plt.subplots(2,3,figsize=(16,8))
    # we only show images extracted from the first 6 videos
    for i, video_file in enumerate(video_path_list[0:6]):
        video_path = os.path.join(DATA_FOLDER, video_folder,video_file)
        capture_image = cv.VideoCapture(video_path)
        ret, frame = capture_image.read()
        frame = cv.cvtColor(frame, cv.COLOR_BGR2RGB)
        ax[i//3, i%3].imshow(frame)
        ax[i//3, i%3].set_title(f"Video: {video_file}")
        ax[i//3, i%3].axis('on')
```

In [ ]:
```
same_original_fake_train_sample_video = list(meta_train_df.loc[meta_train_df.original=='vjlzzqnjbb.mp4'].index)
display_image_from_video_list(same_original_fake_train_sample_video)
```

In [ ]:
```
same_original_fake_train_sample_video = list(meta_train_df.loc[meta_train_df.original=='cofxbtbnsf.mp4'].index)
display_image_from_video_list(same_original_fake_train_sample_video)
```

In [ ]:
```
#Test video files
#Let's also look to few of the test data files.
```

In [ ]:
```
test_videos = pd.DataFrame(list(os.listdir(os.path.join(DATA_FOLDER, TEST_FOLDER))), columns=['video'])
```

In [ ]:
```
test_videos.head()
```

In [ ]:
```
display_image_from_video(os.path.join(DATA_FOLDER, TEST_FOLDER, test_videos.iloc[0].video))
```

In [ ]:
```
#Let's look to some more videos from test set.
```

In [ ]:
```
display_image_from_video_list(test_videos.sample(3).video, TEST_FOLDER)
```

In [ ]:
```python
#Face detection
#From (Face Detection using OpenCV) by @serkanpeldek we got and slightly modified the functions to extract face, profile face, eyes and smile.

#The class ObjectDetector initialize the cascade classifier (using the imported resource).
#The function detect uses a method of the CascadeClassifier to detect objects into images - in this case the face, eye, smile or profile face.

class ObjectDetector():
    '''
    Class for Object Detection
    '''
    def __init__(self,object_cascade_path):
        '''
        param: object_cascade_path - path for the *.xml defining the parameters for {face, eye, smile, profile}
        detection algorithm
        source of the haarcascade resource is: https://github.com/opencv/opencv/tree/master/data/haarcascades
        '''

        self.objectCascade=cv.CascadeClassifier(object_cascade_path)


    def detect(self, image, scale_factor=1.3,
               min_neighbors=5,
               min_size=(20,20)):
        '''
        Function return rectangle coordinates of object for given image
        param: image - image to process
        param: scale_factor - scale factor used for object detection
        param: min_neighbors - minimum number of parameters considered during object detection
        param: min_size - minimum size of bounding box for object detected
        '''
        rects=self.objectCascade.detectMultiScale(image,
                                                  scaleFactor=scale_factor,
                                                  minNeighbors=min_neighbors,
                                                  minSize=min_size)
        return rects
```

In [ ]:
```python
#We load the resources for frontal face, eye, smile and profile face detection.

#Then we initialize the ObjectDetector objects defined above with the respective resources, to use CascadeClassfier for each specific task.
```

In [ ]:
```python
#Frontal face, profile, eye and smile  haar cascade loaded
frontal_cascade_path= os.path.join(FACE_DETECTION_FOLDER,'haarcascade_frontalface_default.xml')
eye_cascade_path= os.path.join(FACE_DETECTION_FOLDER,'haarcascade_eye.xml')
profile_cascade_path= os.path.join(FACE_DETECTION_FOLDER,'haarcascade_profileface.xml')
smile_cascade_path= os.path.join(FACE_DETECTION_FOLDER,'haarcascade_smile.xml')

#Detector object created
# frontal face
fd=ObjectDetector(frontal_cascade_path)
# eye
ed=ObjectDetector(eye_cascade_path)
# profile face
pd=ObjectDetector(profile_cascade_path)
```

```
# smile
sd=ObjectDetector(smile_cascade_path)
```

In [ ]:
```
#We also define a function for detection and display of all these specific objects.

#The function call the detect method of the ObjectDetector object. For each object we are using a different shape and color, as following:

#Frontal face: green rectangle;
#Eye: red circle;
#Smile: red rectangle;
#Profile face: blue rectangle.
#Note: due to a huge amount of false positive, we deactivate for now the smile detector.
```

In [ ]:
```
def detect_objects(image, scale_factor, min_neighbors, min_size):
    '''
    Objects detection function
    Identify frontal face, eyes, smile and profile face and display the detected objects over the image
    param: image - the image extracted from the video
    param: scale_factor - scale factor parameter for `detect` function of ObjectDetector object
    param: min_neighbors - min neighbors parameter for `detect` function of ObjectDetector object
    param: min_size - minimum size parameter for f`detect` function of ObjectDetector object
    '''

    image_gray=cv.cvtColor(image, cv.COLOR_BGR2GRAY)


    eyes=ed.detect(image_gray,
                   scale_factor=scale_factor,
                   min_neighbors=min_neighbors,
                   min_size=(int(min_size[0]/2), int(min_size[1]/2)))

    for x, y, w, h in eyes:
        #detected eyes shown in color image
        cv.circle(image,(int(x+w/2),int(y+h/2)),(int((w + h)/4)),(0, 0,255),3)

    # deactivated due to many false positive
    #smiles=sd.detect(image_gray,
    #                 scale_factor=scale_factor,
    #                 min_neighbors=min_neighbors,
    #                 min_size=(int(min_size[0]/2), int(min_size[1]/2)))

    #for x, y, w, h in smiles:
    #    #detected smiles shown in color image
    #    cv.rectangle(image,(x,y),(x+w, y+h),(0, 0,255),3)


    profiles=pd.detect(image_gray,
                   scale_factor=scale_factor,
                   min_neighbors=min_neighbors,
                   min_size=min_size)

    for x, y, w, h in profiles:
        #detected profiles shown in color image
        cv.rectangle(image,(x,y),(x+w, y+h),(255, 0,0),3)
```

```
    faces=fd.detect(image_gray,
                    scale_factor=scale_factor,
                    min_neighbors=min_neighbors,
                    min_size=min_size)

    for x, y, w, h in faces:
        #detected faces shown in color image
        cv.rectangle(image,(x,y),(x+w, y+h),(0, 255,0),3)

    # image
    fig = plt.figure(figsize=(10,10))
    ax = fig.add_subplot(111)
    image = cv.cvtColor(image, cv.COLOR_BGR2RGB)
    ax.imshow(image)
```

In [ ]:
```
#The following function extracts an image
#from a video and then call the function that extracts the face rectangle from the image and display the rectangle above the image.
```

In [ ]:
```
def extract_image_objects(video_file, video_set_folder=TRAIN_SAMPLE_FOLDER):
    '''
    Extract one image from the video and then perform face/eyes/smile/profile detection on the image
    param: video_file - the video from which to extract the image from which we extract the face
    '''
    video_path = os.path.join(DATA_FOLDER, video_set_folder,video_file)
    capture_image = cv.VideoCapture(video_path)
    ret, frame = capture_image.read()
    #frame = cv.cvtColor(frame, cv.COLOR_BGR2RGB)
    detect_objects(image=frame,
                   scale_factor=1.3,
                   min_neighbors=5,
                   min_size=(50, 50))
```

In [ ]:
```
#We apply the function for face detection for a selection of images from train sample videos.
```

In [ ]:
```
same_original_fake_train_sample_video = list(meta_train_df.loc[meta_train_df.original=='vjlzzqnjbb.mp4'].index)
for video_file in same_original_fake_train_sample_video[1:4]:
    print(video_file)
    extract_image_objects(video_file)
```

In [ ]:
```
train_subsample_video = list(meta_train_df.sample(3).index)
for video_file in train_subsample_video:
    print(video_file)
    extract_image_objects(video_file)
```

In [ ]:
```
subsample_test_videos = list(test_videos.sample(3).video)
for video_file in subsample_test_videos:
    print(video_file)
    extract_image_objects(video_file, TEST_FOLDER)
```

In [ ]:
```
fake_videos = list(meta_train_df.loc[meta_train_df.label=='FAKE'].index)
```

In [ ]:
```
from IPython.display import HTML
from base64 import b64encode
```

```
def play_video(video_file, subset=TRAIN_SAMPLE_FOLDER):
    '''
    Display video
    param: video_file - the name of the video file to display
    param: subset - the folder where the video file is located (can be TRAIN_SAMPLE_FOLDER or TEST_Folder)
    '''
    video_url = open(os.path.join(DATA_FOLDER, subset,video_file),'rb').read()
    data_url = "data:video/mp4;base64," + b64encode(video_url).decode()
    return HTML("""<video width=500 controls><source src="%s" type="video/mp4"></video>""" % data_url)
```

In [ ]:
```
play_video(fake_videos[0])
```

In [ ]:
```
play_video(fake_videos[1])
```

In [ ]:
```
play_video(fake_videos[2])
```

In [ ]:
```
play_video(fake_videos[3])
```

In [ ]:
```
play_video(fake_videos[4])
```

In [ ]:
```
play_video(fake_videos[5])
```

In [ ]:
```

```