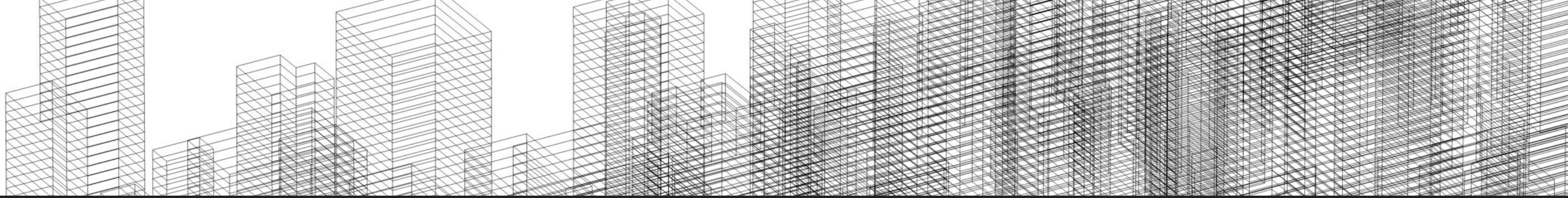


Deepfake Detection with OpenCV

A Project by Noah Bledsoe, Michael Hammond, and Pralad Neupane

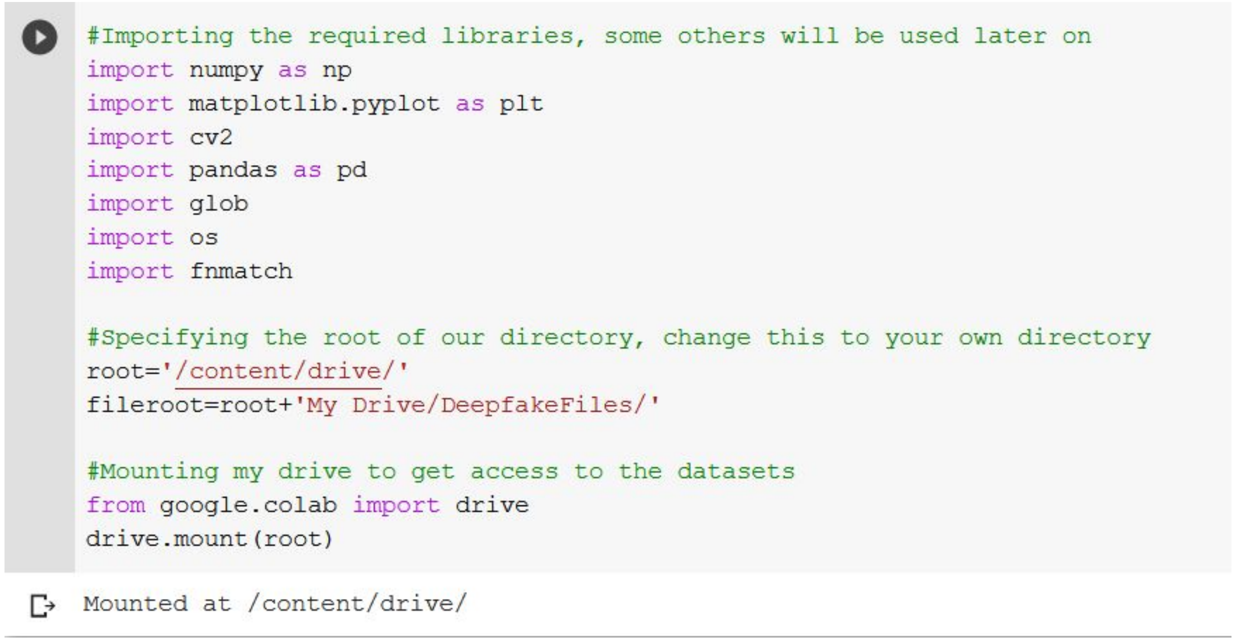
The Problem

- It’s become incredibly easy to replace the face of one person in a video with the face of another person by using a pre-trained generative adversarial network (GAN).
- Constant social media picture uploads make it very easy to train these networks into creating very realistic “deepfakes” due to the massive amount of data
- Deepfakes can be used to influence elections, ruin reputations, create fake pornography, and further increase the mistrust in media

The Goal

- The goal is to use a machine learning-based method to differentiate between altered deepfake videos and non-altered original videos
- Steps:
  - Get a database of videos that contain both deepfakes and originals in order to train and test our method
  - Use various methods to compare the differences between deepfake and original frames
  - Use these differences as features to train a machine-learning algorithm, a support vector machine for example
  - After training the data, compare our results with the testing set to evaluate how our algorithm performed by using metrics such as an ROC curve

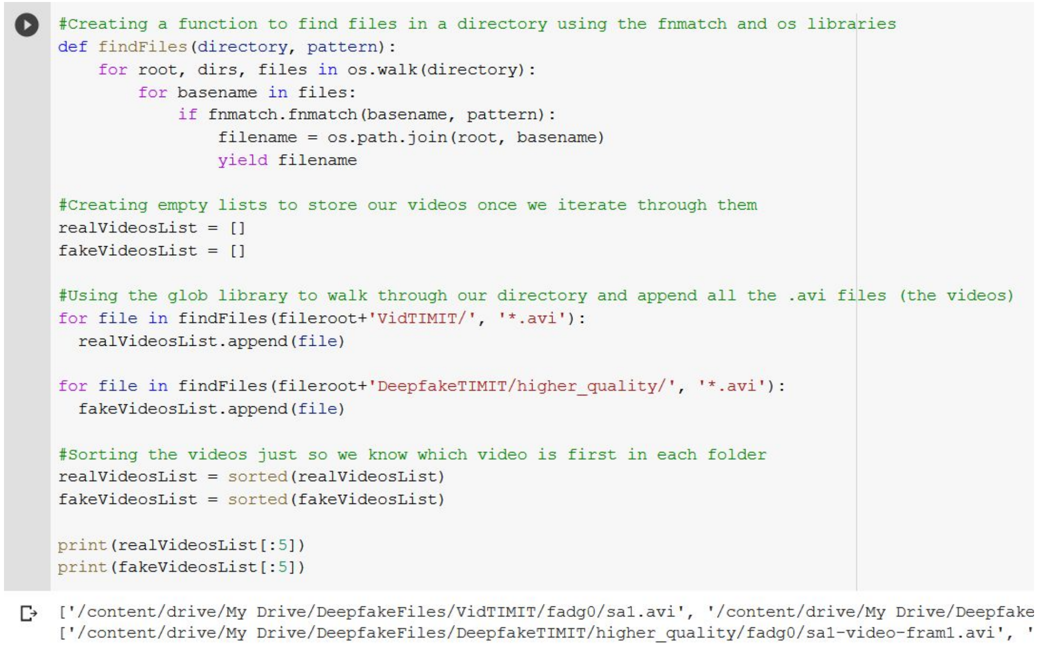
Importing Libraries and Mounting Google Drive



```
#Importing the required libraries, some others will be used later on
import numpy as np
import matplotlib.pyplot as plt
import cv2
import pandas as pd
import glob
import os
import fnmatch

#Specifying the root of our directory, change this to your own directory
root='/content/drive/'
fileroot=root+'My Drive/DeepfakeFiles/'

#Mounting my drive to get access to the datasets
from google.colab import drive
drive.mount(root)
```

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

Iterating Through the Deepfake and Original Video Folders

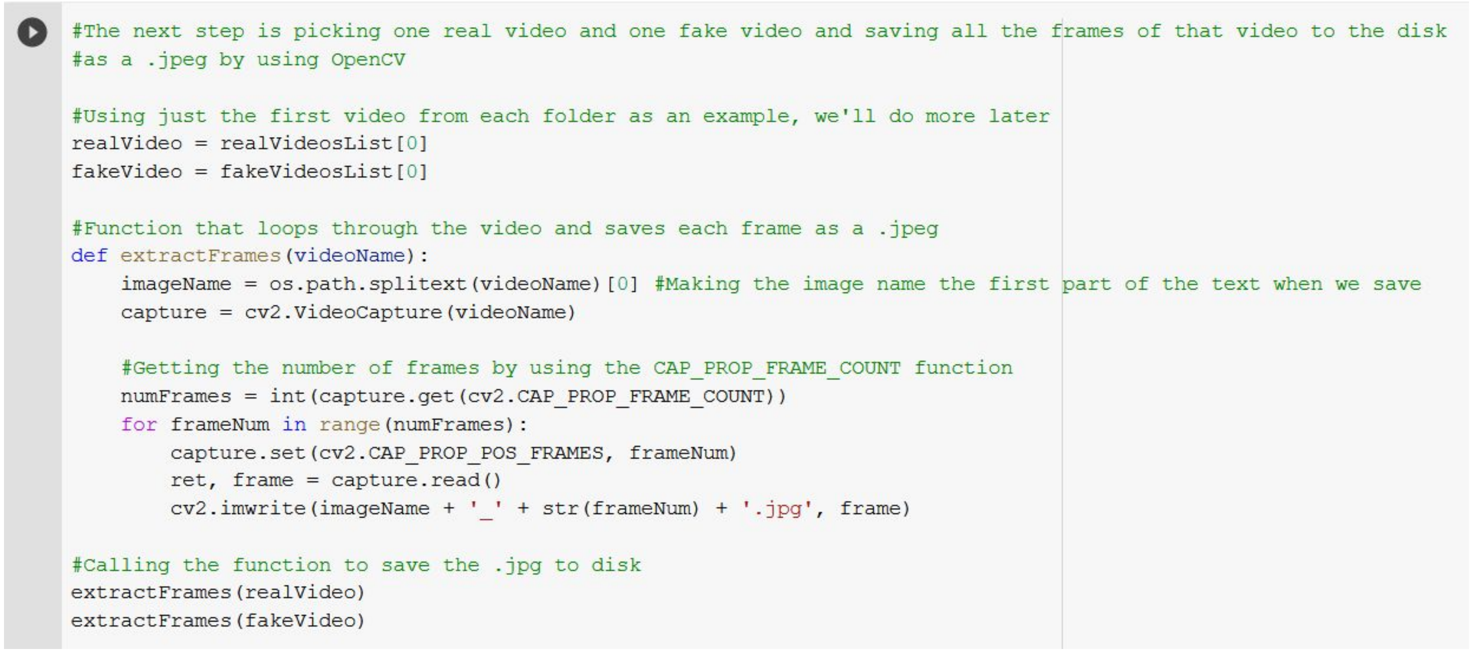


```
#Creating a function to find files in a directory using the fnmatch and os libraries
def findFiles(directory, pattern):
    for root, dirs, files in os.walk(directory):
        for basename in files:
            if fnmatch.fnmatch(basename, pattern):
                filename = os.path.join(root, basename)
                yield filename

#Creating empty lists to store our videos once we iterate through them
realVideosList = []
fakeVideosList = []

#Using the glob library to walk through our directory and append all the .avi files (the videos)
for file in findFiles(fileroot+'VidTIMIT/', '*.avi'):
  realVideosList.append(file)

for file in findFiles(fileroot+'DeepfakeTIMIT/higher_quality/', '*.avi'):
  fakeVideosList.append(file)

#Sorting the videos just so we know which video is first in each folder
realVideosList = sorted(realVideosList)
fakeVideosList = sorted(fakeVideosList)

print(realVideosList[:5])
print(fakeVideosList[:5])
```

```
['/content/drive/My Drive/DeepfakeFiles/VidTIMIT/fadg0/sa1.avi', '/content/drive/My Drive/Deepfake
['/content/drive/My Drive/DeepfakeFiles/DeepfakeTIMIT/higher_quality/fadg0/sa1-video-fram1.avi', '
```

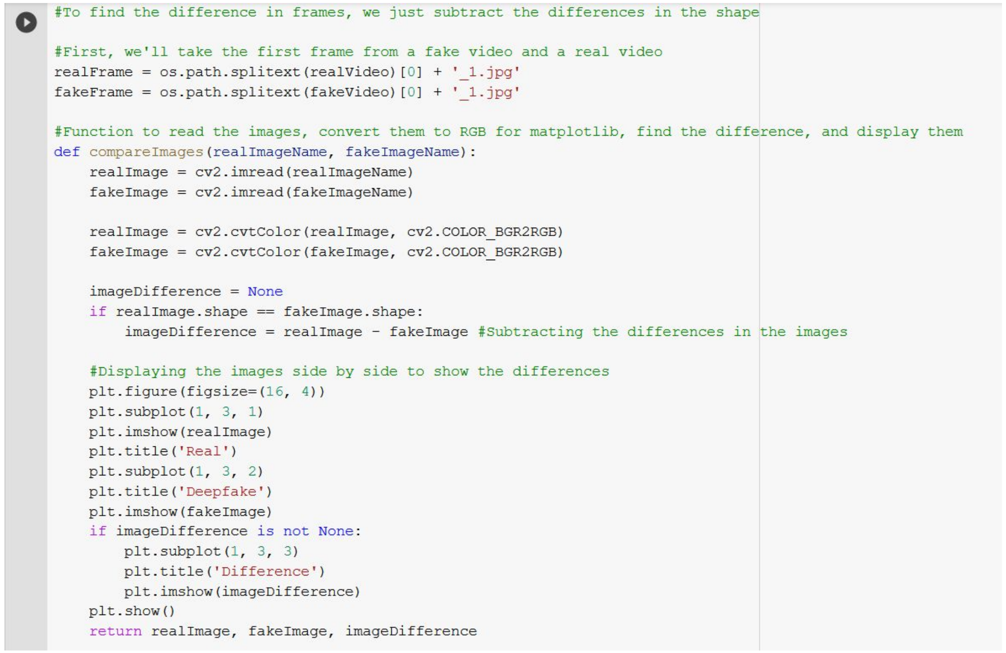
Extracting Each Frame From a Video

In order to compare the two, we need to analyze each frame, not the entire video.



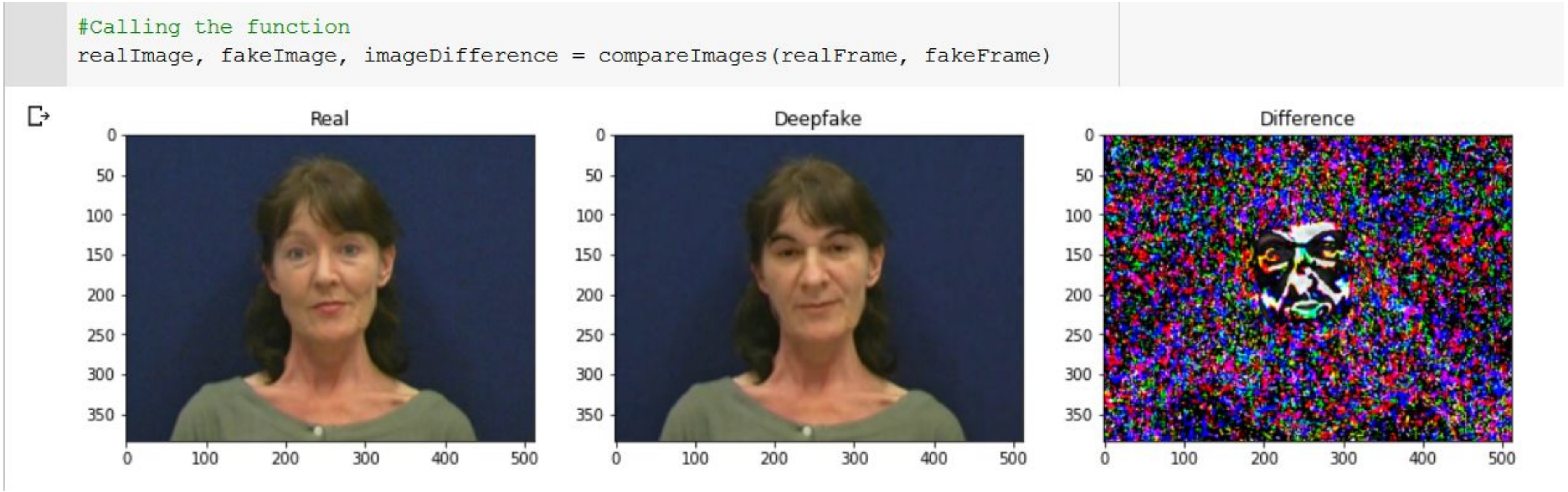
```python
#The next step is picking one real video and one fake video and saving all the frames of that video to the disk
#as a .jpeg by using OpenCV

#Using just the first video from each folder as an example, we'll do more later
realVideo = realVideosList[0]
fakeVideo = fakeVideosList[0]

#Function that loops through the video and saves each frame as a .jpeg
def extractFrames(videoName):
    imageName = os.path.splitext(videoName)[0] #Making the image name the first part of the text when we save
    capture = cv2.VideoCapture(videoName)

    #Getting the number of frames by using the CAP_PROP_FRAME_COUNT function
    numFrames = int(capture.get(cv2.CAP_PROP_FRAME_COUNT))
    for frameNum in range(numFrames):
        capture.set(cv2.CAP_PROP_POS_FRAMES, frameNum)
        ret, frame = capture.read()
        cv2.imwrite(imageName + '_' + str(frameNum) + '.jpg', frame)

#Calling the function to save the .jpg to disk
extractFrames(realVideo)
extractFrames(fakeVideo)
```

Finding the Difference in Frames



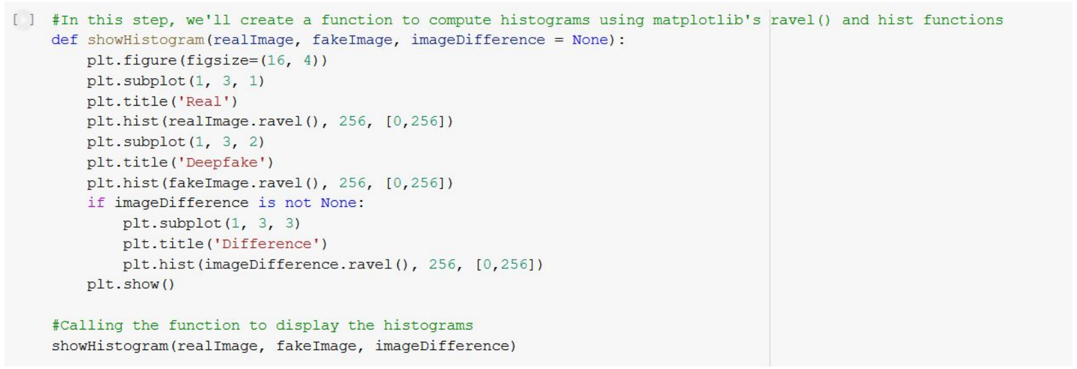
```python
#To find the difference in frames, we just subtract the differences in the shape

#First, we'll take the first frame from a fake video and a real video
realFrame = os.path.splitext(realVideo)[0] + '_1.jpg'
fakeFrame = os.path.splitext(fakeVideo)[0] + '_1.jpg'

#Function to read the images, convert them to RGB for matplotlib, find the difference, and display them
def compareImages(realImageName, fakeImageName):
    realImage = cv2.imread(realImageName)
    fakeImage = cv2.imread(fakeImageName)

    realImage = cv2.cvtColor(realImage, cv2.COLOR_BGR2RGB)
    fakeImage = cv2.cvtColor(fakeImage, cv2.COLOR_BGR2RGB)

    imageDifference = None
    if realImage.shape == fakeImage.shape:
        imageDifference = realImage - fakeImage #Subtracting the differences in the images

    #Displaying the images side by side to show the differences
    plt.figure(figsize=(16, 4))
    plt.subplot(1, 3, 1)
    plt.imshow(realImage)
    plt.title('Real')
    plt.subplot(1, 3, 2)
    plt.title('Deepfake')
    plt.imshow(fakeImage)
    if imageDifference is not None:
        plt.subplot(1, 3, 3)
        plt.title('Difference')
        plt.imshow(imageDifference)
    plt.show()
    return realImage, fakeImage, imageDifference
```

Showing the Difference in Frames

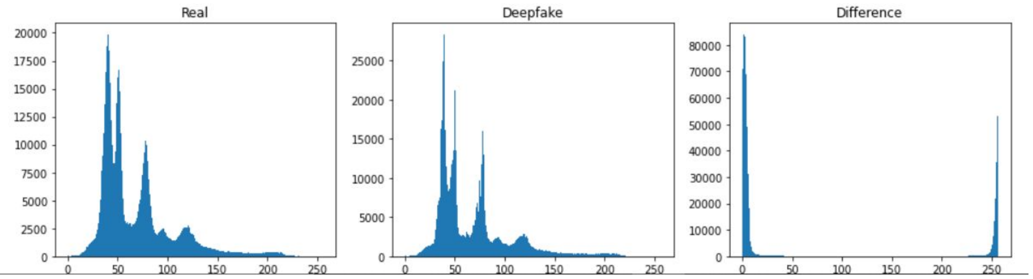


```
#Calling the function
realImage, fakeImage, imageDifference = compareImages(realFrame, fakeFrame)
```

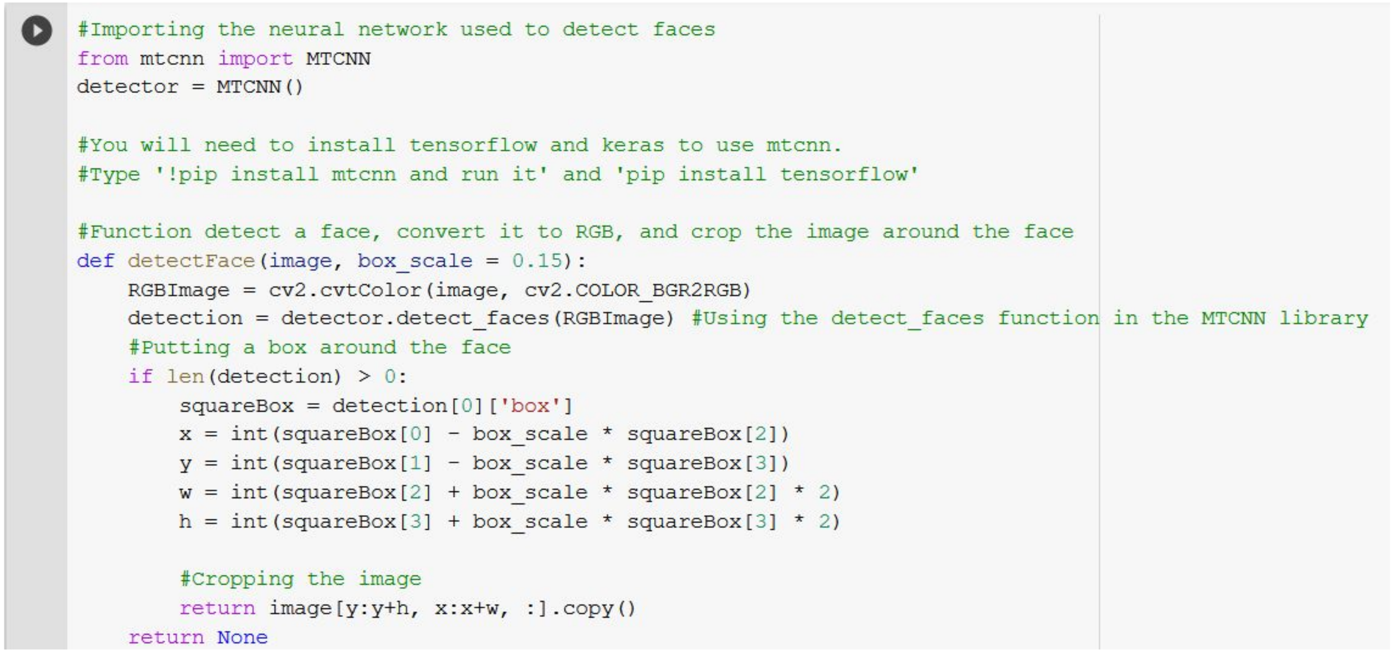
Exploratory Analysis of Frames



```
#In this step, we'll create a function to compute histograms using matplotlib's ravel() and hist functions
def showHistogram(realImage, fakeImage, imageDifference = None):
    plt.figure(figsize=(16, 4))
    plt.subplot(1, 3, 1)
    plt.title('Real')
    plt.hist(realImage.ravel(), 256, [0,256])
    plt.subplot(1, 3, 2)
    plt.title('Deepfake')
    plt.hist(fakeImage.ravel(), 256, [0,256])
    if imageDifference is not None:
        plt.subplot(1, 3, 3)
        plt.title('Difference')
        plt.hist(imageDifference.ravel(), 256, [0,256])
    plt.show()

#Calling the function to display the histograms
showHistogram(realImage, fakeImage, imageDifference)
```

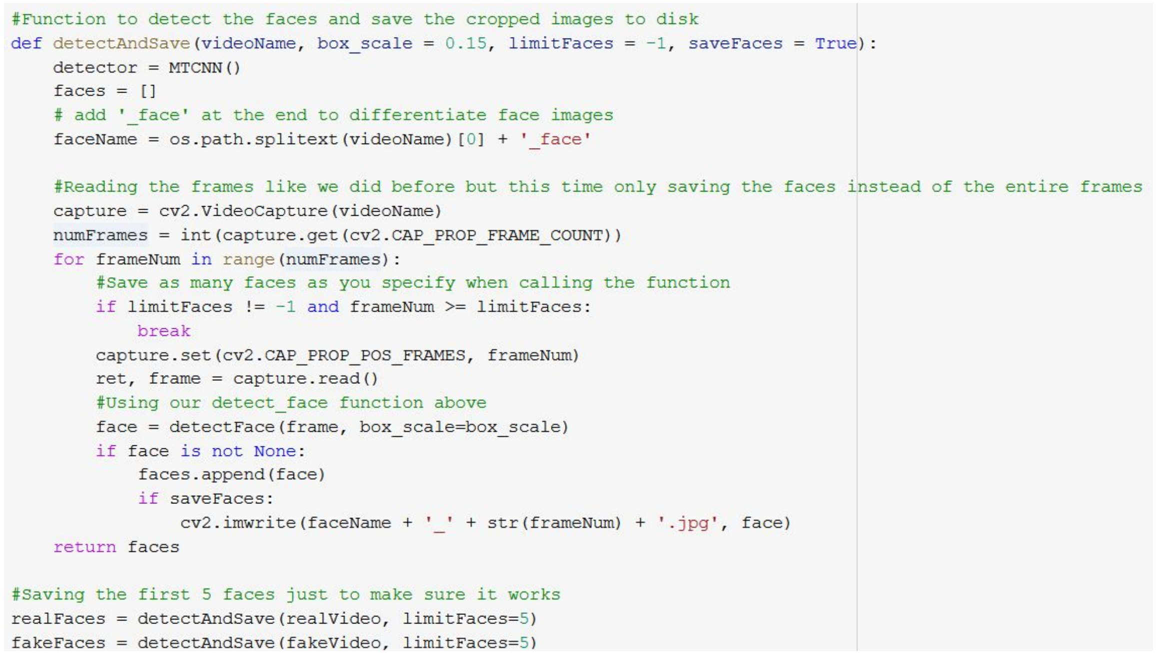


Using a Neural Network to Detect Faces in a Frame



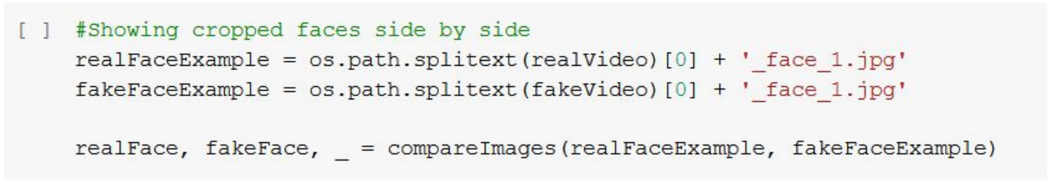
```
#Importing the neural network used to detect faces
from mtcnn import MTCNN
detector = MTCNN()

#You will need to install tensorflow and keras to use mtcnn.
#Type '!pip install mtcnn and run it' and 'pip install tensorflow'

#Function detect a face, convert it to RGB, and crop the image around the face
def detectFace(image, box_scale = 0.15):
    RGBImage = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    detection = detector.detect_faces(RGBImage) #Using the detect_faces function in the MTCNN library
    #Putting a box around the face
    if len(detection) > 0:
        squareBox = detection[0]['box']
        x = int(squareBox[0] - box_scale * squareBox[2])
        y = int(squareBox[1] - box_scale * squareBox[3])
        w = int(squareBox[2] + box_scale * squareBox[2] * 2)
        h = int(squareBox[3] + box_scale * squareBox[3] * 2)

        #Cropping the image
        return image[y:y+h, x:x+w, :].copy()
    return None
```

Saving the Faces to Disc

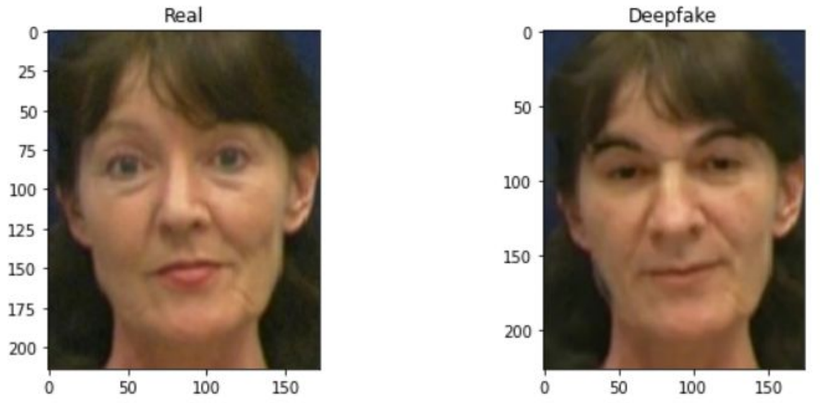


```python
#Function to detect the faces and save the cropped images to disk
def detectAndSave(videoName, box_scale = 0.15, limitFaces = -1, saveFaces = True):
    detector = MTCNN()
    faces = []
    # add '_face' at the end to differentiate face images
    faceName = os.path.splitext(videoName)[0] + '_face'

    #Reading the frames like we did before but this time only saving the faces instead of the entire frames
    capture = cv2.VideoCapture(videoName)
    numFrames = int(capture.get(cv2.CAP_PROP_FRAME_COUNT))
    for frameNum in range(numFrames):
        #Save as many faces as you specify when calling the function
        if limitFaces != -1 and frameNum >= limitFaces:
            break
        capture.set(cv2.CAP_PROP_POS_FRAMES, frameNum)
        ret, frame = capture.read()
        #Using our detect_face function above
        face = detectFace(frame, box_scale=box_scale)
        if face is not None:
            faces.append(face)
            if saveFaces:
                cv2.imwrite(faceName + '_' + str(frameNum) + '.jpg', face)
    return faces

#Saving the first 5 faces just to make sure it works
realFaces = detectAndSave(realVideo, limitFaces=5)
fakeFaces = detectAndSave(fakeVideo, limitFaces=5)
```

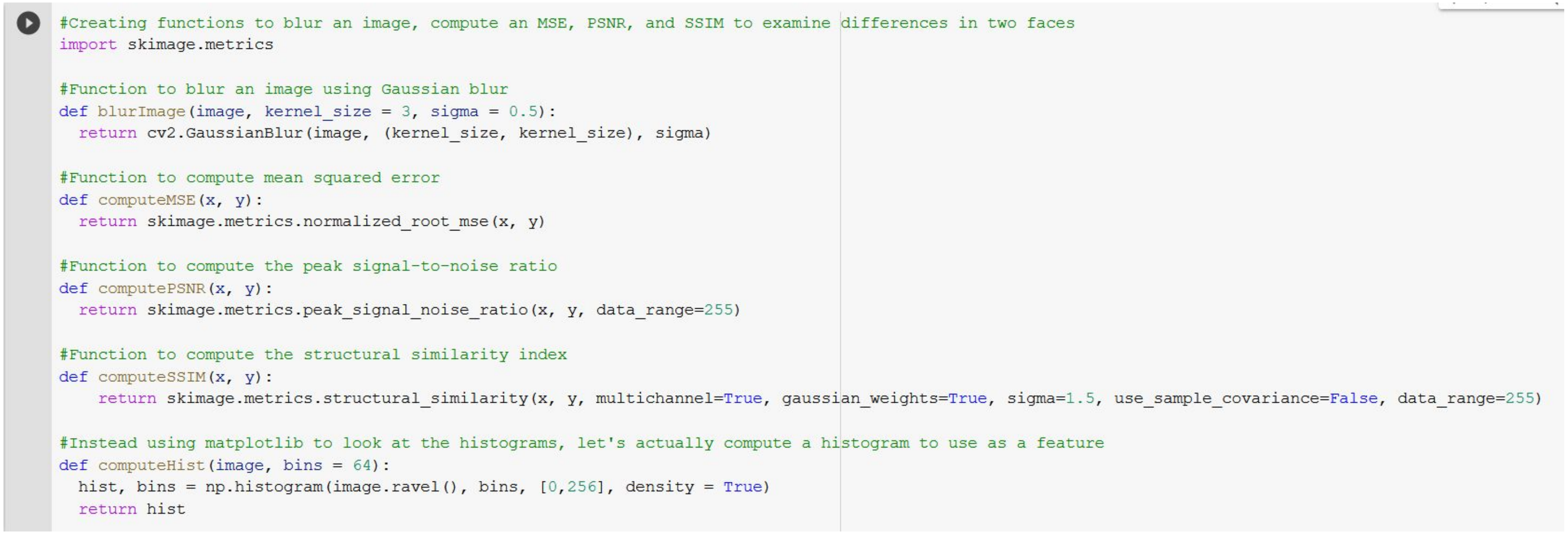
Showcasing Cropped Facial Images



```
[ ] #Showing cropped faces side by side
    realFaceExample = os.path.splitext(realVideo)[0] + '_face_1.jpg'
    fakeFaceExample = os.path.splitext(fakeVideo)[0] + '_face_1.jpg'

    realFace, fakeFace, _ = compareImages(realFaceExample, fakeFaceExample)
```

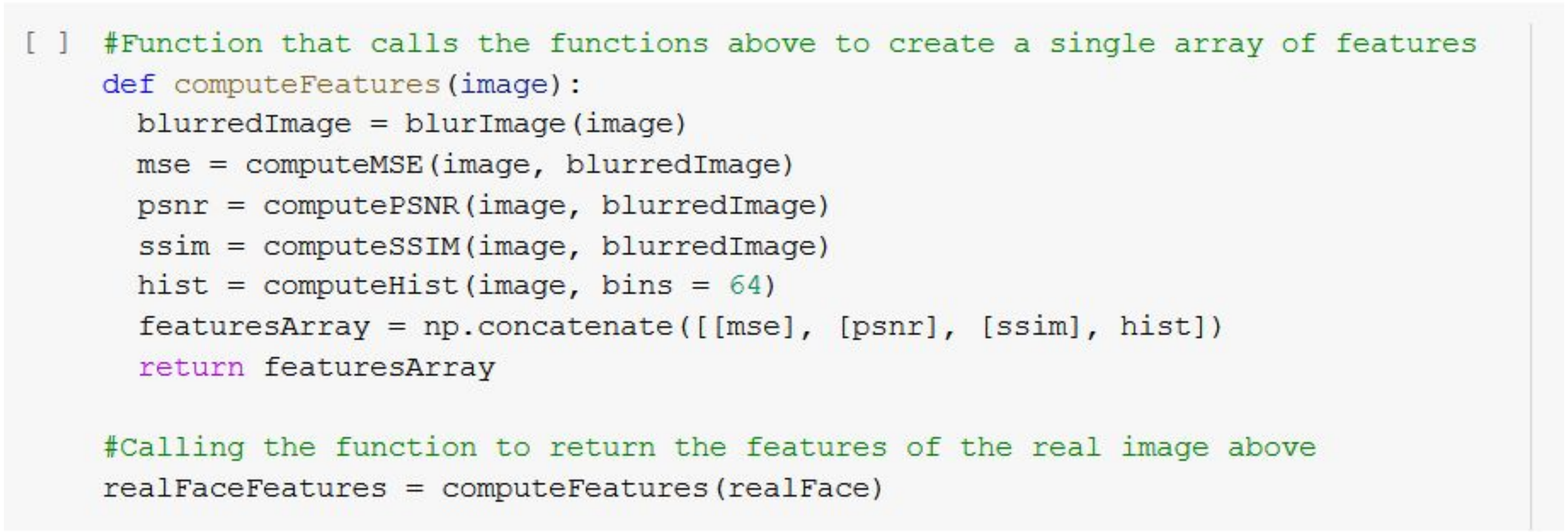


Identifying Features in Faces Using SKImage



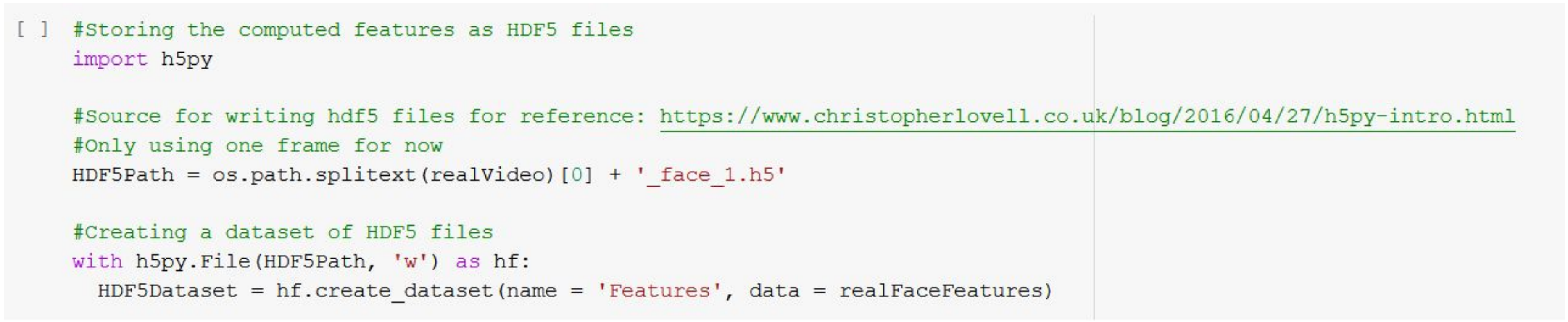
```
#Creating functions to blur an image, compute an MSE, PSNR, and SSIM to examine differences in two faces
import skimage.metrics

#Function to blur an image using Gaussian blur
def blurImage(image, kernel_size = 3, sigma = 0.5):
  return cv2.GaussianBlur(image, (kernel_size, kernel_size), sigma)

#Function to compute mean squared error
def computeMSE(x, y):
  return skimage.metrics.normalized_root_mse(x, y)

#Function to compute the peak signal-to-noise ratio
def computePSNR(x, y):
  return skimage.metrics.peak_signal_noise_ratio(x, y, data_range=255)

#Function to compute the structural similarity index
def computeSSIM(x, y):
    return skimage.metrics.structural_similarity(x, y, multichannel=True, gaussian_weights=True, sigma=1.5, use_sample_covariance=False, data_range=255)

#Instead using matplotlib to look at the histograms, let's actually compute a histogram to use as a feature
def computeHist(image, bins = 64):
  hist, bins = np.histogram(image.ravel(), bins, [0,256], density = True)
  return hist
```

Concatenating All Features into Array



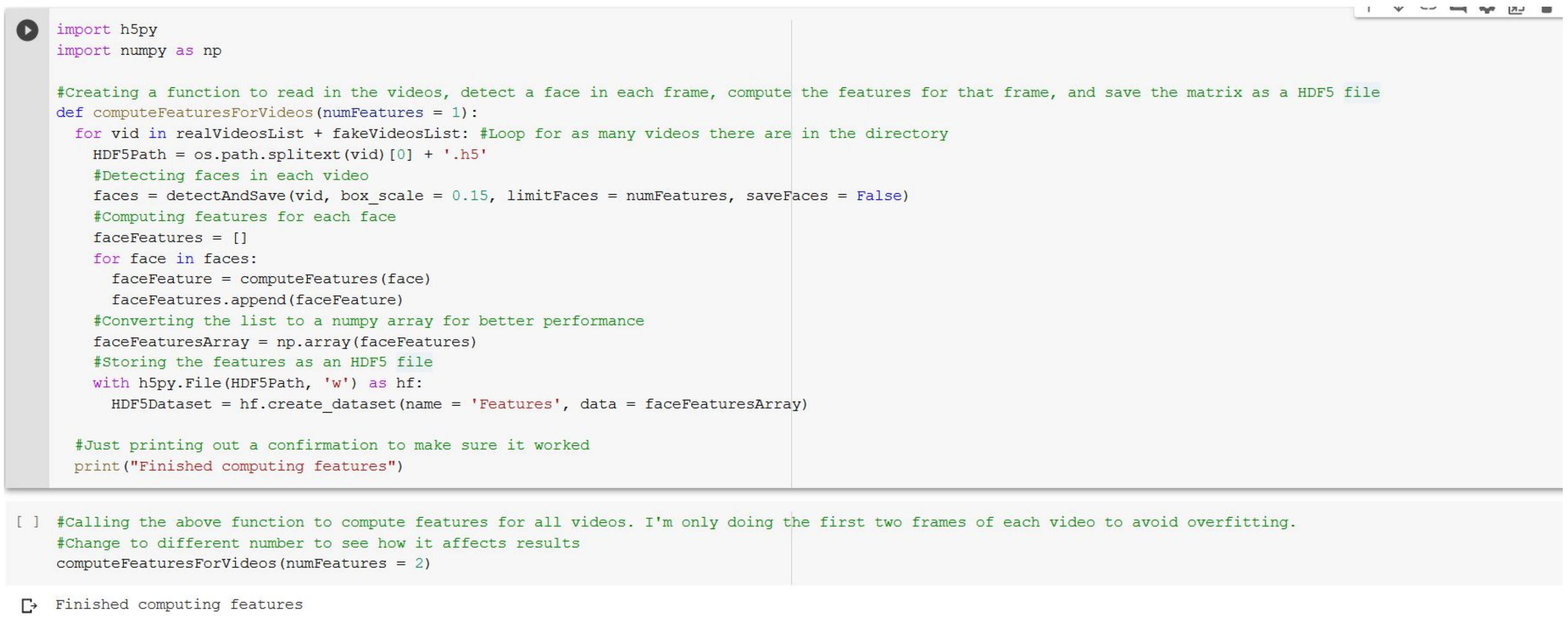
```
[ ]  #Function that calls the functions above to create a single array of features
     def computeFeatures(image):
       blurredImage = blurImage(image)
       mse = computeMSE(image, blurredImage)
       psnr = computePSNR(image, blurredImage)
       ssim = computeSSIM(image, blurredImage)
       hist = computeHist(image, bins = 64)
       featuresArray = np.concatenate([[mse], [psnr], [ssim], hist])
       return featuresArray

     #Calling the function to return the features of the real image above
     realFaceFeatures = computeFeatures(realFace)
```

Storing Features Array as a HDF5 File



```
[ ]  #Storing the computed features as HDF5 files
     import h5py

     #Source for writing hdf5 files for reference: https://www.christopherlovell.co.uk/blog/2016/04/27/h5py-intro.html
     #Only using one frame for now
     HDF5Path = os.path.splitext(realVideo)[0] + '_face_1.h5'

     #Creating a dataset of HDF5 files
     with h5py.File(HDF5Path, 'w') as hf:
       HDF5Dataset = hf.create_dataset(name = 'Features', data = realFaceFeatures)
```

Computing Features for all Frames

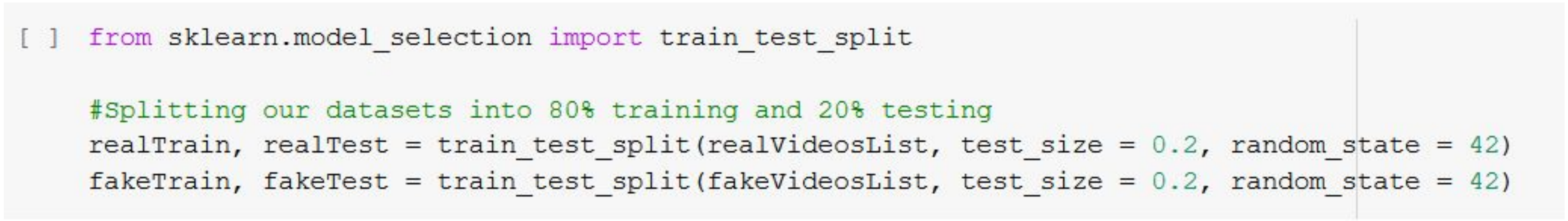


```
import h5py
import numpy as np

#Creating a function to read in the videos, detect a face in each frame, compute the features for that frame, and save the matrix as a HDF5 file
def computeFeaturesForVideos(numFeatures = 1):
  for vid in realVideosList + fakeVideosList: #Loop for as many videos there are in the directory
    HDF5Path = os.path.splitext(vid)[0] + '.h5'
    #Detecting faces in each video
    faces = detectAndSave(vid, box_scale = 0.15, limitFaces = numFeatures, saveFaces = False)
    #Computing features for each face
    faceFeatures = []
    for face in faces:
      faceFeature = computeFeatures(face)
      faceFeatures.append(faceFeature)
    #Converting the list to a numpy array for better performance
    faceFeaturesArray = np.array(faceFeatures)
    #Storing the features as an HDF5 file
    with h5py.File(HDF5Path, 'w') as hf:
      HDF5Dataset = hf.create_dataset(name = 'Features', data = faceFeaturesArray)

  #Just printing out a confirmation to make sure it worked
  print("Finished computing features")
```

```
[ ] #Calling the above function to compute features for all videos. I'm only doing the first two frames of each video to avoid overfitting.
    #Change to different number to see how it affects results
    computeFeaturesForVideos(numFeatures = 2)
```

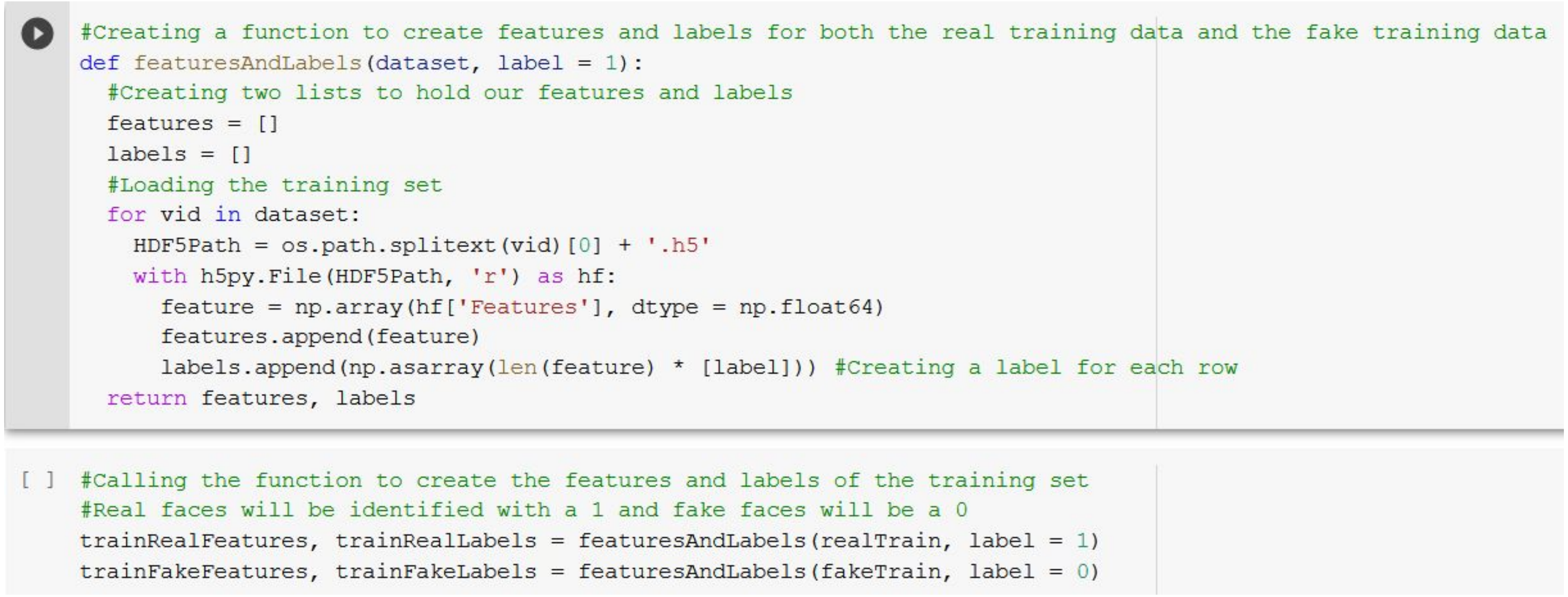
```
Finished computing features
```

Splitting our Datasets into Training/Testing



```
[ ]  from sklearn.model_selection import train_test_split

     #Splitting our datasets into 80% training and 20% testing
     realTrain, realTest = train_test_split(realVideosList, test_size = 0.2, random_state = 42)
     fakeTrain, fakeTest = train_test_split(fakeVideosList, test_size = 0.2, random_state = 42)
```

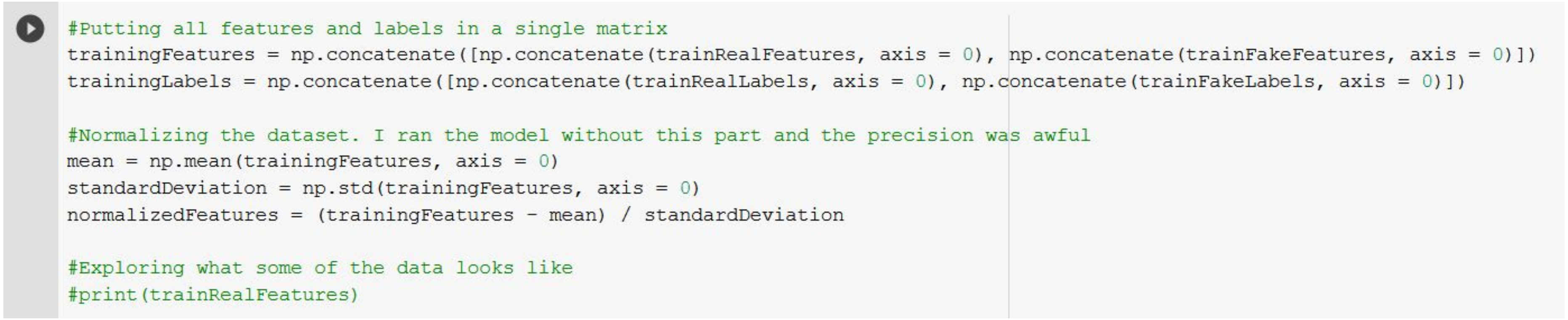
Computing Features and Labels



```
#Creating a function to create features and labels for both the real training data and the fake training data
def featuresAndLabels(dataset, label = 1):
  #Creating two lists to hold our features and labels
  features = []
  labels = []
  #Loading the training set
  for vid in dataset:
    HDF5Path = os.path.splitext(vid)[0] + '.h5'
    with h5py.File(HDF5Path, 'r') as hf:
      feature = np.array(hf['Features'], dtype = np.float64)
      features.append(feature)
      labels.append(np.asarray(len(feature) * [label])) #Creating a label for each row
  return features, labels
```

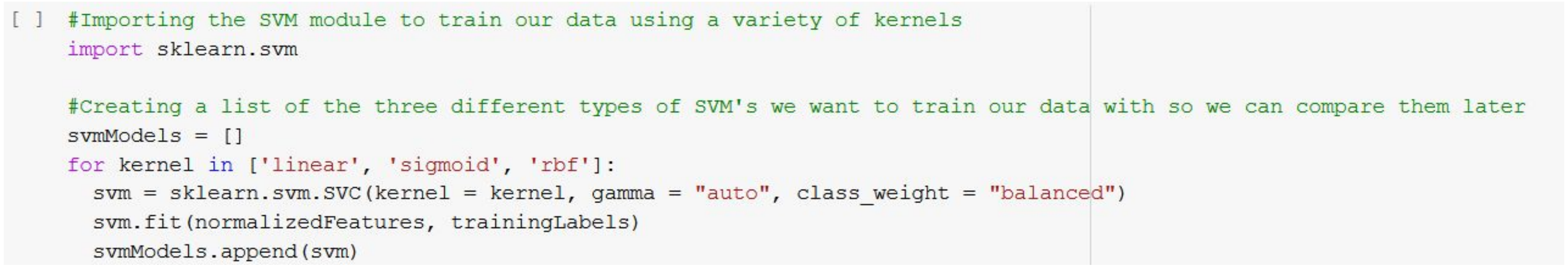
```
#Calling the function to create the features and labels of the training set
#Real faces will be identified with a 1 and fake faces will be a 0
trainRealFeatures, trainRealLabels = featuresAndLabels(realTrain, label = 1)
trainFakeFeatures, trainFakeLabels = featuresAndLabels(fakeTrain, label = 0)
```

Concatenating Features and Labels/Normalizing the Data



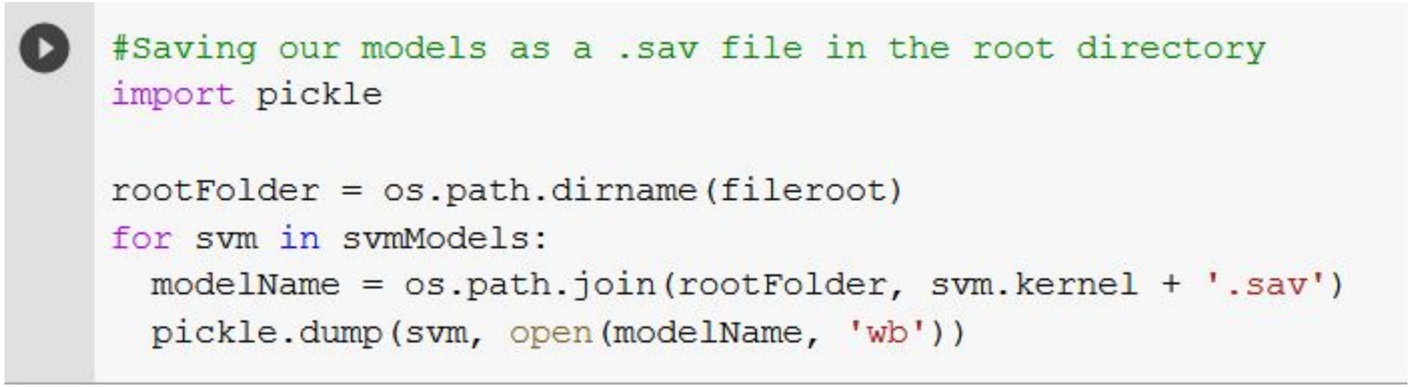
```
#Putting all features and labels in a single matrix
trainingFeatures = np.concatenate([np.concatenate(trainRealFeatures, axis = 0), np.concatenate(trainFakeFeatures, axis = 0)])
trainingLabels = np.concatenate([np.concatenate(trainRealLabels, axis = 0), np.concatenate(trainFakeLabels, axis = 0)])

#Normalizing the dataset. I ran the model without this part and the precision was awful
mean = np.mean(trainingFeatures, axis = 0)
standardDeviation = np.std(trainingFeatures, axis = 0)
normalizedFeatures = (trainingFeatures - mean) / standardDeviation

#Exploring what some of the data looks like
#print(trainRealFeatures)
```

Running the Normalized Data Through Three SVM Kernels



```
[ ]  #Importing the SVM module to train our data using a variety of kernels
     import sklearn.svm

     #Creating a list of the three different types of SVM's we want to train our data with so we can compare them later
     svmModels = []
     for kernel in ['linear', 'sigmoid', 'rbf']:
       svm = sklearn.svm.SVC(kernel = kernel, gamma = "auto", class_weight = "balanced")
       svm.fit(normalizedFeatures, trainingLabels)
       svmModels.append(svm)
```

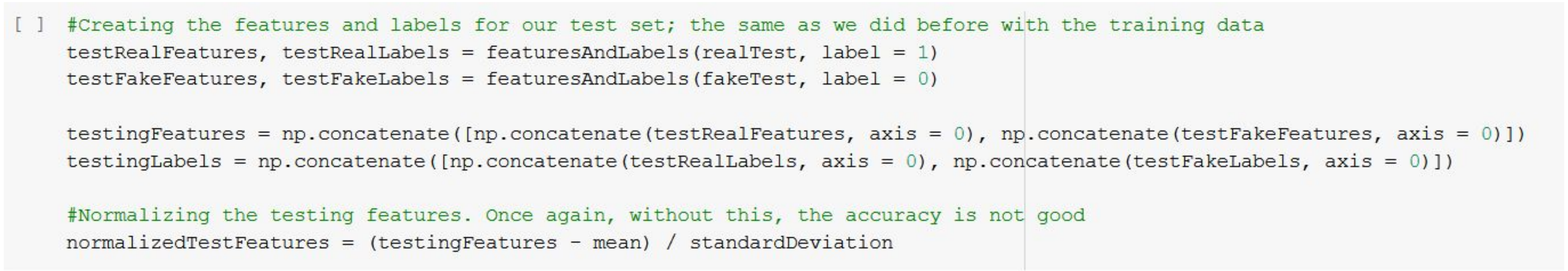
Using Pickle to Save The Models



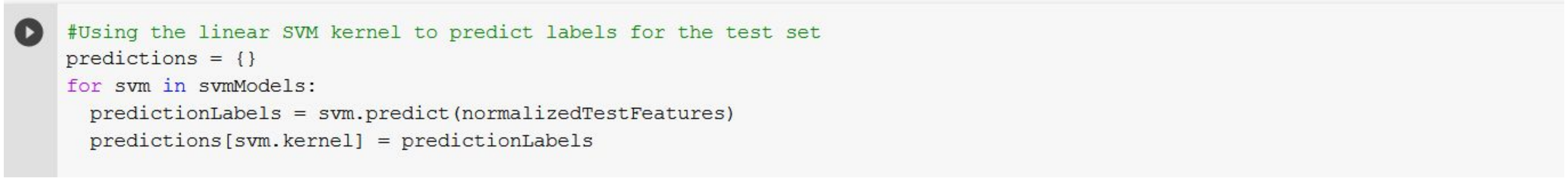
```
#Saving our models as a .sav file in the root directory
import pickle

rootFolder = os.path.dirname(fileroot)
for svm in svmModels:
  modelName = os.path.join(rootFolder, svm.kernel + '.sav')
  pickle.dump(svm, open(modelName, 'wb'))
```

Using our Trained SVM to Make Predictions with the Testing Set

The prediction will return a 1 if the model thinks the image is real and 0 if it thinks the image is fake

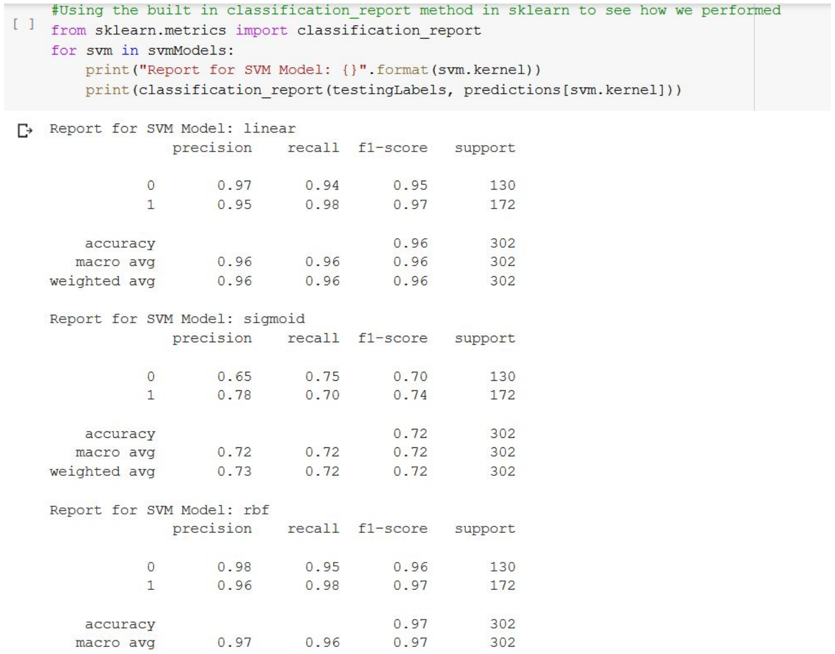


```
#Creating the features and labels for our test set; the same as we did before with the training data
testRealFeatures, testRealLabels = featuresAndLabels(realTest, label = 1)
testFakeFeatures, testFakeLabels = featuresAndLabels(fakeTest, label = 0)

testingFeatures = np.concatenate([np.concatenate(testRealFeatures, axis = 0), np.concatenate(testFakeFeatures, axis = 0)])
testingLabels = np.concatenate([np.concatenate(testRealLabels, axis = 0), np.concatenate(testFakeLabels, axis = 0)])

#Normalizing the testing features. Once again, without this, the accuracy is not good
normalizedTestFeatures = (testingFeatures - mean) / standardDeviation
```



```
#Using the linear SVM kernel to predict labels for the test set
predictions = {}
for svm in svmModels:
  predictionLabels = svm.predict(normalizedTestFeatures)
  predictions[svm.kernel] = predictionLabels
```

Seeing how the Different SVM Kernels Performed

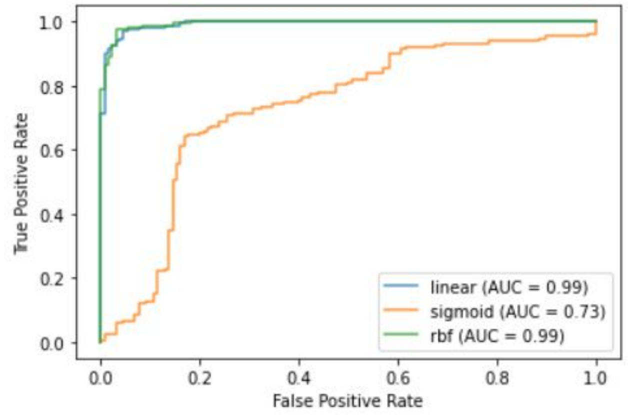


```
#Using the built in classification_report method in sklearn to see how we performed
from sklearn.metrics import classification_report
for svm in svmModels:
    print("Report for SVM Model: {}".format(svm.kernel))
    print(classification_report(testingLabels, predictions[svm.kernel]))
```

```
Report for SVM Model: linear
              precision    recall  f1-score   support

           0       0.97      0.94      0.95       130
           1       0.95      0.98      0.97       172

    accuracy                           0.96       302
   macro avg       0.96      0.96      0.96       302
weighted avg       0.96      0.96      0.96       302

Report for SVM Model: sigmoid
              precision    recall  f1-score   support

           0       0.65      0.75      0.70       130
           1       0.78      0.70      0.74       172

    accuracy                           0.72       302
   macro avg       0.72      0.72      0.72       302
weighted avg       0.73      0.72      0.72       302

Report for SVM Model: rbf
              precision    recall  f1-score   support

           0       0.98      0.95      0.96       130
           1       0.96      0.98      0.97       172

    accuracy                           0.97       302
   macro avg       0.97      0.96      0.97       302
```

Analyzing the Results via ROC Curve

```
[ ]  #Lastly, we'll graph the ROC curve for each one to get another metric of how it performed
     from sklearn.metrics import plot_roc_curve
     for svm in svmModels:
       graph = plot_roc_curve(svm, normalizedTestFeatures, testingLabels, ax = plt.gca(), alpha = 0.8, name = svm.kernel)

     plt.show()
```


True Positive Rate
False Positive Rate
linear (AUC = 0.99)
sigmoid (AUC = 0.73)
rbf (AUC = 0.99)