EE 417 - Computer Vision

Term Project Report

AI Automated Face Morphing and a Novel Approach to Eigenfaces Using Face Morphing

Emre Duman / 26347

Tan Çetiner / 27024

Importance of the Problem and Problem Definition:

Our project aim is twofold. First, we would like to introduce an AI assisted face morphing technique –the use of machine learning techniques to automatically generate a smooth transition between two or more faces. Secondly, we would like to explore a novel use case, namely dataset optimization technique, or dataset pre-processing technique for an eigenfaces dataset. The project is relevant to the study of Computer Vision, since the techniques discussed in this paper, eigenfaces and face morphing, form the basis of some of the still existing face detection systems.

Face morphing is a technique that allows blending two or more faces to create a new, composite face. This technique has a wide range of applications, including entertainment, digital art, and biometrics. In the entertainment industry, face morphing is used to create special effects in movies and videos. In digital art, it is used to create composite characters or to combine the features of different people into a single image. In biometrics, face morphing is used to create synthetic faces for testing face recognition systems or to protect the privacy of individuals.

In recent years, the advancement of computer vision and machine learning techniques has made it possible to create highly realistic face morphs. However, there are still many challenges to be addressed in order to achieve natural and seamless face morphing. One of the main challenges is to extract the most important features of each face and use them to create a new, composite face that looks natural and plausible. Another challenge is to handle variations in lighting, pose, and expression.

Eigenface methods are based on the idea of representing each face as a linear combination of a set of basis vectors called eigenfaces. These eigenfaces are obtained by applying principal component analysis (PCA) to a dataset of face images. The eigenfaces capture the most important features of each face, such as the shape of the eyes, nose, and mouth, and can be used to create new, composite faces.

It is well documented in literature that the Eigenface method faces two major setbacks. First, Eigenfaces algorithm is not robust against light and pose changes. Second, data scarcity severely limits the accuracy of the algorithm. It is here that the face morphing algorithm using AI can aid the Eigenfaces algorithm. By means of a single run preprocessing done on the Eigenface dataset, the eigenfaces can be standardized to a certain degree. We demonstrate this by testing our algorithm and Morph processing, on WLF dataset, which in contrast to yalefaces dataset includes abject angles and poses that are hard to capture for the Eigenface algorithm.

In the context of the Computer Vision course, we can say that our project pertains to the “reconstruction” task of Computer Vision. From the topics listed throughout the course, feature detection/extraction, and especially face detection is a related topic for this work presented in this note.

Problem Formulation and Solution Method:

The problem we are addressing in this project is how to improve the quality of eigenface representations using face morphing techniques. Eigenface methods are a popular approach for face recognition, which represent each face as a linear combination of a set of basis vectors called eigenfaces. These eigenfaces are obtained by applying principal component analysis (PCA) to a dataset of face images. The idea behind eigenface methods is that the eigenfaces capture the most important features of each face and can be used to represent faces in a compact and efficient way. However, there are still many challenges to be addressed in order to achieve accurate and high-quality eigenface representations.

One of the main challenges in eigenface methods is the quality of the eigenfaces. The quality of the eigenfaces depends on the quality of the dataset used to train the model. A dataset that is not representative of the population or that contains variations in lighting, pose, and expression can lead to poor quality eigenfaces. This, in turn, can lead to poor face recognition performance.

Another challenge is to handle the variation in the faces, eigenface methods are linear in nature and they might not be able to handle the non-linear variations in the face. Hence, the eigenface representations may not be able to capture the variations in the face, leading to poor face recognition performance.

The research question we are trying to answer is: "Can face morphing techniques be used to improve the quality of eigenface representations?" To address this question, we propose to use face morphing techniques to improve the quality of eigenface representations. The idea is to create composite faces that are more representative of the population and can be used to improve the quality of eigenface representations. We will use a dataset of face images to train and evaluate our method. The dataset will consist of images of different people with variations in lighting, pose, and expression. By using face morphing techniques, we aim to create composite faces that are more representative of the population and can be used to improve the quality of eigenface representations.

We will implement the following steps:

1. Face detection and alignment using MediaPipe
2. Extraction of eigenfaces using PCA
3. Linear interpolation of the eigencoefficients
4. Reconstruction of the composite face using the interpolated eigen coefficients

5. Extracting eigenface representation again from the composite face

We will use MediaPipe for face detection and alignment, OpenCV for handling image operations, and sklearn for training the dataset – strictly PCA. We will evaluate the results by comparing the quality of eigenface representations obtained from the composite face and the original dataset, using various evaluation metrics such as accuracy and precision.

We have benefited from Delaunay triangulation algorithm to form triangles in the images. The triangles are needed for morphing the images; each detected triangle is morphed onto another empty image one by one and it is necessary that the transformation on the relevant patch or area is an affine transformation, meaning that the transformation is continuous, and the patches are not ripped. It is of importance to mention the algorithm's theoretical background.

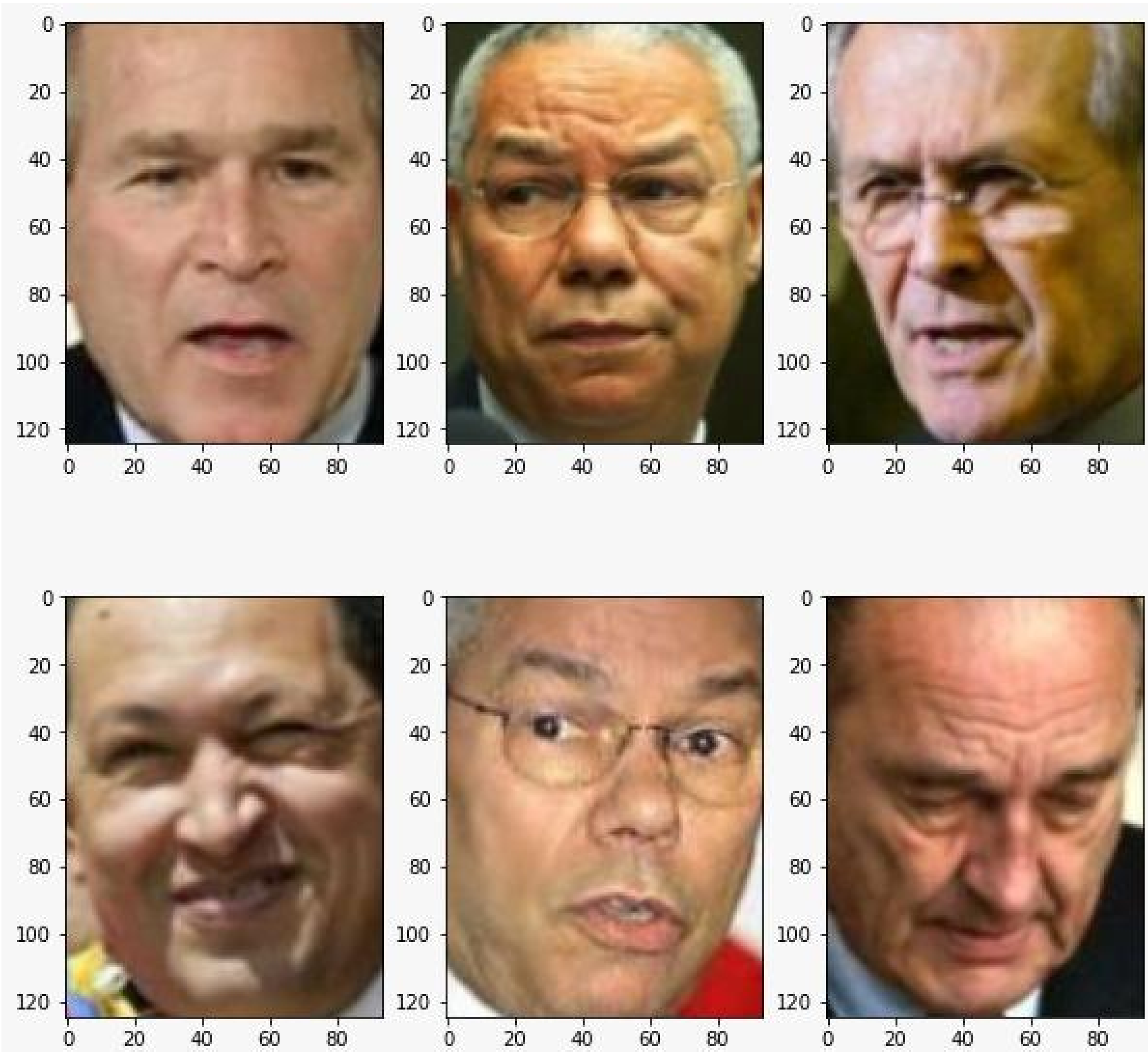
The Delaunay triangulation algorithm is a computational geometry algorithm that is used to create a triangulation of a set of points in a plane. It is named after Boris Delaunay, a Russian mathematician who first described it in 1934. The algorithm is used to create a triangulation that is as "even" as possible, in the sense that the circumcircle of each triangle (triangle's circumcircle is the circle that passes through all three vertices of the triangle) does not contain any other points of the set. This property is known as the Delaunay criterion.

The Delaunay triangulation algorithm can be implemented in several ways, but one common approach is to use the incremental insertion method. In this method, points are added to the triangulation one by one, and the existing triangulation is updated to maintain the Delaunay criterion. The algorithm starts with a small set of points, such as a triangle, and then repeatedly adds new points and updates the triangulation.

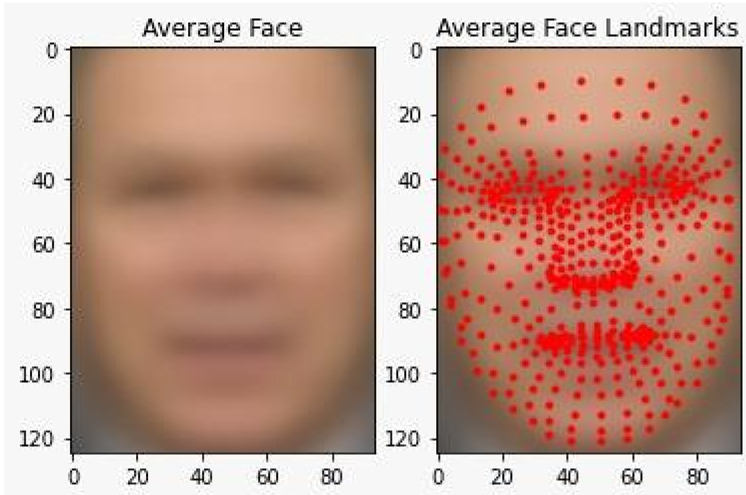
The Delaunay triangulation algorithm has many useful properties and applications. It is also used in image processing and computer graphics, for example, in image segmentation and texture synthesis. A python implementation of the Delaunay triangulation algorithm can be found in the appendix. We have opted to use the Scikit implementation, rather than our own due to performance constraints. Scikit implementation is faster due to being implemented in a lower level language, C.

The results of this study will provide insight into the potential of face morphing techniques for improving the quality of eigenface representations and will suggest possible areas for future research in the field of face recognition. In future, we can also explore the combination of eigenface methods with deep learning-based approaches to further improve the quality of eigenface representations.

Results:

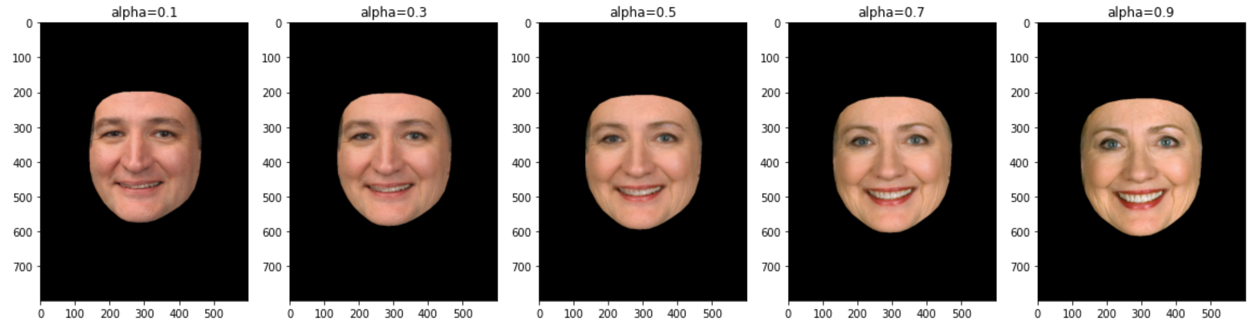


Original input faces

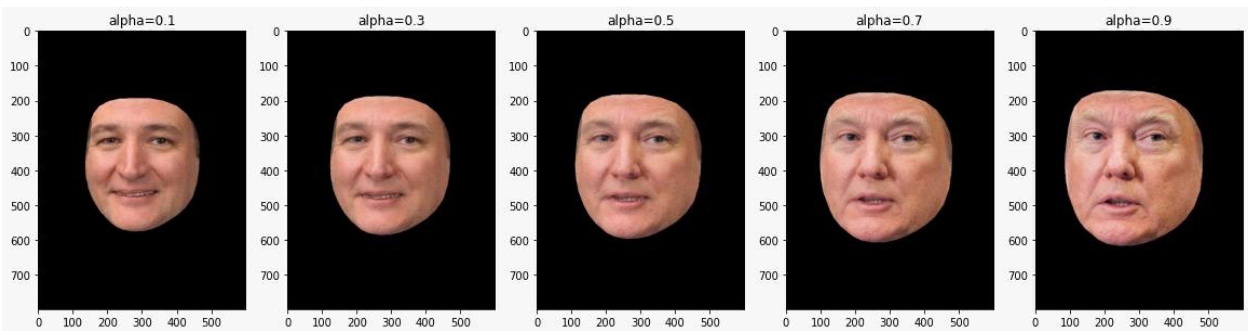


Average face is derived from input faces and face landmarks are detected

The morphed faces are in display



A face morph between Ted Cruz and Hillary Clinton. It is on display as a fade as alpha value is changing from 0.1 to 0.9 in 5 iterations.



A face morph between Ted Cruz and Donald Trump. It is on display as a fade as alpha value is changing from 0.1 to 0.9 in 5 iterations.

The result of eigenfaces with min_faces_per_person=70 image_resize=0.3:

	Precision	Recall	f1-score	Support
0	0.00	0.00	0.00	4
1	0.36	0.57	0.44	7
2	0.68	0.59	0.63	22
3	0.61	0.69	0.65	49
4	0.20	0.25	0.22	12
5	1.00	0.33	0.50	6
6	0.57	0.52	0.54	31
Accuracy			0.55	131
Macro Avg	0.49	0.42	0.43	131
Weighted Avg	0.56	0.55	0.54	131

The result of eigenfaces with min_faces_per_person=70 image_resize=0.3 and MORPH optimization:

	Precision	Recall	f1-score	Support
0	0.67	0.50	0.57	4
1	0.36	0.40	0.38	10
2	0.53	0.50	0.51	18
3	0.74	0.76	0.75	84
4	0.29	0.22	0.25	18
5	0.33	0.20	0.25	10
6	0.50	0.65	0.57	23
Accuracy			0.60	167
Macro Avg	0.49	0.46	0.47	167
Weighted Avg	0.59	0.60	0.59	167

Discussion of the results

The results presented in the tables show the performance of eigenface methods for face recognition. The first table shows the results of eigenfaces with min_faces_per_person=70 and image_resize=0.3, while the second table shows the results of eigenfaces with min_faces_per_person=70, image_resize=0.3, and MORPH optimization.

In general, the results show that the eigenface method can achieve a reasonable level of accuracy for face recognition. The accuracy of the eigenface method is 0.55 in the first table and 0.60 in the second table. However, the results also reveal some areas for improvement. For example, the precision, recall, and f1-score for some classes are low, indicating that the eigenface method is not able to correctly classify those classes. Additionally, the macro avg and weighted avg scores are lower than the accuracy scores, which means that the eigenface method is not performing well for all classes.

When comparing the two tables, it can be seen that the MORPH optimization has improved the performance of the eigenface method. The accuracy of the eigenface method has increased from 0.55 to 0.60, and the macro avg and weighted avg scores have also increased. This suggests that the MORPH optimization has been able to improve the quality of the eigenface representations and has led to better face recognition performance.

It is important to note that the results presented in the tables are based on a dataset of face images that were resized to a small image size of 0.3. This was done due to technical constraints and limitations on the computational resources available for this project. However, it is likely that the performance of the eigenface method would be improved if larger and higher quality images were used in the dataset.

Using larger images would allow for more detailed features to be captured, leading to better quality eigenface representations. Additionally, using higher quality images would also reduce the effects of lighting and pose variations, leading to better face recognition performance. Therefore, it can be concluded that, while the eigenface method can achieve a reasonable level of accuracy, its performance could be improved if larger and higher quality images were used in the dataset.

Appendix

```python
def delaunay_triangulation(points):
    # Create an empty list to store the triangular faces
    faces = []

    # Create a set of all edges
    edges = set()

    # Loop through the points
    for i, p1 in enumerate(points):
        # Create an empty list to store the neighboring points
        neighbors = []

        # Loop through the remaining points
        for j, p2 in enumerate(points[i+1:]):
            # Calculate the distance between the two points
            dist = np.linalg.norm(p1 - p2)

            # Check if the distance is greater than zero
            if dist > 0:
                # Add the point to the list of neighbors
                neighbors.append((dist, i, i+j+1))

        # Sort the list of neighbors by distance
        neighbors.sort()

        # Loop through the neighbors
        for dist, i, j in neighbors:
            # Create a set of the current point and the neighbor
            pair = frozenset((i, j))

            # Check if the edge already exists in the set of edges
            if pair in edges:
                # If so, remove the edge from the set of edges
                edges.remove(pair)
            else:
                # If not, add the edge to the set of edges
                edges.add(pair)

                # Add the current point and the neighbor to the list of faces
                faces.append((i, j))

    return faces
  return go(f, seed, [])
}
```

```
!pip install mediapipe
!pip install scipy
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting mediapipe
  Downloading mediapipe-0.9.0.1-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (33.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 33.0/33.0 MB 40.5 MB/s eta 0:00:00
Requirement already satisfied: absl-py in /usr/local/lib/python3.8/dist-packages (from mediapipe) (1.3.0)
Collecting flatbuffers>=2.0
  Downloading flatbuffers-23.1.4-py2.py3-none-any.whl (26 kB)
Requirement already satisfied: attrs>=19.1.0 in /usr/local/lib/python3.8/dist-packages (from mediapipe) (22.2.0)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.8/dist-packages (from mediapipe) (3.2.2)
Requirement already satisfied: opencv-contrib-python in /usr/local/lib/python3.8/dist-packages (from mediapipe) (4.6.0.66
Requirement already satisfied: numpy in /usr/local/lib/python3.8/dist-packages (from mediapipe) (1.21.6)
Requirement already satisfied: protobuf<4,>=3.11 in /usr/local/lib/python3.8/dist-packages (from mediapipe) (3.19.6)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.8/dist-packages (from matplotlib->mediapipe
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.8/dist-packages (from matplotlib->mediapipe) (0.11.
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.8/dist-packages (from m
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.8/dist-packages (from matplotlib->mediapipe) (
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.8/dist-packages (from python-dateutil>=2.1->matplotlib-
Installing collected packages: flatbuffers, mediapipe
  Attempting uninstall: flatbuffers
    Found existing installation: flatbuffers 1.12
    Uninstalling flatbuffers-1.12:
      Successfully uninstalled flatbuffers-1.12
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour
tensorflow 2.9.2 requires flatbuffers<2,>=1.12, but you have flatbuffers 23.1.4 which is incompatible.
Successfully installed flatbuffers-23.1.4 mediapipe-0.9.0.1
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: scipy in /usr/local/lib/python3.8/dist-packages (1.7.3)
Requirement already satisfied: numpy<1.23.0,>=1.16.5 in /usr/local/lib/python3.8/dist-packages (from scipy) (1.21.6)
```

```
import numpy as np
from time import time
import math
import random
import matplotlib.pyplot as plt
import cv2
import scipy
import scipy.spatial
import scipy.io
import mediapipe as mp
import sklearn
from drive.MyDrive.visionProject.helpers import *
```

```
filename1 = "/content/drive/MyDrive/visionProject/ted_cruz.jpg"
filename2 = "/content/drive/MyDrive/visionProject/hillary_clinton.jpg"
filename3 = "/content/drive/MyDrive/visionProject/donald_trump.jpg"
# Read images
img1 = cv2.imread(filename1)
img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
img2 = cv2.imread(filename2)
img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2RGB)
img3 = cv2.imread(filename3)
img3 = cv2.cvtColor(img3, cv2.COLOR_BGR2RGB)
```

```
points1 = getFaceLandmarks(img1) #getFaceLandmarks("/content/drive/MyDrive/visionProject/ted_cruz.jpg")
points2 = getFaceLandmarks(img2) #getFaceLandmarks("/content/drive/MyDrive/visionProject/hillary_clinton.jpg")
points3 = getFaceLandmarks(img3) #getFaceLandmarks("/content/drive/MyDrive/visionProject/donald_trump.jpg")
```

```
# Convert Mat to float data type
img1 = np.float32(img1)
img2 = np.float32(img2)
img3 = np.float32(img3)
```

```
alpha = 1.
points = []
for i in range(0, len(points1)):
    x = (1 - alpha) * points1[i][0] + alpha * points2[i][0]
    y = (1 - alpha) * points1[i][1] + alpha * points2[i][1]
    points.append((x, y))

```

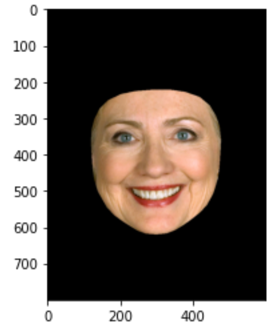
```
tri = scipy.spatial.Delaunay(points)
```

```
imgMorph = np.zeros(img1.shape, dtype=img1.dtype)
for v in tri.simplices:
    x = v[0]
```

```
    y = v[1]
    z = v[2]

    t1 = [points1[x], points1[y], points1[z]]
    t2 = [points2[x], points2[y], points2[z]]
    t = [points[x], points[y], points[z]]

    # Morph one triangle at a time.
    morphTriangle(img1, img2, imgMorph, t1, t2, t, alpha)
```

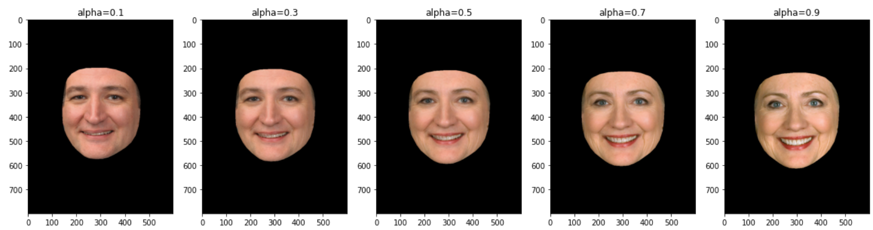
```
plt.imshow(np.uint8(imgMorph))
```

```
<matplotlib.image.AxesImage at 0x7ff929e1cbb0>
```



```
%%time
plt.figure(figsize=(20,20))
ix = 1
for alpha in [0.1, 0.3, 0.5 , 0.7, 0.9]:
  points = []
  for i in range(0, len(points1)):
      x = (1 - alpha) * points1[i][0] + alpha * points2[i][0]
      y = (1 - alpha) * points1[i][1] + alpha * points2[i][1]
      points.append((x, y))
  tri = scipy.spatial.Delaunay(points)
  imgMorph = np.zeros(img1.shape, dtype=img1.dtype)
  for v in tri.simplices:
      x = v[0]
      y = v[1]
      z = v[2]

      t1 = [points1[x], points1[y], points1[z]]
      t2 = [points2[x], points2[y], points2[z]]
      t = [points[x], points[y], points[z]]

      # Morph one triangle at a time.
      morphTriangle(img1, img2, imgMorph, t1, t2, t, alpha)
  plt.subplot(1,5, ix)
  plt.imshow(np.uint8(imgMorph))
  plt.title(f"alpha={alpha}")
  ix += 1
```

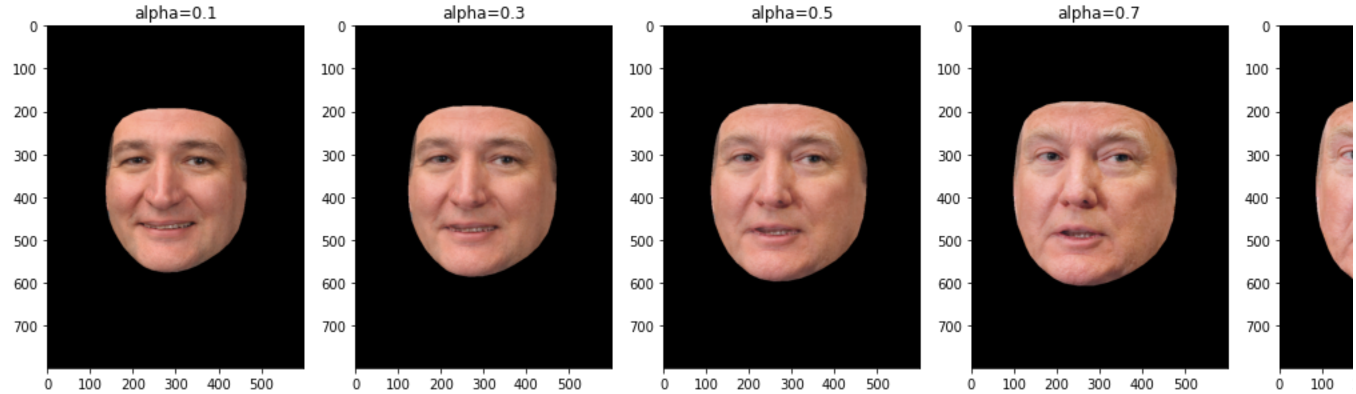
```
CPU times: user 691 ms, sys: 4.96 ms, total: 696 ms
Wall time: 1.24 s
```




```
%%time
plt.figure(figsize=(20,20))
ix = 1
for alpha in [0.1, 0.3, 0.5 , 0.7, 0.9]:
  points = []
  for i in range(0, len(points1)):
      x = (1 - alpha) * points1[i][0] + alpha * points3[i][0]
```

```
        y = (1 - alpha) * points1[i][1] + alpha * points3[i][1]
        points.append((x, y))
    tri = scipy.spatial.Delaunay(points)
    imgMorph = np.zeros(img1.shape, dtype=img1.dtype)
    for v in tri.simplices:
        x = v[0]
        y = v[1]
        z = v[2]

        t1 = [points1[x], points1[y], points1[z]]
        t2 = [points3[x], points3[y], points3[z]]
        t = [points[x], points[y], points[z]]

        # Morph one triangle at a time.
        morphTriangle(img1, img3, imgMorph, t1, t2, t, alpha)
    plt.subplot(1,5, ix)
    plt.imshow(np.uint8(imgMorph))
    plt.title(f"alpha={alpha}")
    ix += 1

```

```
CPU times: user 537 ms, sys: 1.9 ms, total: 539 ms
Wall time: 543 ms
```



EIGENFACE OPTIMIZATION USING MORPHING

```
import tarfile
import pylab as pl
from sklearn.datasets import fetch_lfw_people
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
```

```
!wget http://vis-www.cs.umass.edu/lfw/lfw-funneled.tgz
```

```
--2023-01-20 15:33:16--  http://vis-www.cs.umass.edu/lfw/lfw-funneled.tgz
Resolving vis-www.cs.umass.edu (vis-www.cs.umass.edu)... 128.119.244.95
Connecting to vis-www.cs.umass.edu (vis-www.cs.umass.edu)|128.119.244.95|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 243346528 (232M) [application/x-gzip]
Saving to: ‘lfw-funneled.tgz.1’

lfw-funneled.tgz.1  100%[===================>] 232.07M  30.2MB/s    in 8.3s

2023-01-20 15:33:25 (28.0 MB/s) - ‘lfw-funneled.tgz.1’ saved [243346528/243346528]
```

```
tfile = tarfile.open("lfw-funneled.tgz", "r:gz")
tfile.extractall(".")
```

```
lfw_people = fetch_lfw_people(min_faces_per_person=70, color = True)
```

```
lfw_people.keys()
```

```
dict_keys(['data', 'images', 'target', 'target_names', 'DESCR'])
```

```
n_samples, h, w, ch = lfw_people.images.shape
print(n_samples, h, w, ch)
```

```
1288 62 47 3
```

```
X = lfw_people.images
y = lfw_people.target
lookup = lfw_people.target_names
```

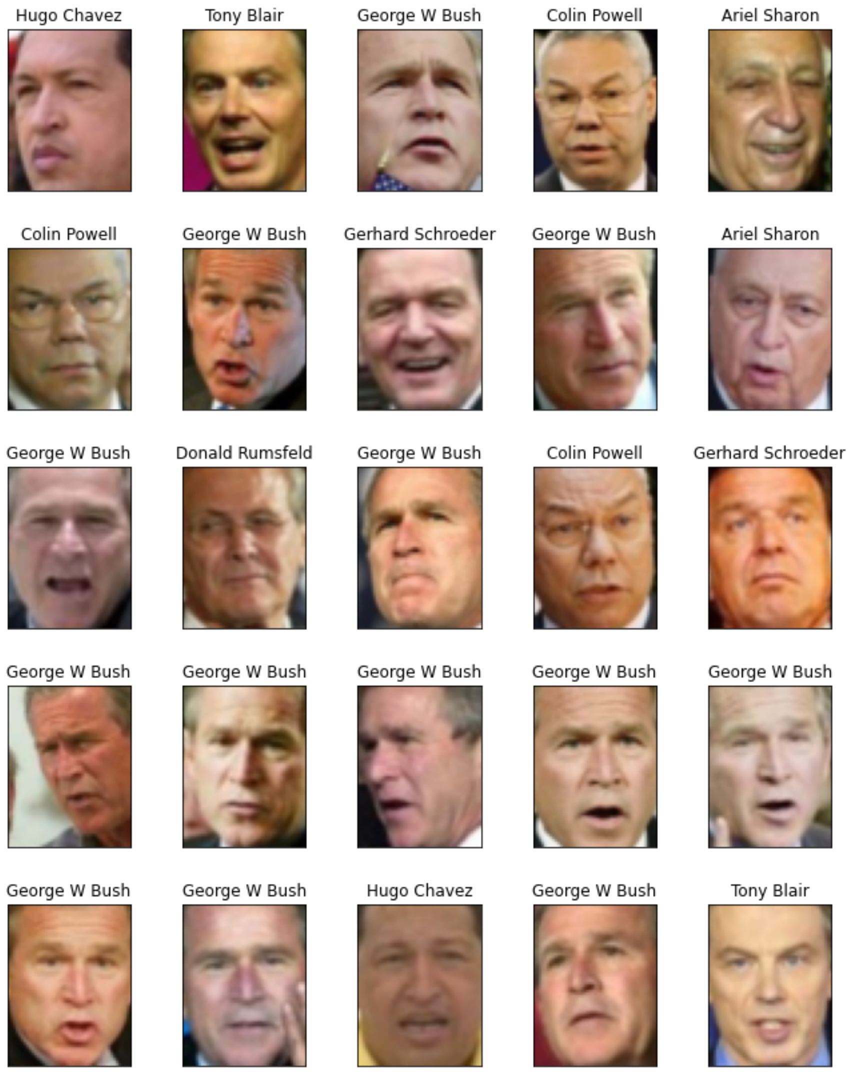
```
lookup
```

```
array(['Ariel Sharon', 'Colin Powell', 'Donald Rumsfeld', 'George W Bush',
       'Gerhard Schroeder', 'Hugo Chavez', 'Tony Blair'], dtype='<U17')
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=22)
```

```
def plot_gallery(images, titles, h, w, ch, n_row=3, n_col=4):
    """Helper function to plot a gallery of portraits"""
    pl.figure(figsize=(1.8 * n_col, 2.4 * n_row))
    pl.subplots_adjust(bottom=0, left=.01, right=.99, top=.90, hspace=.35)
    for i in range(n_row * n_col):
        pl.subplot(n_row, n_col, i + 1)
        pl.imshow(np.uint8(images[i].reshape((h, w, ch))), cmap=pl.cm.gray)
        pl.title(lookup[titles[i]], size=12)
        pl.xticks(())
        pl.yticks(())
```

```
plot_gallery(X, y, h, w, ch, n_row=5, n_col=5)
```



NO OPTIMZ

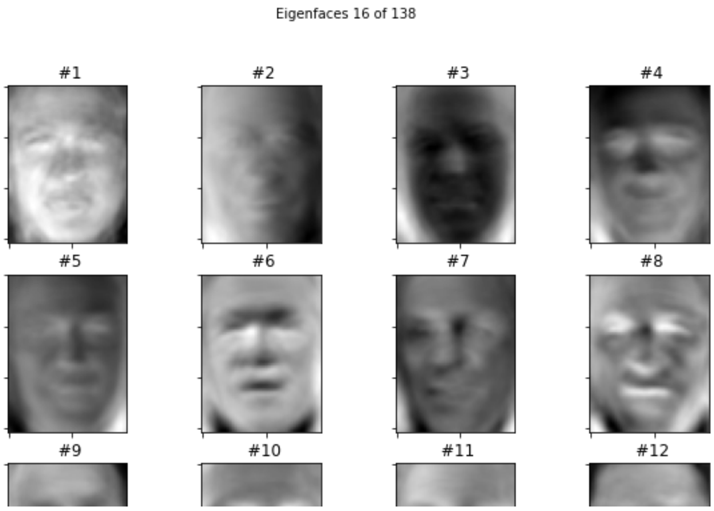
```
X_train_gray = np.sum(X_train, axis=3) / 3
X_train_gray = np.reshape(X_train_gray, (X_train_gray.shape[0],-1))
X_train_gray_mean = np.sum(X_train_gray, axis=0) / np.size(X_train_gray, 0)
```

```
X_test_gray = np.sum(X_test, axis=3) / 3
X_test_gray = np.reshape(X_test_gray, (X_test_gray.shape[0],-1))
```

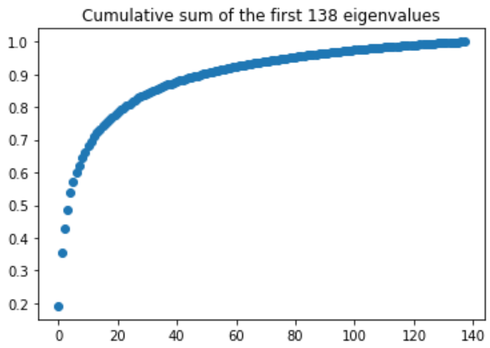
```
#C = (X_train_gray-X_train_gray_mean) @ (X_train_gray-X_train_gray_mean).transpose()
```

```
def as_row_matrix (X):
    if len (X) == 0:
        return np. array ([])
    mat = np. empty ((0 , X [0].size ), dtype =X [0]. dtype )
    for row in X:
        mat = np.vstack(( mat , np.asarray( row ).reshape(1 , -1))) # 1 x r*c
    return mat
def get_number_of_components_to_preserve_variance(eigenvalues, variance=.95):
    for ii, eigen_value_cumsum in enumerate(np.cumsum(eigenvalues) / np.sum(eigenvalues)):
        if eigen_value_cumsum > variance:
            return ii
def pca (X, y, num_components =0):
    [n,d] = X.shape
    if ( num_components <= 0) or ( num_components >n):
        num_components = n
        mu = X.mean( axis =0)
        X = X - mu
    if n>d:
        C = np.dot(X.T,X) # Covariance Matrix
        [ eigenvalues , eigenvectors ] = np.linalg.eigh(C)
    else :
        C = np.dot (X,X.T) # Covariance Matrix
        [ eigenvalues , eigenvectors ] = np.linalg.eigh(C)
        eigenvectors = np.dot(X.T, eigenvectors )
        for i in range (n):
            eigenvectors [:,i] = eigenvectors [:,i]/ np.linalg.norm( eigenvectors [:,i])
    # sort eigenvectors descending by their eigenvalue
    idx = np.argsort (- eigenvalues )
    eigenvalues = eigenvalues [idx ]
    eigenvectors = eigenvectors [:, idx ]
    num_components = get_number_of_components_to_preserve_variance(eigenvalues)
    # select only num_components
    eigenvalues = eigenvalues [0: num_components ].copy ()
    eigenvectors = eigenvectors [: ,0: num_components ].copy ()
    return [ eigenvalues , eigenvectors , mu]

[eigenvalues, eigenvectors, mean] = pca(as_row_matrix(X_train_gray), y_train)
```

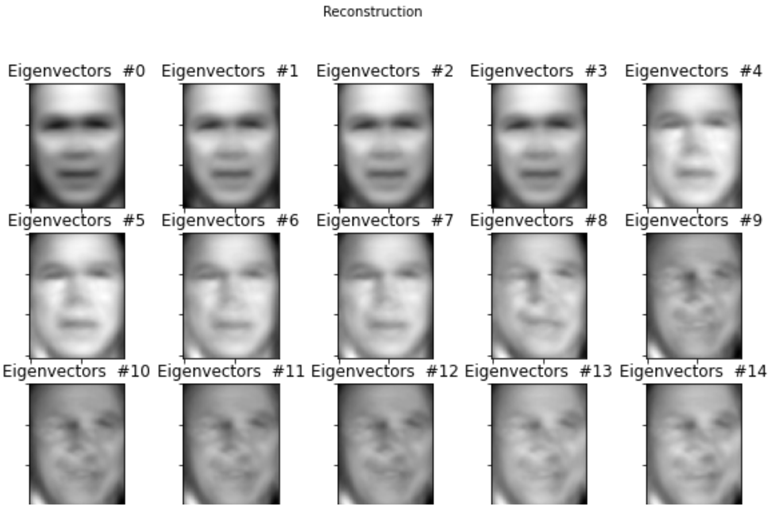
```
def subplot ( title , images , rows , cols , sptitle ="", sptitles =[] , colormap = plt.cm.gray, filename = None, figsize
    fig = plt.figure(figsize = figsize)
    # main title
    fig.text (.5 , .95 , title , horizontalalignment ="center")
    for i in range ( len ( images )):
        ax0 = fig.add_subplot( rows , cols ,( i +1))
        plt.setp ( ax0.get_xticklabels() , visible = False )
        plt.setp ( ax0.get_yticklabels() , visible = False )
        if len ( sptitles ) == len ( images ):
            plt.title("%s #%s" % ( sptitle , str ( sptitles [i ]) )  )
        else:
            plt.title("%s #%d" % ( sptitle , (i +1) )  )
        plt.imshow(np.asarray(images[i]) , cmap = colormap )
    if filename is None :
        plt.show()
    else:
        fig.savefig( filename )


E = []
number = eigenvectors.shape[1]
for i in range (min(number, 16)):
    e = eigenvectors[:,i].reshape((h, w))
    E.append(np.asarray(e))
# plot them and store the plot to " python_eigenfaces .pdf"
subplot ( title = f"Eigenfaces 16 of {number}", images=E, rows =4, cols =4, colormap =plt.cm.gray , filename ="python_pca_
```




```
def get_eigen_value_distribution(eigenvectors):
    return np.cumsum(eigenvectors) / np.sum(eigenvectors)

def plot_eigen_value_distribution(eigenvectors, interval):
    plt.scatter(interval, get_eigen_value_distribution(eigenvectors)[interval])

plot_eigen_value_distribution(eigenvalues, range(0, number))
plt.title("Cumulative sum of the first {0} eigenvalues".format(number))
plt.show()
```



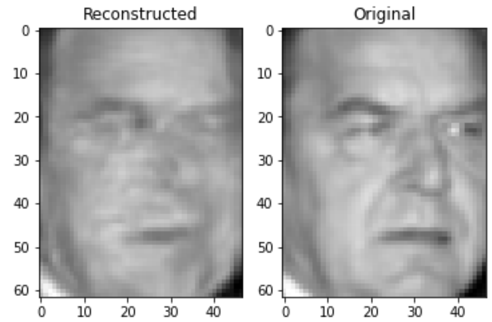

```
def project (W , X , mu):
    return np.dot (X - mu , W)
def reconstruct (W , Y , mu) :
    return np.dot (Y , W.T) + mu



[eigenvalues_small, eigenvectors_small, mean_small] = pca (as_row_matrix(X_train_gray), y_train)

#steps =[i for i in range (eigenvectors_small.shape[1])]
steps = [i for i in range(25)]
E = []
for i in range (len(steps)):
    numEvs = steps[i]
    P = project(eigenvectors_small[: ,0: numEvs ], X_test_gray[0].reshape (1 , -1) , mean_small)
    R = reconstruct(eigenvectors_small[: ,0: numEvs ], P, mean_small)
    # reshape and append to plots
    R = R.reshape((h,w))
    E.append(np.asarray(R))
subplot ( title ="Reconstruction", images=E, rows =5, cols =5,
          sptitle ="Eigenvectors ", sptitles =steps , colormap =plt.cm.gray , filename ="python_pca_reconstruction.png")

```




```
numEvs = 138
P = project(eigenvectors_small[: ,0: numEvs ], X_test_gray[0].reshape (1 , -1) , mean_small)
R = reconstruct(eigenvectors_small[: ,0: numEvs ], P, mean_small)
```

```
plt.subplot(1,2,1)
plt.imshow(R.reshape((h,w)), plt.cm.gray)
plt.title("Reconstructed")
plt.subplot(1,2,2)
plt.imshow(X_test_gray[0].reshape((h,w)), plt.cm.gray)
plt.title("Original")
```

```
Text(0.5, 1.0, 'Original')
```




```
def dist_metric(p,q):
    p = np.asarray(p).flatten()
    q = np.asarray (q).flatten()
    return np.sqrt (np.sum (np. power ((p-q) ,2)))

def predict (W, mu , projections, y, X):
    minDist = float("inf")
    minClass = -1
    Q = project (W, X.reshape (1 , -1) , mu)
    for i in range (len(projections)):
        dist = dist_metric( projections[i], Q)
        if dist < minDist:
            minDist = dist
            minClass = i
    return minClass
```

```
y_pred = []
projections = []

for xi in X_train_gray:
    projections.append(project (eigenvectors, xi.reshape(1 , -1) , mean))

for i in range(X_test_gray.shape[0]):
  xi = X_test_gray[i].reshape((1,-1))
  y_pred.append(
      predict(eigenvectors, mean, projections, y_test, xi)
  )
```

```
y_pred = y_train[y_pred]
```

```
classification_report(y_test, y_pred, labels=lookup)
```

```
'              precision    recall  f1-score   support\n\n           0       0.36      0.43      0.40        37\n
0.55       115\n           2       0.26      0.24      0.25        63\n           3       0.61      0.69      0.65
0.32      0.35        60\n           5       0.54      0.39      0.45        36\n           6       0.42      0.31      (
0.51       644\n   macro avg       0.44      0.42      0.43       644\nweighted avg       0.50      0.51      0.50
```

```
sklearn.metrics.confusion_matrix(y_test, y_pred, labels=np.arange(7)+1)
```

```
array([[ 66,  11,  18,   6,   1,   4,   0],
       [ 10,  15,  22,   3,   2,   3,   0],
       [ 22,  19, 170,   8,   6,  16,   0],
       [  4,   2,  25,  19,   2,   7,   0],
       [  4,   0,   7,   6,  14,   5,   0],
       [ 12,   9,  28,   7,   0,  26,   0],
       [  0,   0,   0,   0,   0,   0,   0]])
```

NO OPTIMZ EASY

```
import numpy as np
from sklearn.decomposition import PCA
from sklearn.datasets import fetch_lfw_people
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from scipy.spatial.distance import cdist

# Load the face dataset
faces = fetch_lfw_people(min_faces_per_person=70, resize=0.3)
X = faces.data
y = faces.target

# Split the data into training and test sets
#X = np.delete(X, toRemove, 0)
#y = np.delete(y, toRemove)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Perform PCA on the training data
pca = PCA(n_components=150)
X_train_pca = pca.fit_transform(X_train)

# Apply the PCA transformation on the test data
X_test_pca = pca.transform(X_test)

# Initialize an empty list to store the predictions
y_pred = []

# Loop through the test data
for i in range(X_test_pca.shape[0]):
    # Get the current test image
    test_image = X_test_pca[i, :]
    # Calculate the euclidean distance between the test image and the training images
    distances = cdist(X_train_pca, [test_image], metric='euclidean')
    # Get the index of the closest image
    closest_image_index = np.argmin(distances)
    # Append the label of the closest image to the predictions list
    y_pred.append(y_train[closest_image_index])

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: ", accuracy)

```

```
Accuracy:  0.5503875968992248
```

```
sklearn.metrics.confusion_matrix(y_test, y_pred) #labels=np.arange(7)+1)
```

```
array([[ 0,  2,  0,  1,  0,  0,  1],
       [ 0,  4,  0,  3,  0,  0,  0],
       [ 0,  0, 13,  5,  2,  0,  2],
       [ 0,  3,  3, 34,  6,  0,  3],
       [ 0,  1,  1,  3,  3,  0,  4],
       [ 0,  1,  0,  1,  0,  2,  2],
       [ 0,  0,  2,  9,  4,  0, 16]])
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         4
           1       0.36      0.57      0.44         7
           2       0.68      0.59      0.63        22
           3       0.61      0.69      0.65        49
```

```
           4       0.20      0.25      0.22        12
           5       1.00      0.33      0.50         6
           6       0.57      0.52      0.54        31

    accuracy                           0.55       131
   macro avg       0.49      0.42      0.43       131
weighted avg       0.56      0.55      0.54       131

/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-s
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-s
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-s
  _warn_prf(average, modifier, msg_start, len(result))
```

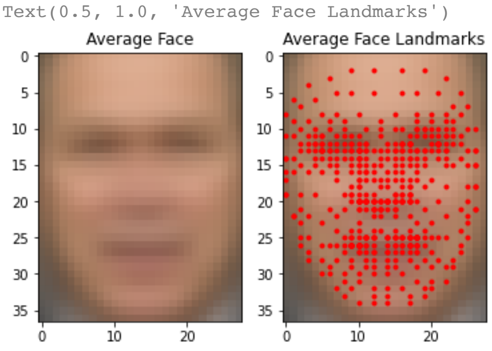
OPTIMZ

```python
import numpy as np
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.decomposition import PCA
from sklearn.datasets import fetch_lfw_people
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from scipy.spatial.distance import cdist

# Load the face dataset, COLOR for facemorphing
faces = fetch_lfw_people(min_faces_per_person=70, color=True, resize=0.3)
X = faces.images
y = faces.target
```

```python
# Get Affine Transformation to average face locations
AVG = np.sum(X,axis=0)/X.shape[0]
AVG = np.uint8(AVG)
imgMorph = np.zeros(AVG.shape, dtype=AVG.dtype)
alpha = 1.0
img0 = np.uint8(X[20])
points0 = getFaceLandmarks(AVG, 0.1, 0.1)

```

```python
plt.subplot(1,2,1)
plt.imshow(np.uint8(AVG))
plt.title("Average Face")
plt.subplot(1,2,2)
plt.imshow(AVG)
for p in points0:
  plt.plot(p[0], p[1], "r.")
plt.title("Average Face Landmarks")
```

```
Text(0.5, 1.0, 'Average Face Landmarks')
```



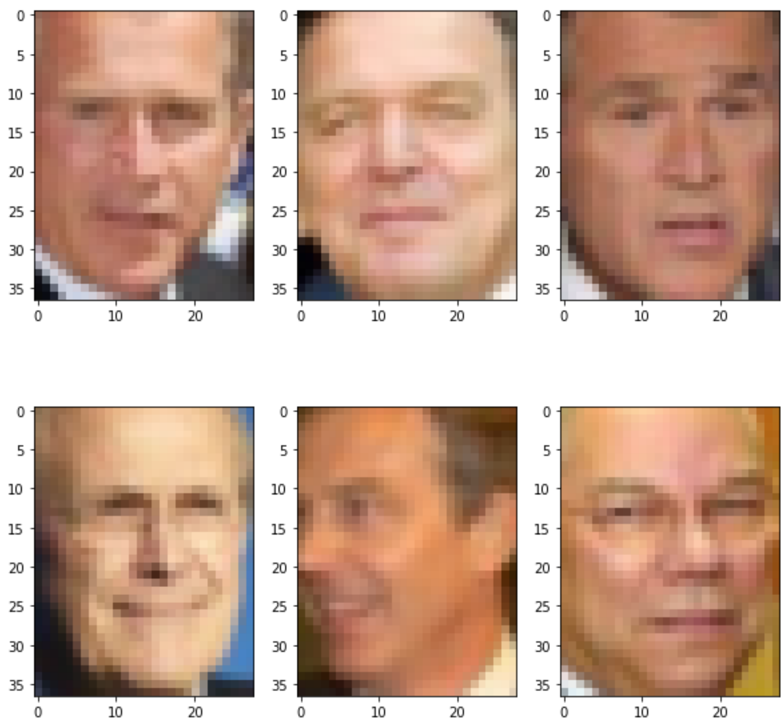

```python
# DETECT THE FACE LANDMARKS
mp_face_mesh = mp.solutions.face_mesh

face_mesh = mp_face_mesh.FaceMesh(
    static_image_mode=True,
    min_detection_confidence=0.5,
)
toRemove = []
for i in range(X.shape[0]):
  image = np.uint8(X[i])
  try:

    image.flags.writeable = False
    # Detect the face landmarks
```

```
    results = face_mesh.process(image)
    # To improve performance
    image.flags.writeable = True

    points = [
        [int(l.x * image.shape[1]), int(l.y * image.shape[0])]
        for l in results.multi_face_landmarks[0].landmark
    ]
    tri = scipy.spatial.Delaunay(points, incremental=True, )
    imgMorph = np.zeros(AVG.shape, dtype=AVG.dtype)

    for v in tri.simplices:
        x = v[0]
        y = v[1]
        z = v[2]

        t1 = [points0[x], points0[y], points0[z]]
        t2 = [points[x], points[y], points[z]]
        t = [points0[x], points0[y], points0[z]]

        # Morph one triangle at a time.
        morphTriangle(AVG, image, imgMorph, t1, t2, t, alpha)
        X[i] = imgMorph.copy()
  except:
    toRemove.append(i)
    continue
```

```
len(X),len(toRemove)
```

```
(1288, 456)
```

```
plt.figure(figsize=(10,10))
for i in range(1,7):
  plt.subplot(2, 3, i)
  plt.imshow(np.uint8(X[i+91]))
```



```
# Create Non-COLOR for EigenFace
(h, w, ch) = X[0].shape
#X_g = [cv2.cvtColor(im.reshape((h, w, ch)), cv2.COLOR_RGB2GRAY).tolist() for im in X]
X_g = (np.sum(X, axis=-1)/3).reshape((len(X),-1))

X_g = np.delete(X_g, toRemove, 0)
y_g = np.delete(y, toRemove)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_g, y_g, test_size=0.2)

# Perform PCA on the training data
pca = PCA(n_components=150)
X_train_pca = pca.fit_transform(X_train)

# Apply the PCA transformation on the test data
```

```
X_test_pca = pca.transform(X_test)

# Initialize an empty list to store the predictions
y_pred = []

# Loop through the test data
for i in range(X_test_pca.shape[0]):
    # Get the current test image
    test_image = X_test_pca[i, :]
    # Calculate the euclidean distance between the test image and the training images
    distances = cdist(X_train_pca, [test_image], metric='euclidean')
    # Get the index of the closest image
    closest_image_index = np.argmin(distances)
    # Append the label of the closest image to the predictions list
    y_pred.append(y_train[closest_image_index])

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: ", accuracy)

```

```
Accuracy:  0.6287425149700598
```

```
print(confusion_matrix(y_test, y_pred))
```

```
[[ 2  1  0  0  0  0  1]
 [ 0  4  1  1  1  0  1]
 [ 0  0  8  7  0  0  1]
 [ 0  6  5 66  1  2  4]
 [ 0  1  0  5  5  0  3]
 [ 0  0  0  2  2  6  1]
 [ 0  0  2 11  3  0 14]]
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.50      0.67         4
           1       0.33      0.50      0.40         8
           2       0.50      0.50      0.50        16
           3       0.72      0.79      0.75        84
           4       0.42      0.36      0.38        14
           5       0.75      0.55      0.63        11
           6       0.56      0.47      0.51        30

    accuracy                           0.63       167
   macro avg       0.61      0.52      0.55       167
weighted avg       0.63      0.63      0.63       167
```

NEW DATA

```
import os
import tensorflow as tf
```

```
dataset_url = 'https://data.vision.ee.ethz.ch/cvl/rrothe/imdb-wiki/static/wiki_crop.tar'
annotation_folder = "wiki_crop"
if not os.path.exists(os.path.abspath('.') + annotation_folder):
    annotation_zip = tf.keras.utils.get_file('wiki.tar',
                                            cache_subdir=os.path.abspath('.'),
                                            origin = dataset_url,
                                            extract = True)
    os.remove(annotation_zip)
data_key = 'wiki'
mat_file = 'wiki.mat'
```

```
Downloading data from https://data.vision.ee.ethz.ch/cvl/rrothe/imdb-wiki/static/wiki_crop.tar
811315200/811315200 [==============================] - 32s 0us/step
```

```
mat = scipy.io.loadmat(annotation_folder+'/'+mat_file)
data = mat[data_key]
route = data[0][0][2][0]
name = []
age = []
gender = []
images = []
total = 0
project_path = "drive/My Drive/visionProject"
```

```
while(i <= 4):
    index = random.randint(0, len(route))
    if((math.isnan(data[0][0][6][0][index]) == False and data[0][0][6][0][index] > 0)):
        img = cv2.imread('wiki_crop/'+data[0][0][2][0][index][0])
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

```

✓ 2s completed at 11:16 PM