ECON 425 HW9

March 5, 2024

Problem 2 (K-means clustering)

In this problem, you will perform K-means clustering manually, with $K = 2$, on a small example with $n = 6$ observations and $p = 2$ features. The data matrix is

$$X = \begin{pmatrix} 1 & 4 \\ 1 & 3 \\ 0 & 4 \\ 5 & 1 \\ 6 & 2 \\ 4 & 0 \end{pmatrix}$$

(a) Plot the observations.

(b) Randomly assign a cluster label to each observation. You can use the np.random.choice() function to do this. Report the cluster labels for each observation.

(c) Compute the centroid for each cluster.

(d) Assign each observation to the centroid to which it is closest, in terms of Euclidean distance. Report the cluster labels for each observation.

(e) Repeat (c) and (d) until the answers obtained stop changing.

(f) In your plot from (a), color the observations according to the cluster labels obtained.

Problem 3 (PCA on Olivetti faces)

Use the Olivetti faces dataset available through sklearn to do the following.

(a) Fetch and load the data with the fetch_olivetti_faces method from sklearn.datasets.

(b) Demean each face in the data set (no need to divide by standard deviation as every dimension is a number between a fixed range representing a pixel).

(c) Compute and display the first 9 *eigenfaces*. The k-th eigenface of a given face is an image based on the first k principal components only.

(d) Any given face in the data set can be represented as a linear combination of the eigenfaces. For any face in the data set, show how it progresses as we combine $1, 51, 101, \ldots$ eigenfaces, until the full image is recovered.

ML HW 9

March 19, 2024

[40]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

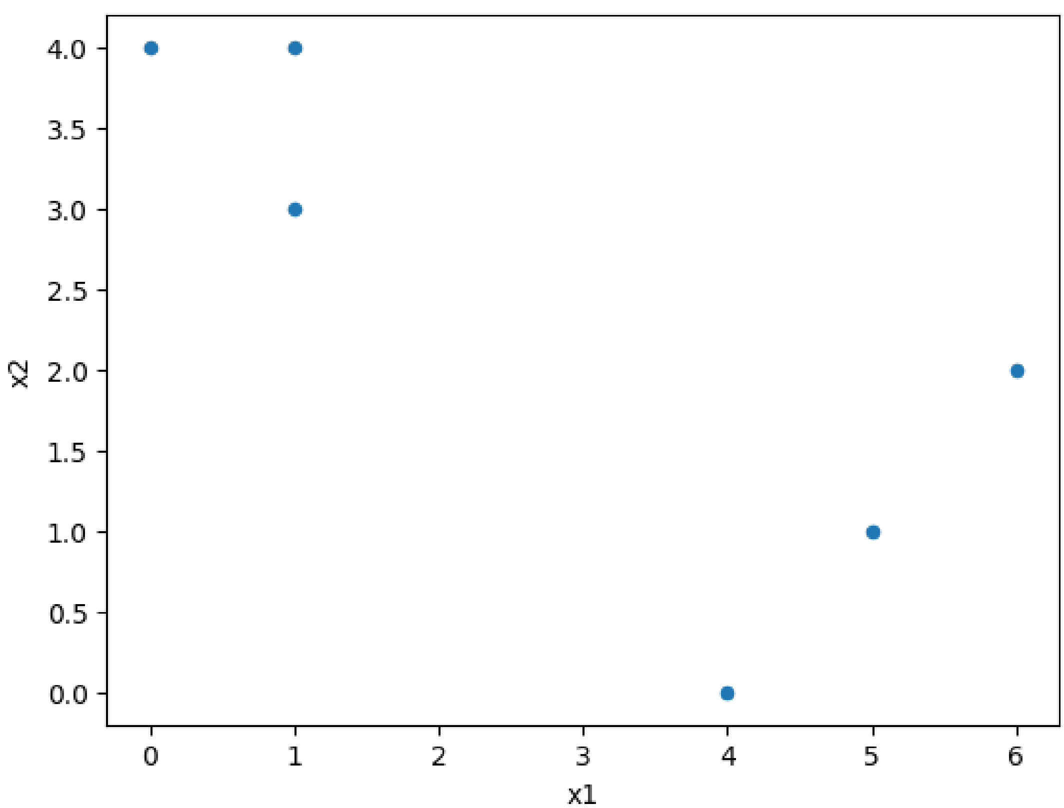
0.1 Problem 2

[41]:
```
df = pd.DataFrame({"x1": [1,1,0,5,6,4], "x2": [4,3,4,1,2,0]})
```

0.1.1 a)

Below I plot the observations from the dataframe.

[42]:
```
df.plot(x= "x1", y="x2", kind='scatter')
plt.show()
```



0.1.2 b)

[46]:
```
for i in df.index:
    df.loc[i, 'cluster'] = np.random.choice([1,2])
    print("Observation " + str(i+1) +" is assigned to cluster " + str(int(df.
 ↪loc[i,'cluster'])))
```

```
Observation 1 is assigned to cluster 2
Observation 2 is assigned to cluster 2
Observation 3 is assigned to cluster 1
Observation 4 is assigned to cluster 1
Observation 5 is assigned to cluster 2
Observation 6 is assigned to cluster 1
```

0.1.3 c)

[47]:
```
c1x1= df[df.cluster==1]["x1"].mean()
c1x2= df[df.cluster==1]["x2"].mean()
c2x1= df[df.cluster==2]["x1"].mean()
c2x2= df[df.cluster==2]["x2"].mean()


print(f"The x-coordinate for centroid 1 is {c1x1} and the y-coordinate for␣
 ↪centroid 1 is {c1x2}")
print(f"The x-coordinate for centroid 2 is {c2x1} and the y-coordinate for␣
 ↪centroid 2 is {c2x2}")
```

```
The x-coordinate for centroid 1 is 3.0 and the y-coordinate for centroid 1 is
1.6666666666666667
The x-coordinate for centroid 2 is 2.6666666666666665 and the y-coordinate for
centroid 2 is 3.0
```

0.1.4 d)

Below I assign each observation to the centroid to which it is closest, in terms of Euclidean distance.

[48]:
```
for i in df.index:
    dc1= np.sqrt((df.loc[i,"x1"]-c1x1)**2 + (df.loc[i,"x2"]-c1x2)**2)
    dc2= np.sqrt((df.loc[i,"x1"]-c2x1)**2 + (df.loc[i,"x2"]-c2x2)**2)

    if dc1<dc2:
        df.loc[i,"cluster"] =1
        print(f"Observation {i+1} is now in cluster 1")
    else:
        df.loc[i,"cluster"] =2
        print(f"Observation {i+1} is now in cluster 2")
```

```
Observation 1 is now in cluster 2
Observation 2 is now in cluster 2
Observation 3 is now in cluster 2
Observation 4 is now in cluster 1
Observation 5 is now in cluster 1
Observation 6 is now in cluster 1
```

0.1.5 e)

Below I repeat c) and d) until the answers stop changing. We can observe that after one iteration in the while loop, the cluster assignments are still the same as in d).

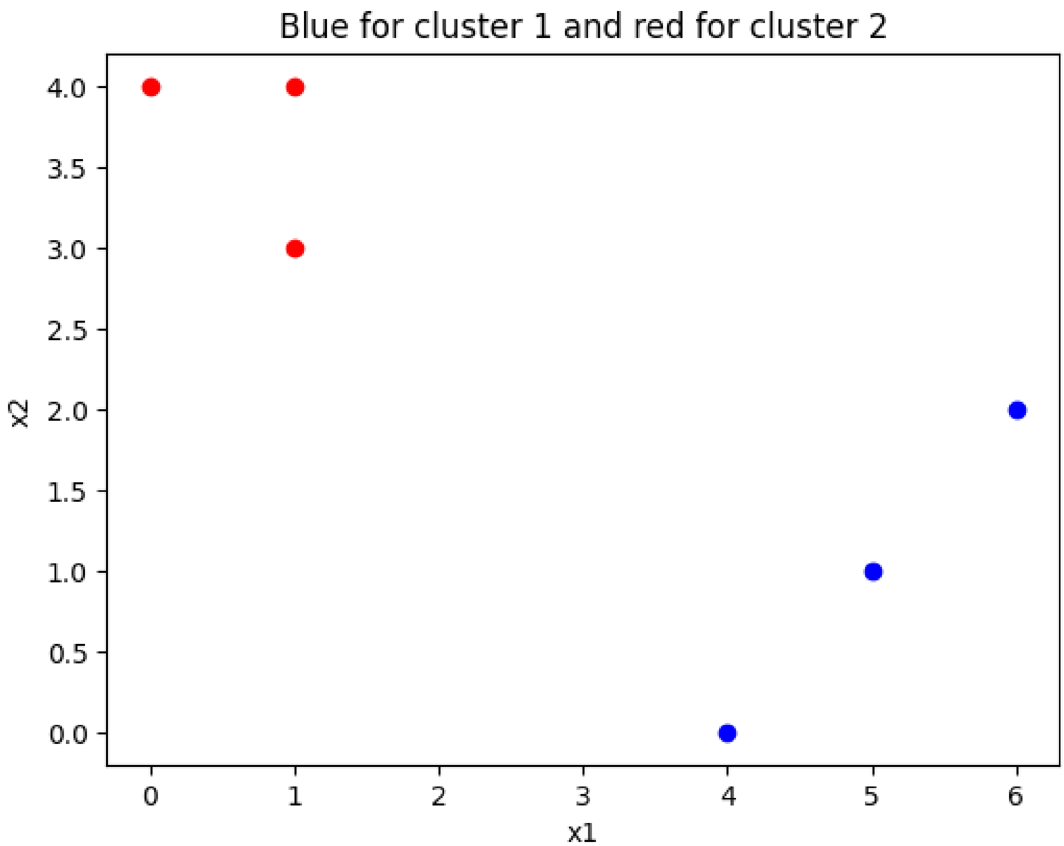
[49]:
```
check =[0,0,0,0,0,0]

while (check != df.cluster).all():

    check = df['cluster'].copy()

    c1x1= df[df.cluster==1]["x1"].mean()
    c1x2= df[df.cluster==1]["x2"].mean()
    c2x1= df[df.cluster==2]["x1"].mean()
    c2x2= df[df.cluster==2]["x2"].mean()


    for i in df.index:
        dc1= np.sqrt((df.loc[i,"x1"]-c1x1)**2 + (df.loc[i,"x2"]-c1x2)**2)
        dc2= np.sqrt((df.loc[i,"x1"]-c2x1)**2 + (df.loc[i,"x2"]-c2x2)**2)

        if dc1<dc2:
            df.loc[i,"cluster"] =1
            print(f"Observation {i+1} is now in cluster 1")
        else:
            df.loc[i,"cluster"] =2
            print(f"Observation {i+1} is now in cluster 2")
```

```
Observation 1 is now in cluster 2
Observation 2 is now in cluster 2
Observation 3 is now in cluster 2
Observation 4 is now in cluster 1
Observation 5 is now in cluster 1
Observation 6 is now in cluster 1
```

0.1.6 f)

[53]:
```
c1 =df[df.cluster==1]
c2 =df[df.cluster==2]
plt.scatter(c1.x1, c1.x2, color='blue')
plt.scatter(c2.x1,c2.x2, color='red')
```

```
plt.xlabel("x1")
plt.ylabel("x2")
plt.title("Blue for cluster 1 and red for cluster 2")
plt.show()
```



0.2 Probelm 3

0.2.1 a)

Below I fetch and load the data with the fetch_olivetti_faces method from sklearn.datasets

[66]:
```
from sklearn.datasets import fetch_olivetti_faces

faces = fetch_olivetti_faces()

df = pd.DataFrame(faces.data)
```

0.2.2 b)

I then demean each face in the dataset.

[67]:
```
demean = df.mean(axis=0)
```

[68]:
```
df = df.sub(demean, axis = 1)
```

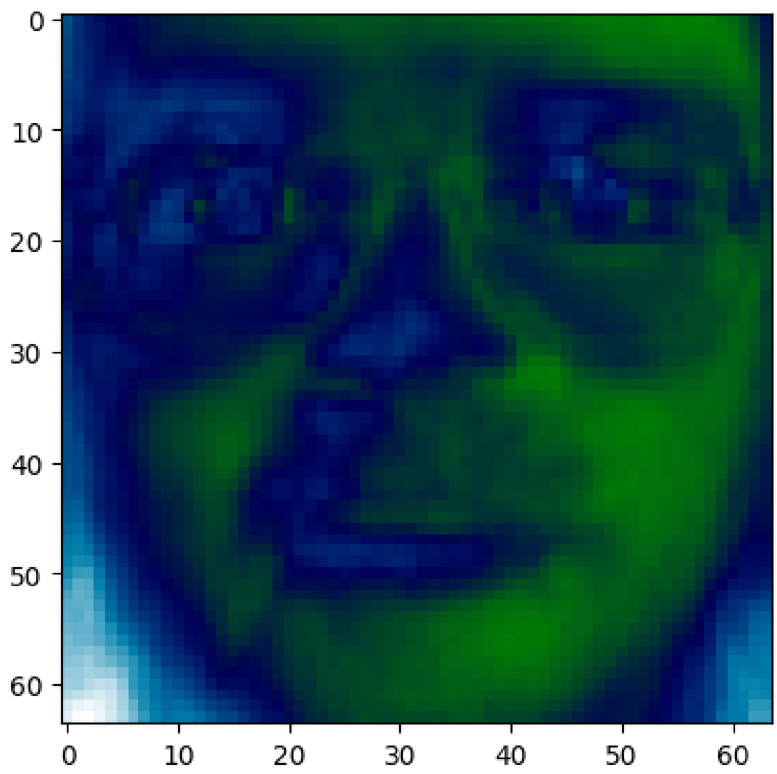
0.2.3 c)

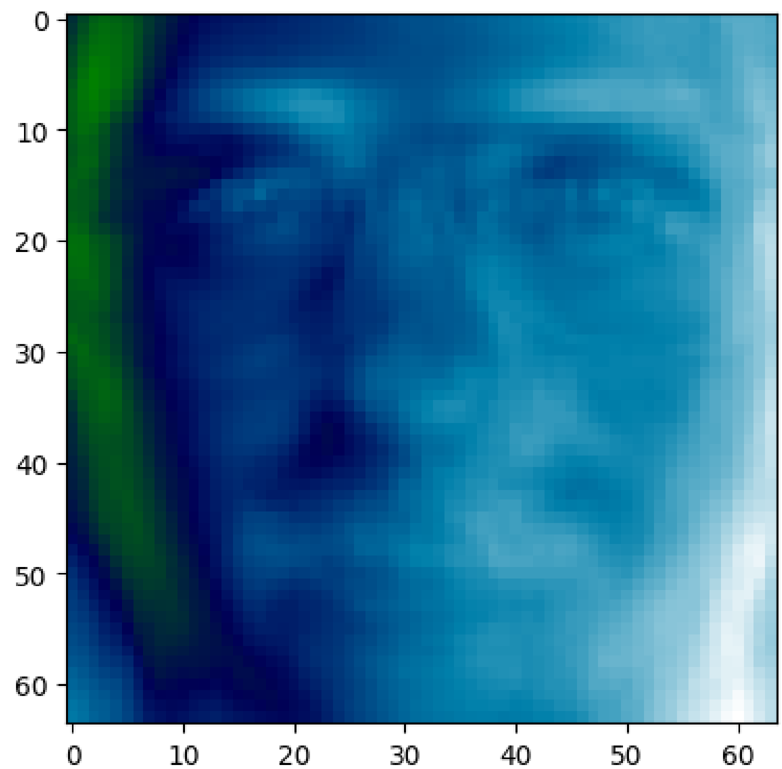
Below I will compute 9 eigenfaces using PCA.

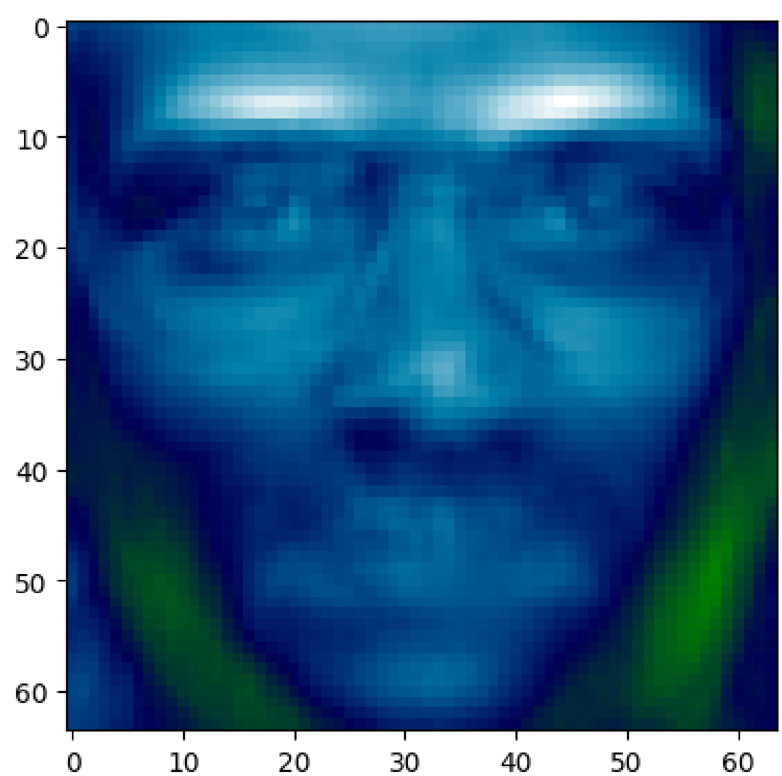
[85]:
```
from sklearn.decomposition import PCA


ef = 9
pca = PCA(n_components=ef)
pca.fit(df)
```

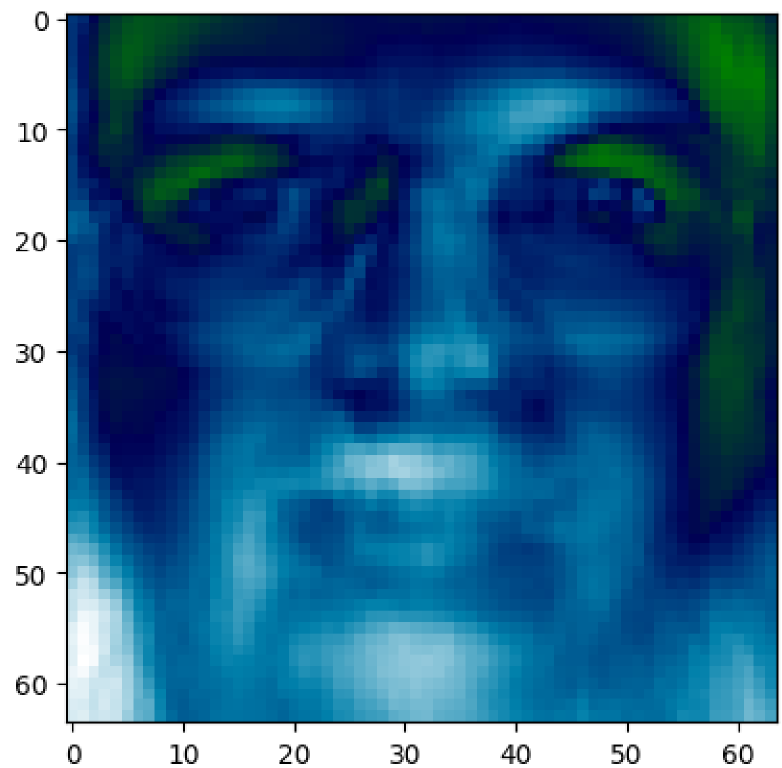
[85]:
```
PCA(n_components=9)
```

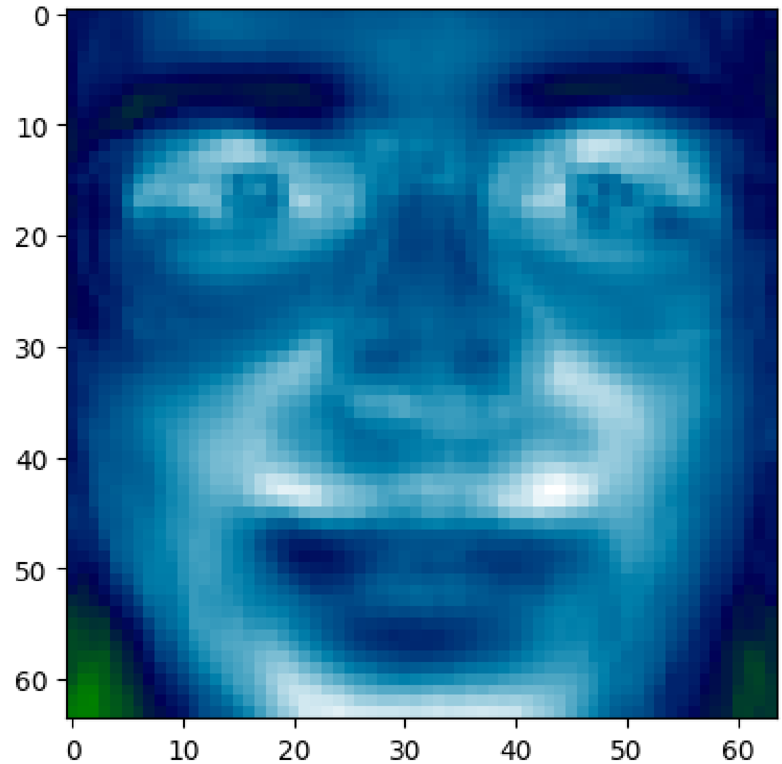
[127]:
```
for i in range(9):
    plt.imshow(pca.components_[i].reshape(64, 64), cmap='ocean')
    plt.show()
```

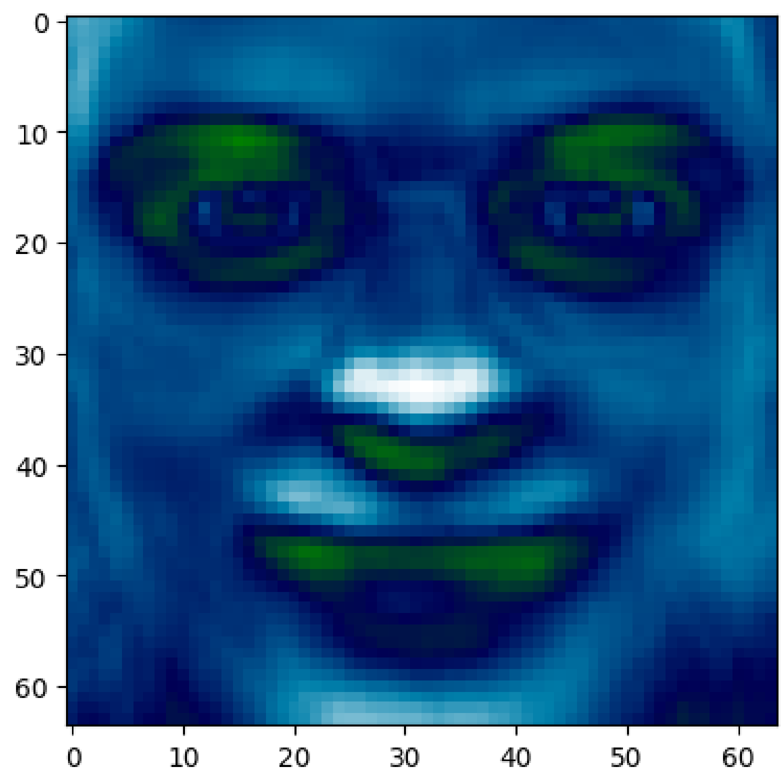


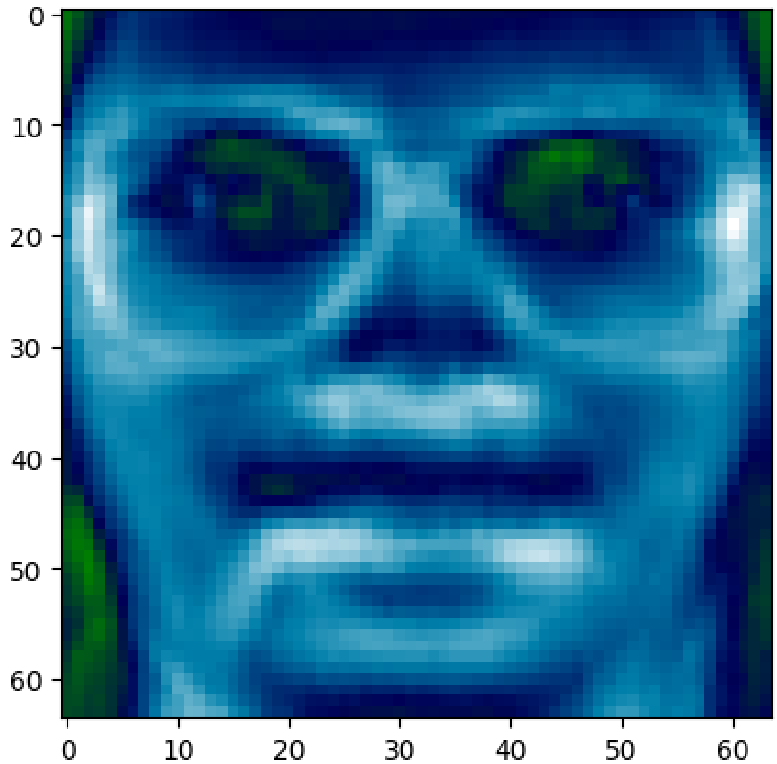

0
10
20
30
40
50
60
0
10
20
30
40
50
60

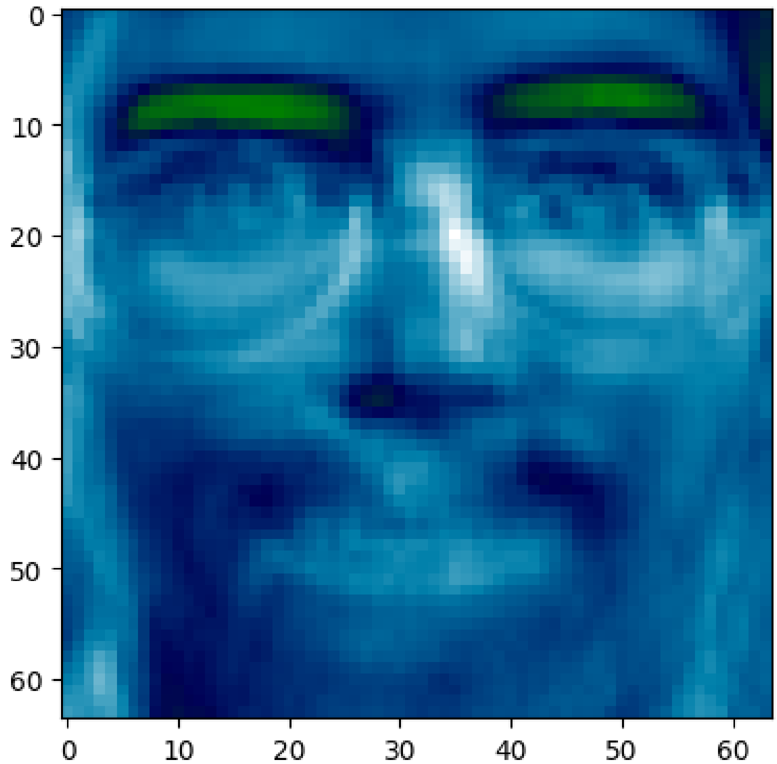

0
10
20
30
40
50
60
0
10
20
30
40
50
60

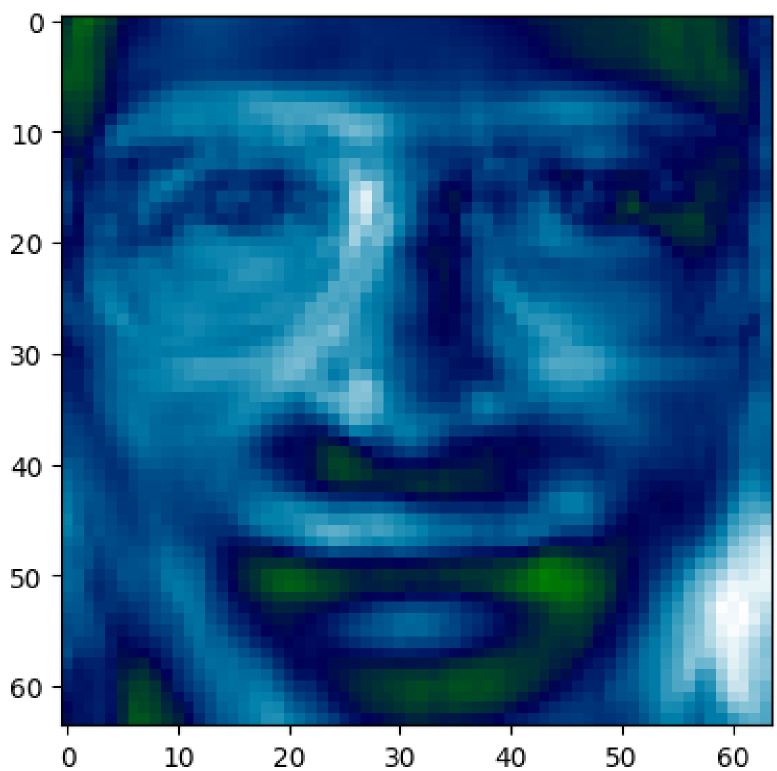

0
10
20
30
40
50
60
0
10
20
30
40
50
60


0
10
20
30
40
50
60
0
10
20
30
40
50
60


0
10
20
30
40
50
60
0
10
20
30
40
50
60


0
10
20
30
40
50
60
0
10
20
30
40
50
60


0
10
20
30
40
50
60
0
10
20
30
40
50
60


0
10
20
30
40
50
60
0
10
20
30
40
50
60

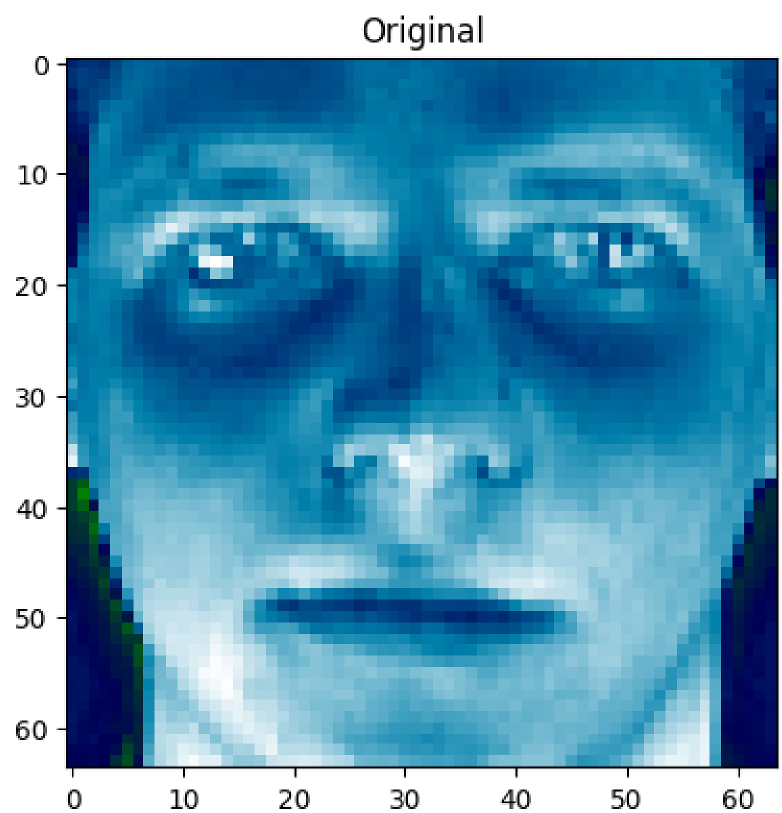
0.2.4 d)

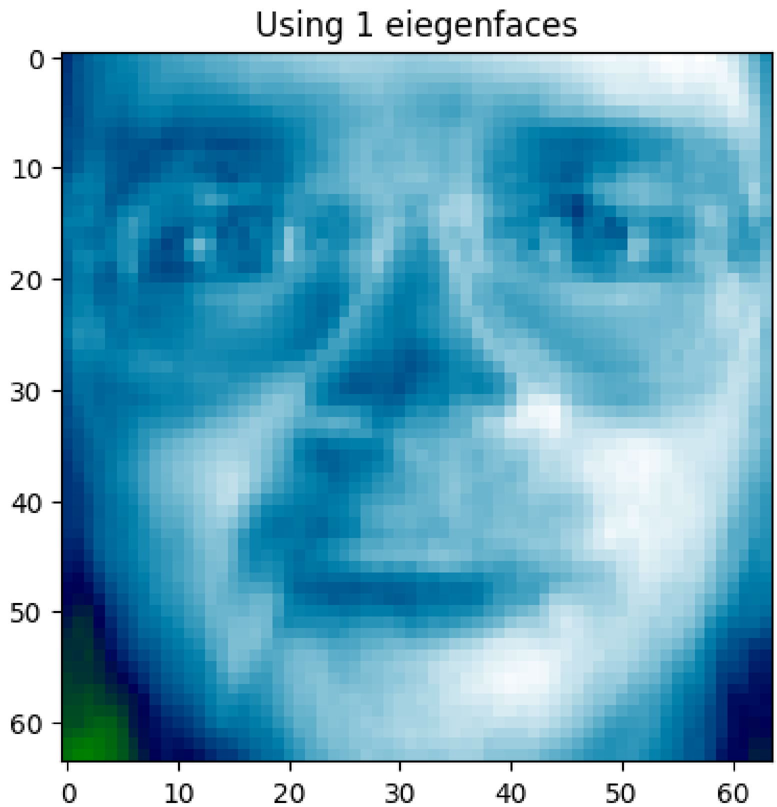
Below I reconstruct the first face. I begin by storing the first face in face1.

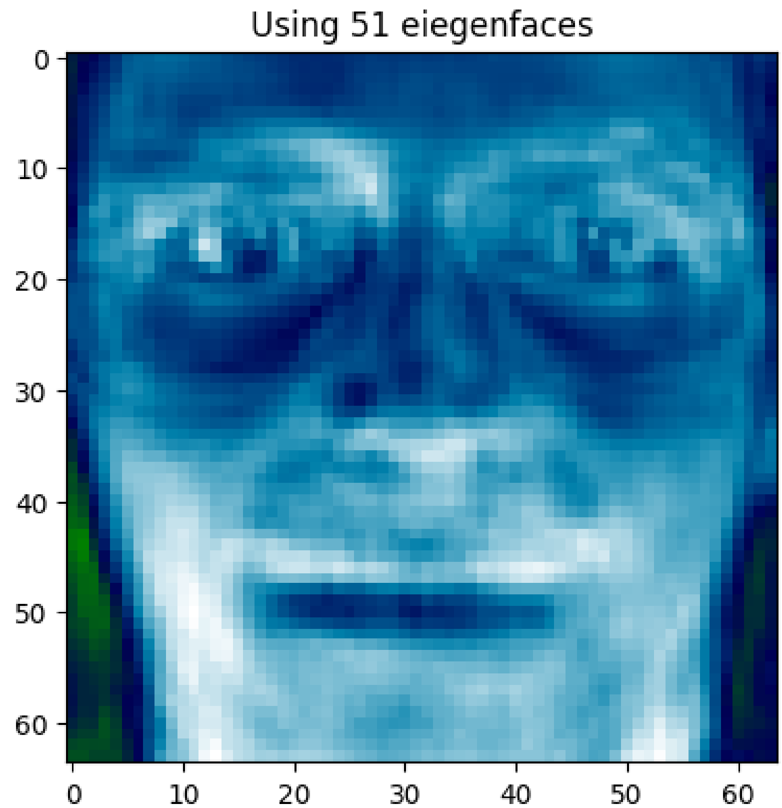
[104]:
```
face1= df.iloc[0]
```

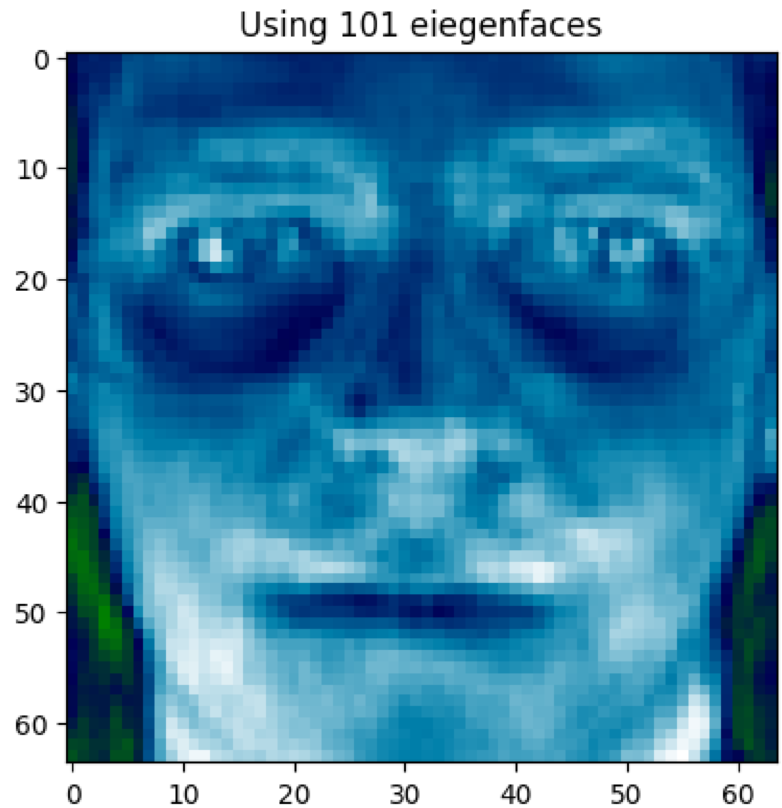
Then, for each number of eigenfaces, i fit a new PCA model and plot the result.

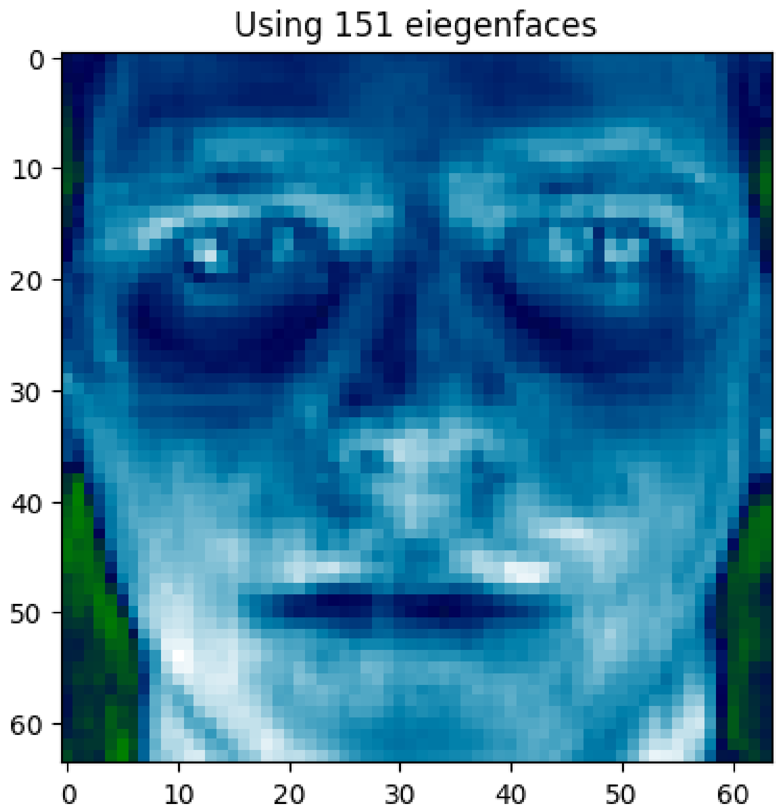
[124]:
```
efs = [1, 51, 101, 151, 201, 251, 301, 351]

plt.imshow(face1.values.reshape(64, 64), cmap='ocean')
plt.title("Original")
plt.show()

for comps in efs:
    cur_pca = PCA(n_components=comps)
    cur_pca.fit(df)
    cur_comps = cur_pca.transform(face1.values.reshape(1, -1))
    cur_face = cur_pca.inverse_transform(cur_comps)
    plt.imshow(cur_face.reshape(64, 64), cmap='ocean')
    plt.title(f"Using {comps} eiegenfaces")
    plt.show()
```

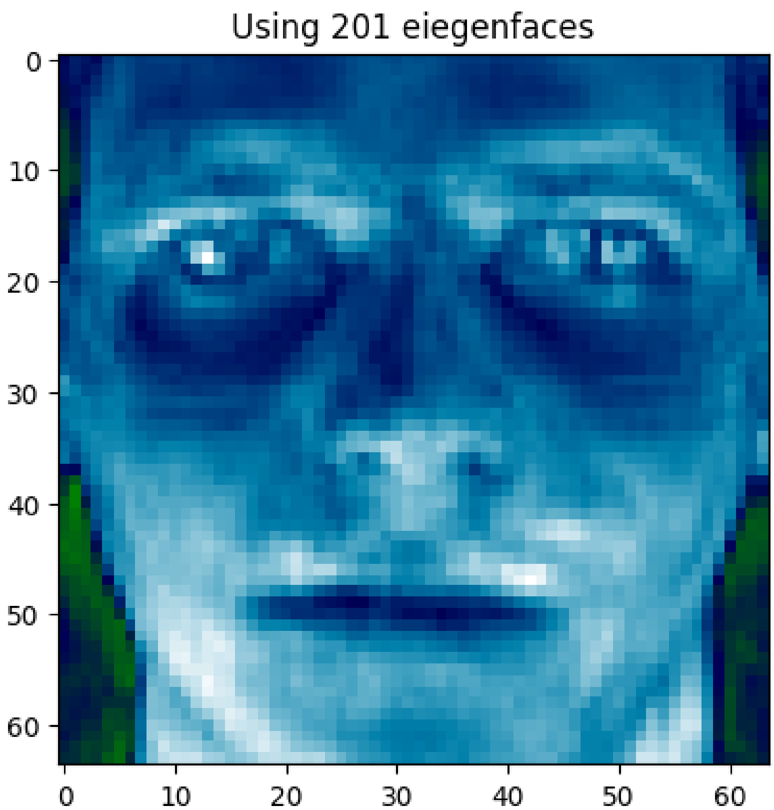


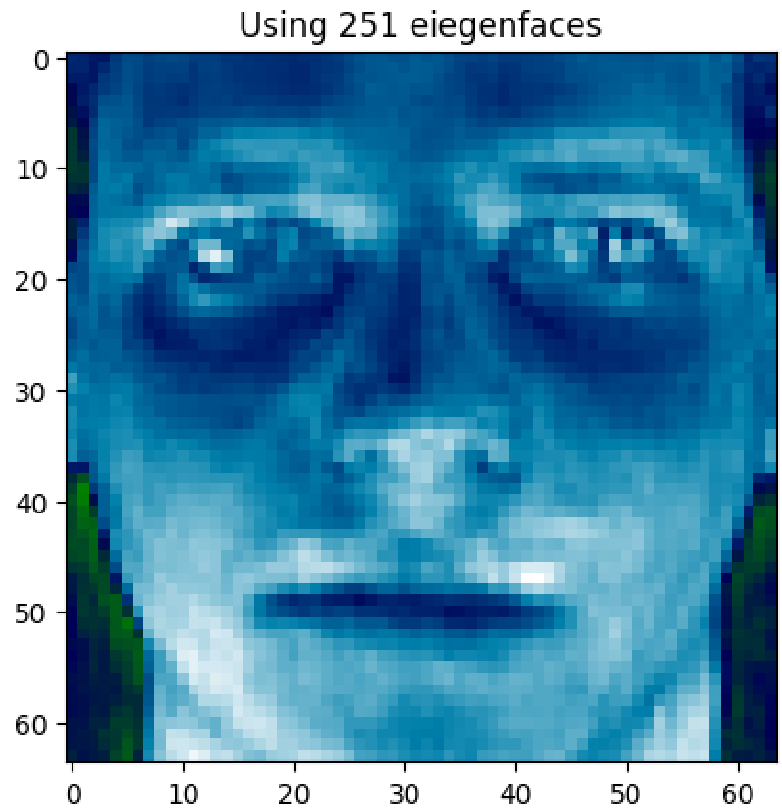

Using 1 eiegenfaces
0
10
20
30
40
50
60
0
10
20
30
40
50
60

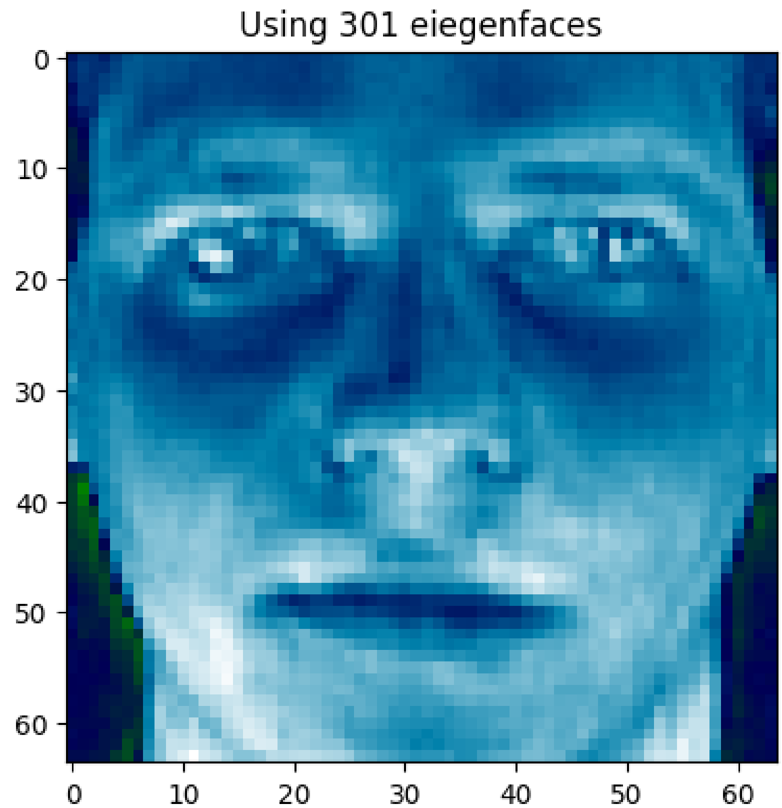

Using 51 eiegenfaces
0
10
20
30
40
50
60
0
10
20
30
40
50
60

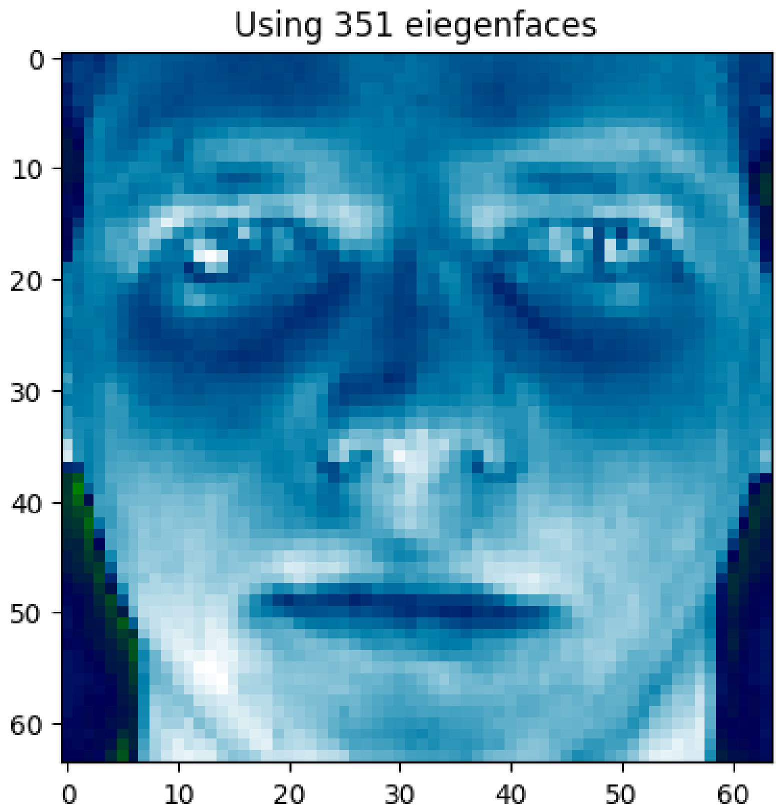

Using 101 eiegenfaces
0
10
20
30
40
50
60
0
10
20
30
40
50
60


Using 151 eiegenfaces
0
10
20
30
40
50
60
0
10
20
30
40
50
60


Using 201 eiegenfaces
0
10
20
30
40
50
60
0
10
20
30
40
50
60


Using 251 eiegenfaces
0
10
20
30
40
50
60
0
10
20
30
40
50
60


Using 301 eiegenfaces
0
10
20
30
40
50
60
0
10
20
30
40
50
60


Using 351 eiegenfaces
0
10
20
30
40
50
60
0
10
20
30
40
50
60

```
[ ]:
```