Praca_domowa_6

May 30, 2021

[1]:
```python
import numpy as np
import pandas as pd

from matplotlib import pyplot as plt

from sklearn.datasets import fetch_olivetti_faces
from sklearn.decomposition import PCA
from sklearn.metrics import mean_squared_error
```

1 Dataset

[2]:
```python
faces = fetch_olivetti_faces(shuffle=True, random_state=123)
```

[3]:
```python
print(faces.DESCR)
```

```
.. _olivetti_faces_dataset:

The Olivetti faces dataset
--------------------------

`This dataset contains a set of face images`_ taken between April 1992 and
April 1994 at AT&T Laboratories Cambridge. The
:func:`sklearn.datasets.fetch_olivetti_faces` function is the data
fetching / caching function that downloads the data
archive from AT&T.

.. _This dataset contains a set of face images:
http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html

As described on the original website:

    There are ten different images of each of 40 distinct subjects. For some
    subjects, the images were taken at different times, varying the lighting,
    facial expressions (open / closed eyes, smiling / not smiling) and facial
    details (glasses / no glasses). All the images were taken against a dark
    homogeneous background with the subjects in an upright, frontal position
    (with tolerance for some side movement).
```

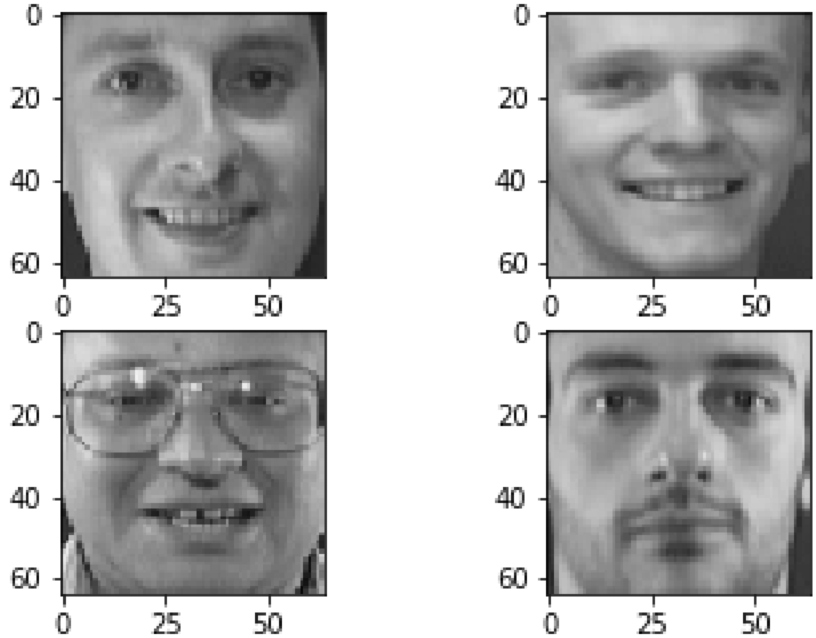
```
**Data Set Characteristics:**

    =================   =====================
    Classes                                40
    Samples total                         400
    Dimensionality                       4096
    Features            real, between 0 and 1
    =================   =====================

The image is quantized to 256 grey levels and stored as unsigned 8-bit
integers; the loader will convert these to floating point values on the
interval [0, 1], which are easier to work with for many algorithms.

The "target" for this database is an integer from 0 to 39 indicating the
identity of the person pictured; however, with only 10 examples per class, this
relatively small dataset is more interesting from an unsupervised or
semi-supervised perspective.

The original dataset consisted of 92 x 112, while the version available here
consists of 64x64 images.

When using these images, please give credit to AT&T Laboratories Cambridge.
```

[4]:
```python
data = faces.data
data.shape
```

[4]:
```
(400, 4096)
```

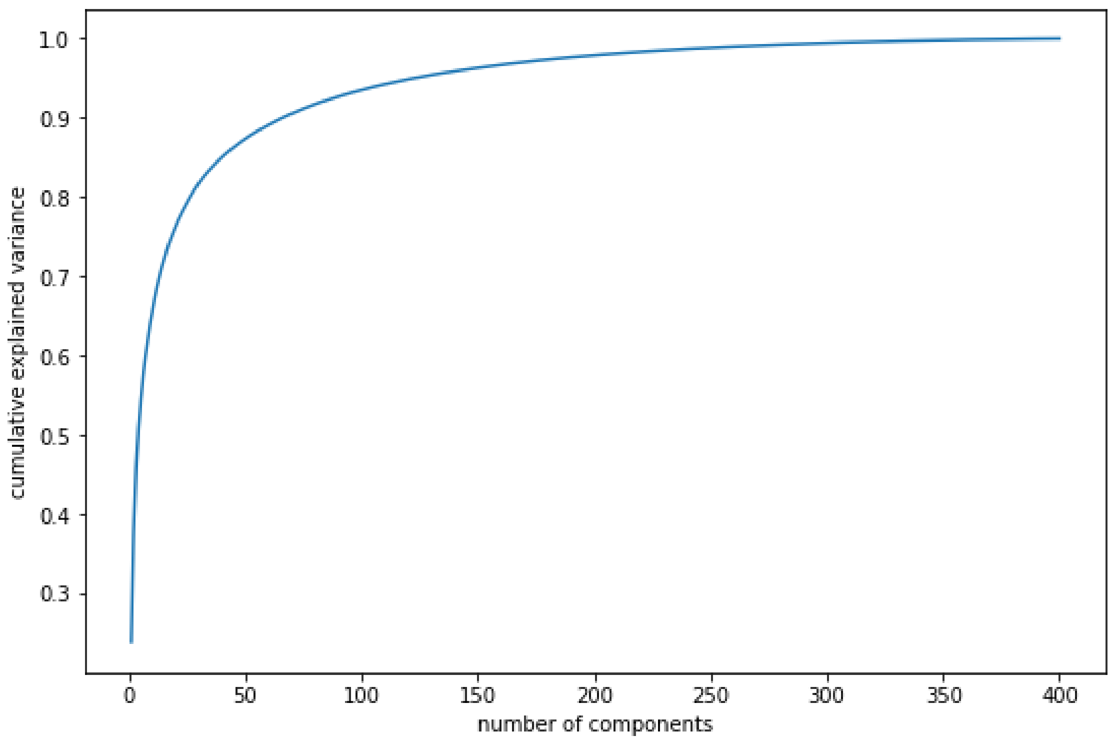
1.1 Wybrane zdjęcia

[5]:
```python
def arr_2_img(data, i=None):
    plt.gray()
    if i is not None:
        plt.subplot(2,2,i+1)
        plt.imshow(data[i].reshape(64,64), interpolation='nearest', vmin=0,␣
 ↪vmax=1)
    else:
        plt.imshow(data.reshape(64,64), interpolation='nearest', vmin=0, vmax=1)

arr_2_img(data, 0)
arr_2_img(data, 1)
arr_2_img(data, 2)
arr_2_img(data, 3)
```



2 PCA

[6]:
```
pca = PCA().fit(data)

plt.figure(figsize=(9,6))
plt.plot(range(1, len(pca.explained_variance_ratio_)+1), np.cumsum(pca.
↪explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance');
```



Przy około 100 komponentach widać załamanie krzywej na wykresie, to znaczy, że powinniśmy dobrać taki parametr `n_components`.

[7]:
```
pca_100 = PCA(n_components=100).fit(data)
data_transformed = pca_100.transform(data)
```

[8]:
```
data_transformed.shape
```

[8]: (400, 100)

[9]:
```
compression = data.shape[1] / data_transformed.shape[1]
print('Stopien kompresji = ' + str(round(compression, 2)))
```

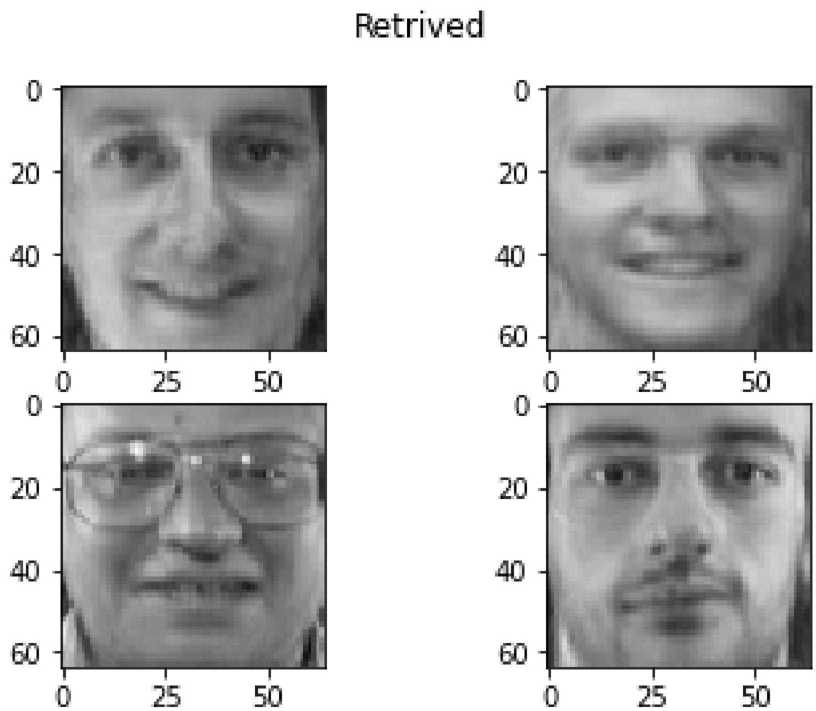
```
Stopien kompresji = 40.96
```

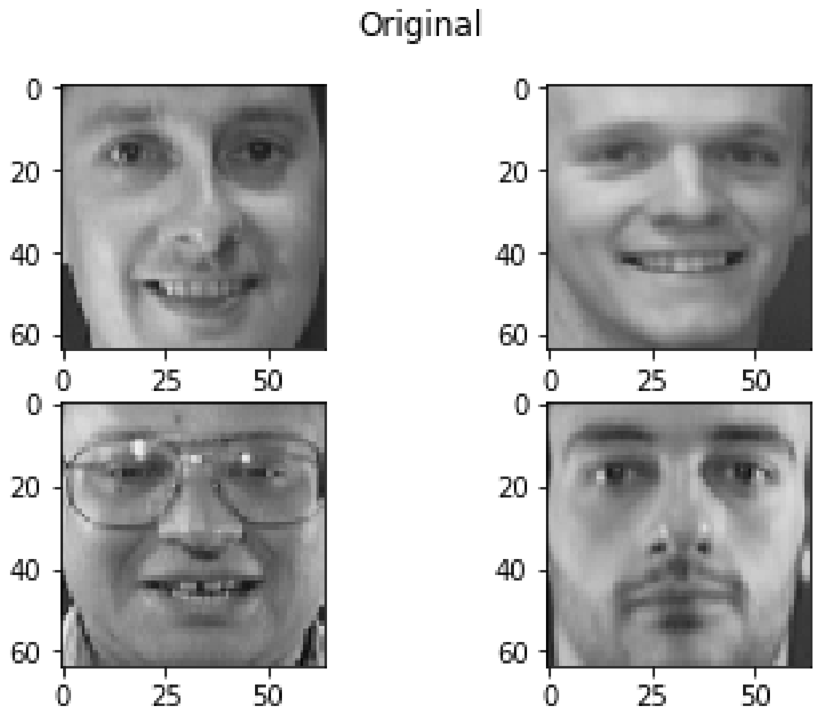
[10]:
```
retrived = pca_100.inverse_transform(data_transformed)
retrived.shape
```

[10]: (400, 4096)

2.1 Porównanie oryginalnych zdjęć z odzyskanymi poprzez odwrotną transformację

[11]:
```
def show_samples(dataset, name):
    arr_2_img(dataset, 0)
    arr_2_img(dataset, 1)
    arr_2_img(dataset, 2)
    arr_2_img(dataset, 3)
    plt.suptitle(name)
    plt.show()

show_samples(retrived, "Retrived")
show_samples(data, "Original")
```






```
[12]: for i in range(4):
          print('RMSE for '+ str(i) + ' photo ' + str(mean_squared_error(data[i],␣
       ↪retrived[i], squared=False)))
```

```
RMSE for 0 photo 0.037413906
RMSE for 1 photo 0.029693998
RMSE for 2 photo 0.038345173
RMSE for 3 photo 0.03613318
```

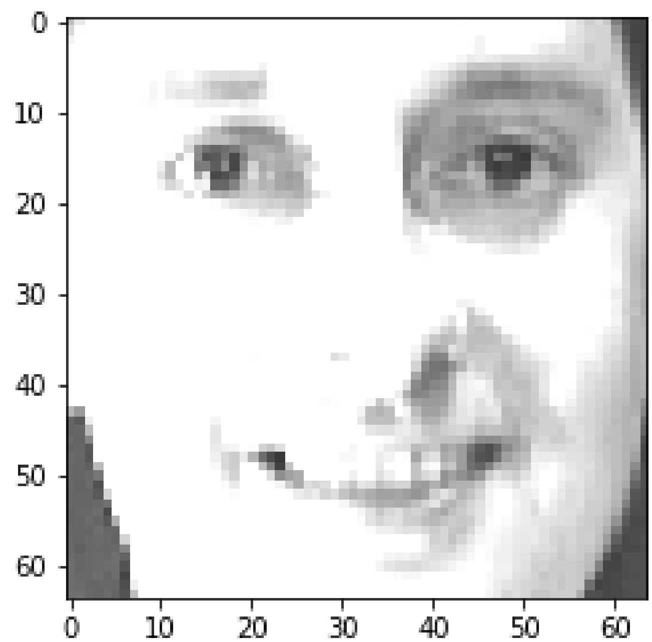
```
[13]: print('RMSE for full dataset =  ' + str(mean_squared_error(data, retrived,␣
       ↪squared=False)))
```

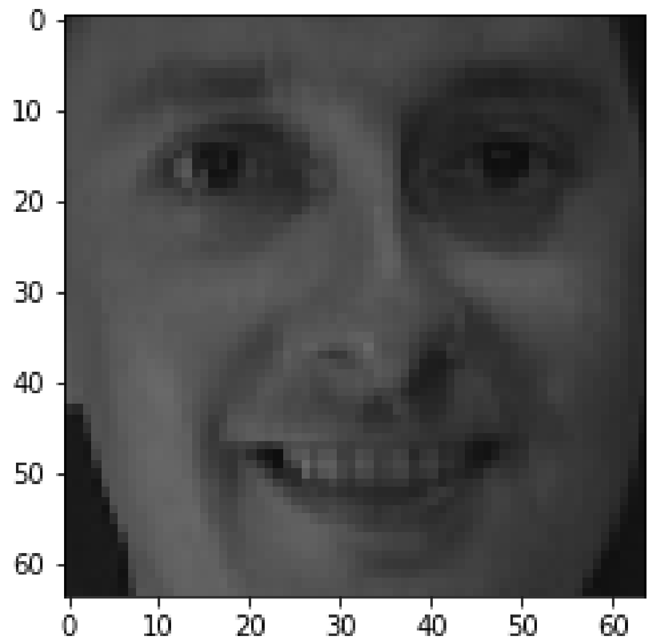
```
RMSE for full dataset =  0.03429769
```

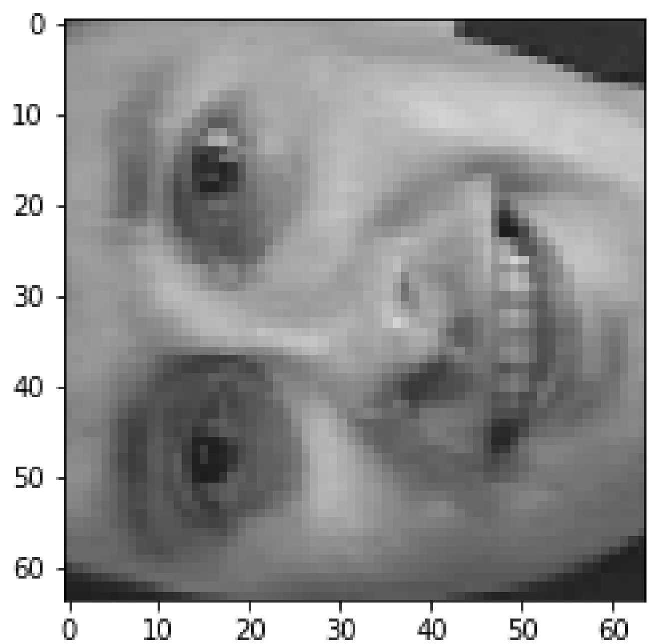
3 Przekształcenie oryginalnych zdjęć

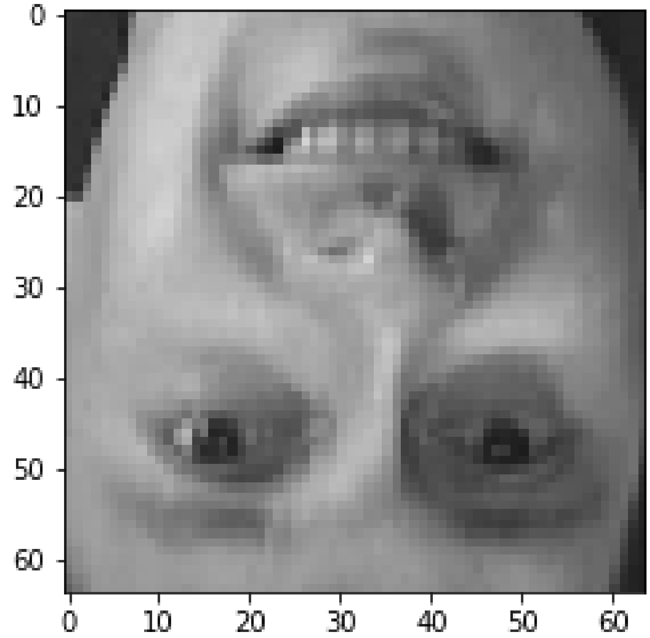
```
[14]: def rotate_90_add_sym(one_d_arr):
          return one_d_arr.reshape(64,64).T.reshape(4096)

      def bright(one_d_arr, factor):
          return one_d_arr * factor

      def flip_ud(one_d_arr):
```

```
    return np.flipud(one_d_arr.reshape(64,64)).reshape(4096)


arr_2_img(bright(data[0], 2))
plt.show()
arr_2_img(bright(data[0], 0.5))
plt.show()
arr_2_img(rotate_90_add_sym(data[0]))
plt.show()
arr_2_img(flip_ud(data[0]))
```




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

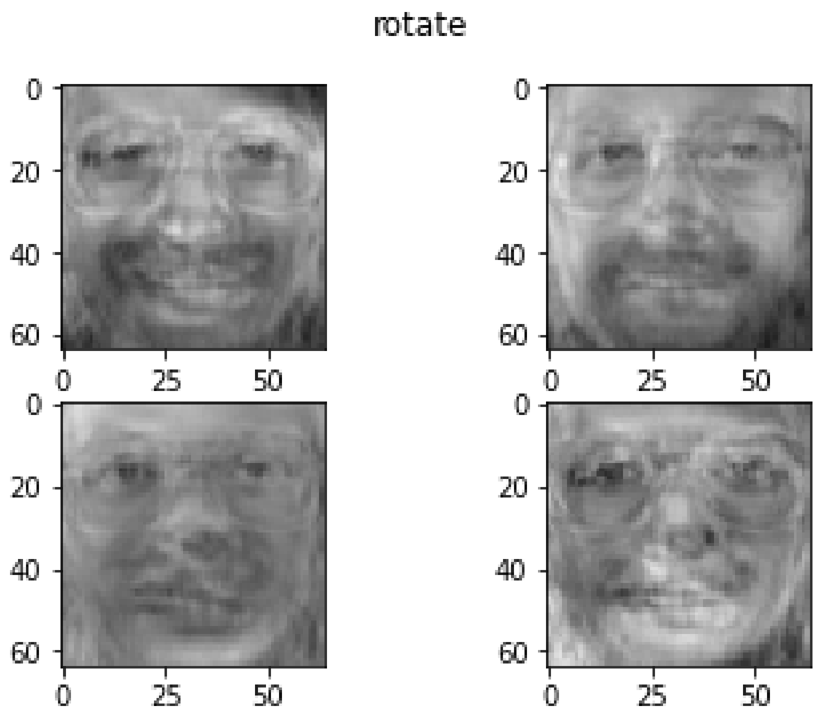

0
10
20
30
40
50
60
0 10 20 30 40 50 60

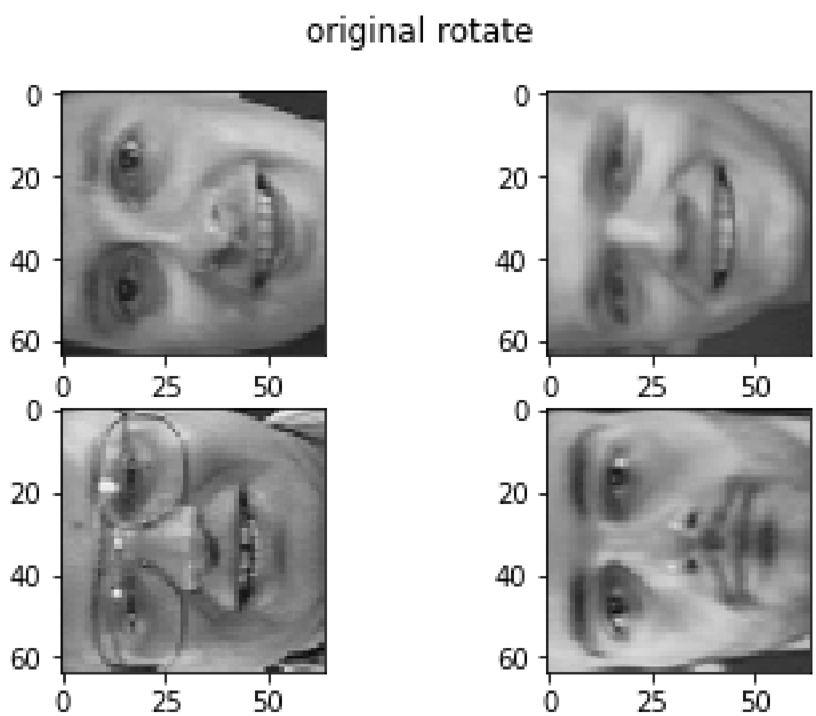


[15]:
```
rotated_data = np.array(list(map(rotate_90_add_sym,data)))
fliped_data = np.array(list(map(flip_ud,data)))
bright_data = np.array(list(map(lambda x: bright(x, 2),data)))
dark_data = np.array(list(map(lambda x: bright(x, 0.5),data)))
datas = {'rotate':rotated_data, 'flip':fliped_data, 'bright':bright_data,␣
 ↪'dark':dark_data}
```

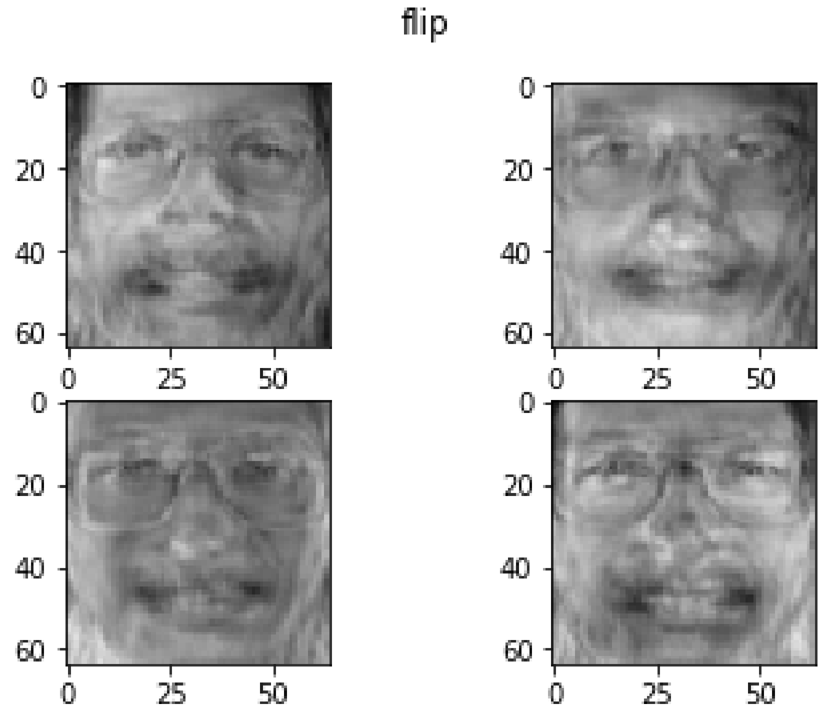
[16]:
```
datas_retrived = {}
for name,item in datas.items():
    datas_retrived[name] = pca_100.inverse_transform(pca_100.transform(item))
```

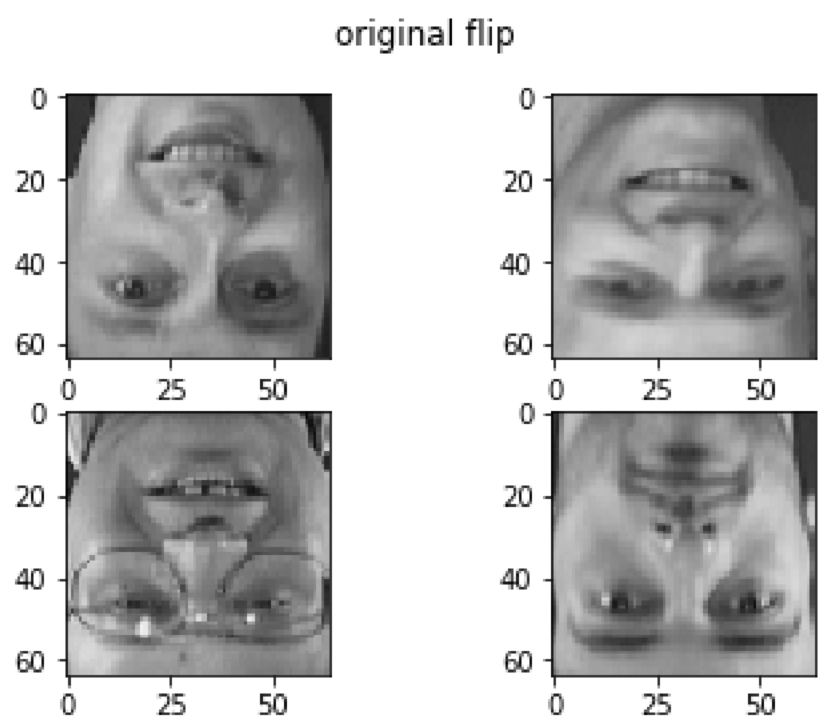
[17]:
```
for name, item in datas_retrived.items():
    show_samples(item, name)
    show_samples(datas[name], 'original '+name)
    print('RMSE for "' + name + '" dataset =  ' +
          str(mean_squared_error(datas[name], datas_retrived[name],␣
 ↪squared=False)))

show_samples(data, "Original")
```


rotate
0
20
40
60
0
25
50


original rotate
0
20
40
60
0
25
50

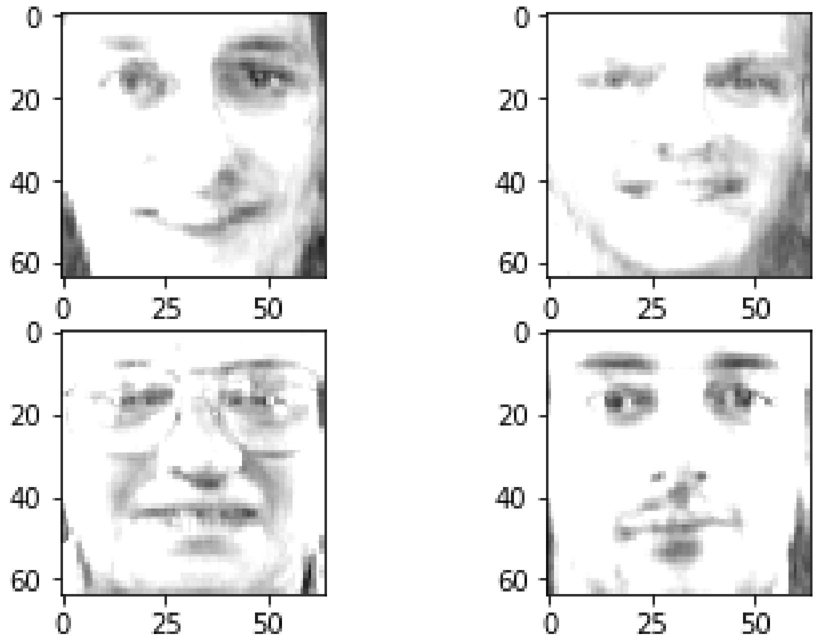
```
RMSE for "rotate" dataset =  0.09184682
```



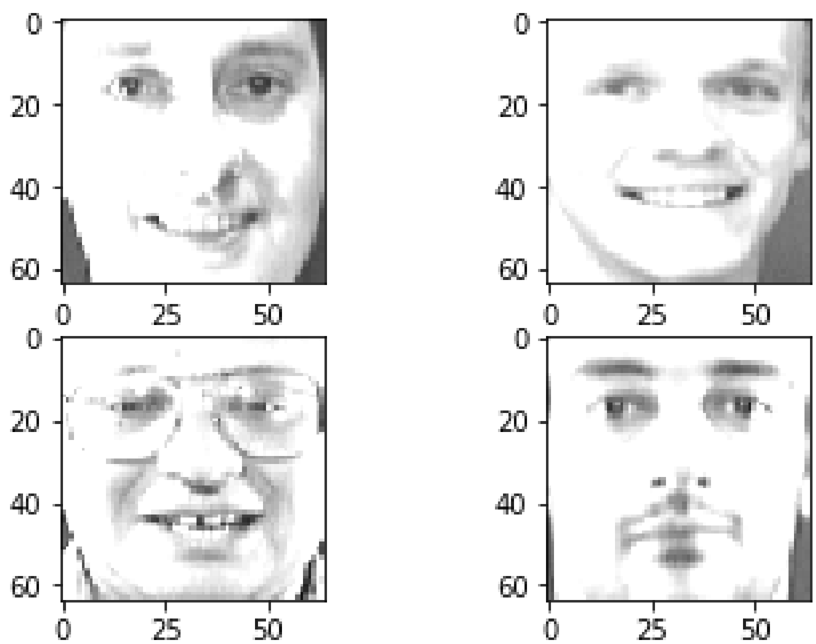


```
RMSE for "flip" dataset =  0.089369446
```

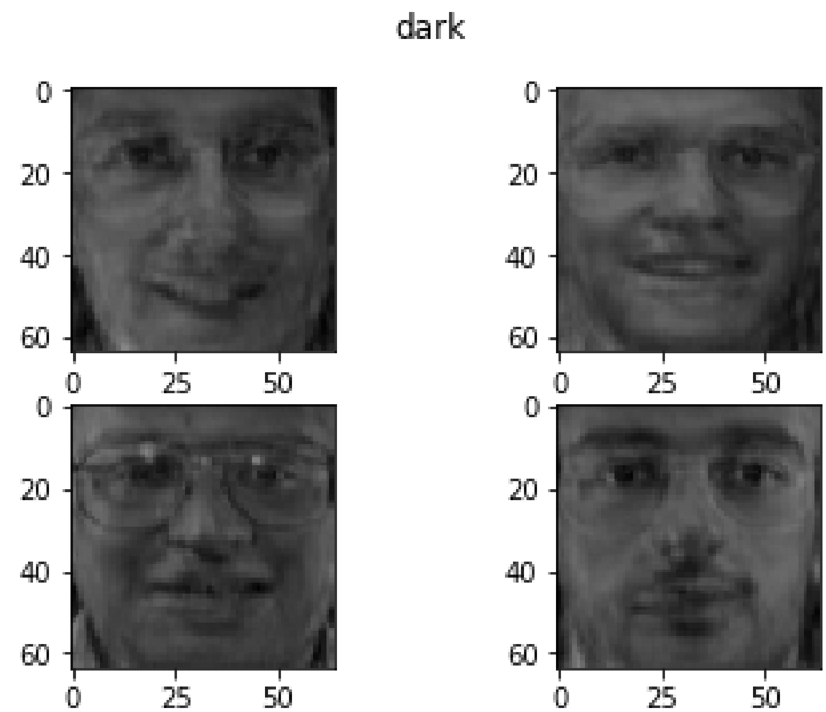
bright

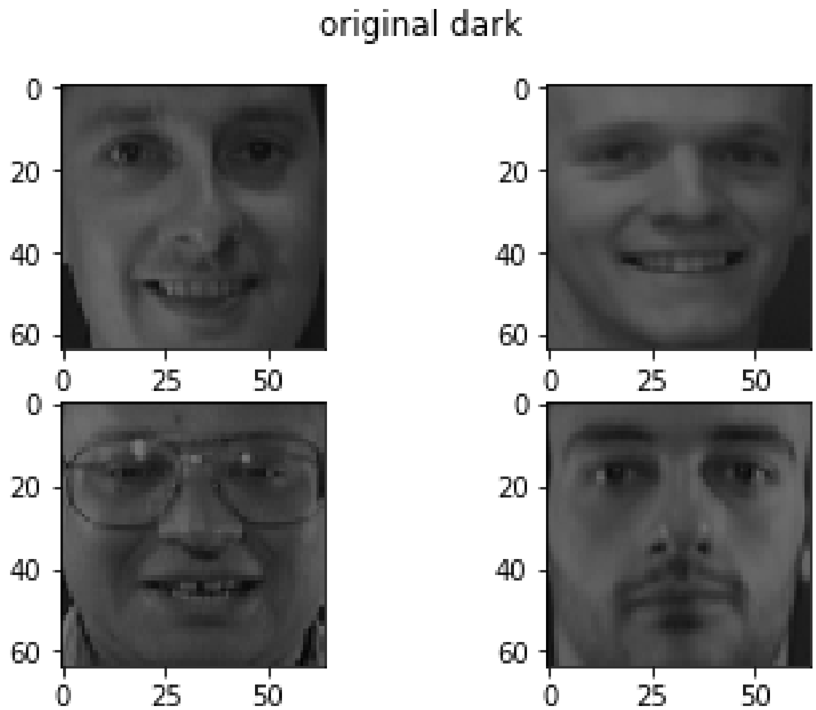


original bright

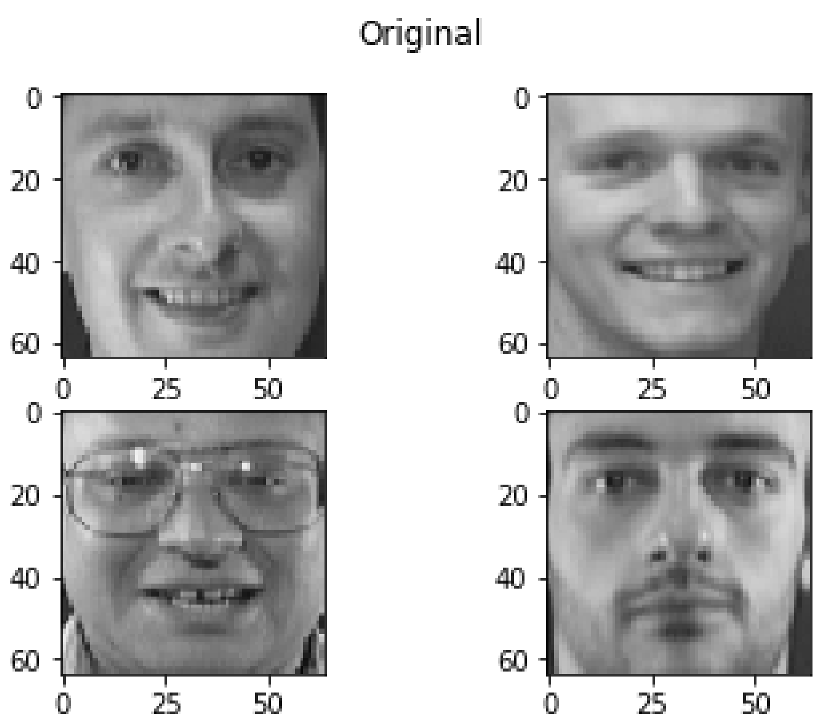


```
RMSE for "bright" dataset =  0.07901445
```





RMSE for "dark" dataset = 0.025259517



RMSE dla danych obróconych lub odbitych symetrycznie jest największe, wizualnie zdjęcia po odwrotnej transformacji też nie przypominają tych przed PCA. Co ciekawe, jasne zdjęcia też mają duże RMSE, ale spowodowane jest to wzrostem bezwzględnych wartości poszczególnych pikseli. Z tego samego powodu RMSE dla przyciemnionych zdjęć jest mniejsze niż dla oryginalnego zbioru. Aby móc porówać RMSE możemy je podzielić przez średnią wartość pikseli, wtedy powinniśmy otrzymać porównywalne wyniki.

[18]:
```
for name, item in datas_retrived.items():
    print('RMSE adjusted for brightness for "' + name + '" dataset =  ' +
          str(mean_squared_error(datas[name], datas_retrived[name],␣
 ↪squared=False) / np.mean(datas[name])))

print('RMSE adjusted for brightness for "original" dataset =  ' +
          str(mean_squared_error(data, retrived, squared=False) / np.
 ↪mean(data)))
```

```
RMSE adjusted for brightness for "rotate" dataset =  0.1678971
RMSE adjusted for brightness for "flip" dataset =  0.16336843
RMSE adjusted for brightness for "bright" dataset =  0.072219685
RMSE adjusted for brightness for "dark" dataset =  0.09234941
RMSE adjusted for brightness for "original" dataset =  0.06269659
```

Teraz widać, że przeskalowane względem średniej jasności RMSE jest najmniejsze dla oryginalnych obrazów, a dla obróconych jest zdecydowananie większe.

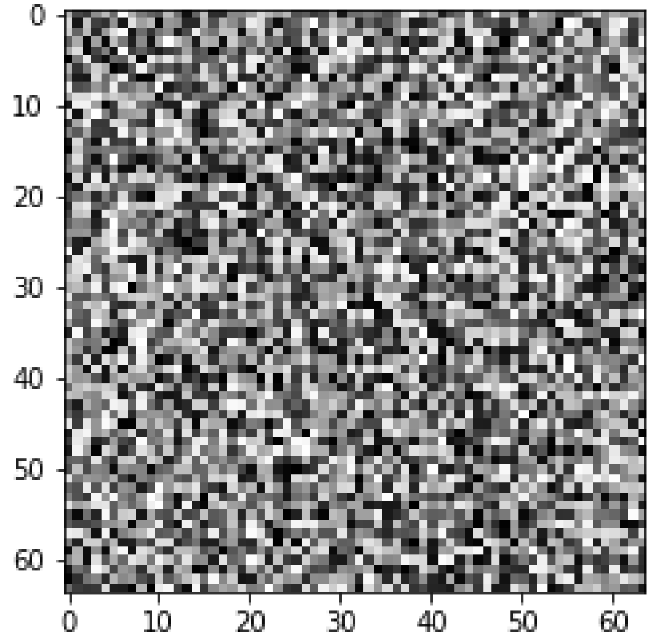
3.1 Do czego może służyć PCA?

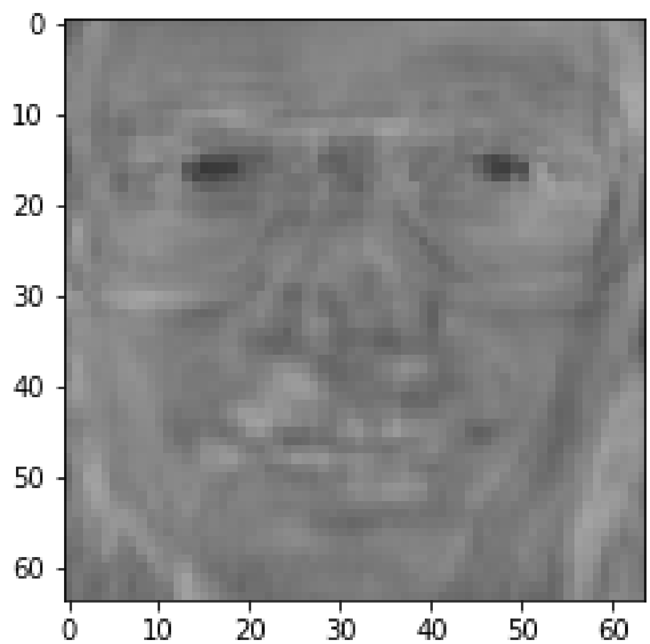
Ten algorytm może służyć do wykrywania niestandardowej orientacji zdjęcia i triggerować automatyczny obrót. Takie narzędzie mogłoby znaleźć zastosowanie w aparatach fotograficznych lub aplikacjach do przeglądania zdjęć. Orientacja wszystkich portretów mogłyby być automatycznie ustawiana.

3.2 PCA losowego szumu

Byłem ciekawy jak wygląda PCA dla losowego obrazka. Poniżej widać że algorytm zapamiętał średnie wysy twarzy i dopasował do nich szum. Widać też zarys okularów.

[19]:
```
np.random.seed(123)
rand = np.random.rand(4096).reshape(1,-1)
arr_2_img(rand)
plt.show()
rand_inv = pca_100.inverse_transform(pca_100.transform(rand))
arr_2_img(rand_inv)
plt.show()
print('RMSE adjusted for brightness for "random" photo =  ' +
          str(mean_squared_error(rand, rand_inv, squared=False) / np.
 ↪mean(rand)))
```






```
RMSE adjusted for brightness for "random" photo =  0.5011007511392306
```