spytek_mikolaj_pd6

May 29, 2021

1 Praca domowa 6

Mikołaj Spytek

[1]:
```
from sklearn.datasets import fetch_olivetti_faces
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
faces = fetch_olivetti_faces()
df = faces.data
images = faces.images
```

[2]:
```
def draw_faces(data):
    fig, axs = plt.subplots(4,4, figsize=(16,16))
    for i in range(16):
        axs[i//4, i%4].imshow(data[i].reshape(64,64), cmap="gray")
    plt.show()

def darken(data, value):
    newdata = []
    for i in range(len(data)):
        if data[i]>value:
            newdata.append(data[i]-value)
        else:
            newdata.append(0)
    return np.array(newdata)
```

1.1 Część 0

Rysowanie niektórych obrazków

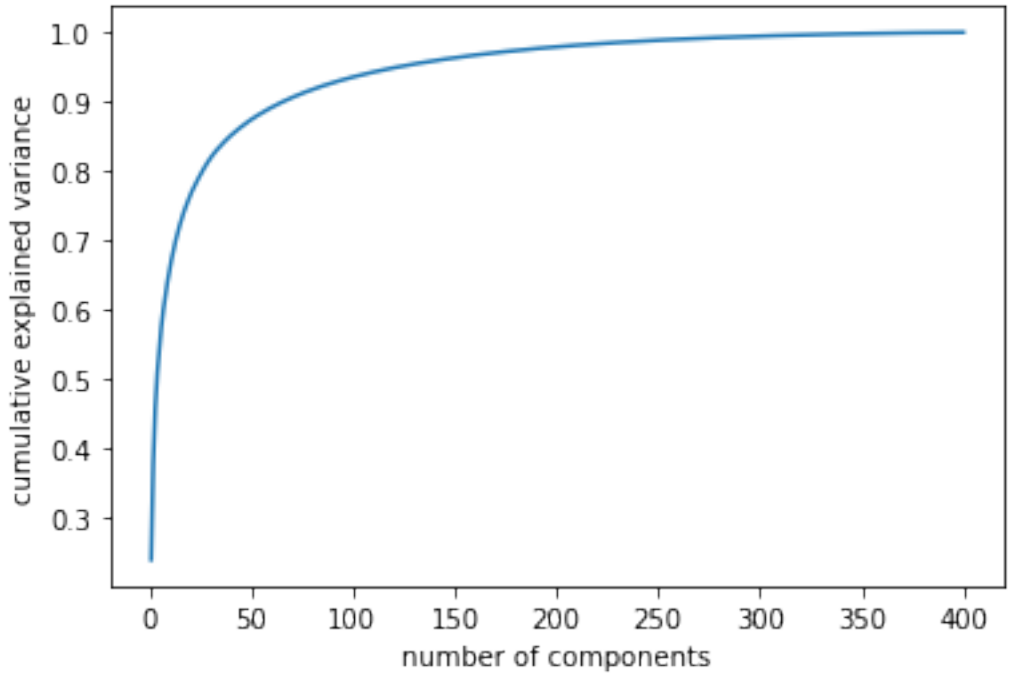
[3]:
```
draw_faces(df)
```

1.2 Część 1

PCA i dobór ilości współrzędnych

[4]:
```
from sklearn.decomposition import PCA

pca = PCA()

pca.fit(df)
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')
plt.show()
```

```
np.cumsum(pca.explained_variance_ratio_)[250]
```



[4]: 0.9887424

Na podstawie wykresu, możemy przyjąć, że 250 będzie odpowiednią liczbą komponentów, ponad 98% wariancji jest już wyjaśnione.

[5]:
```
newpca= PCA(n_components=250)

newpca.fit(df)

df_after_pca = newpca.transform(df)

compression_rate = len(df[0])/len(df_after_pca[0])

print("Stopień kompresji to: {}".format(compression_rate))
```

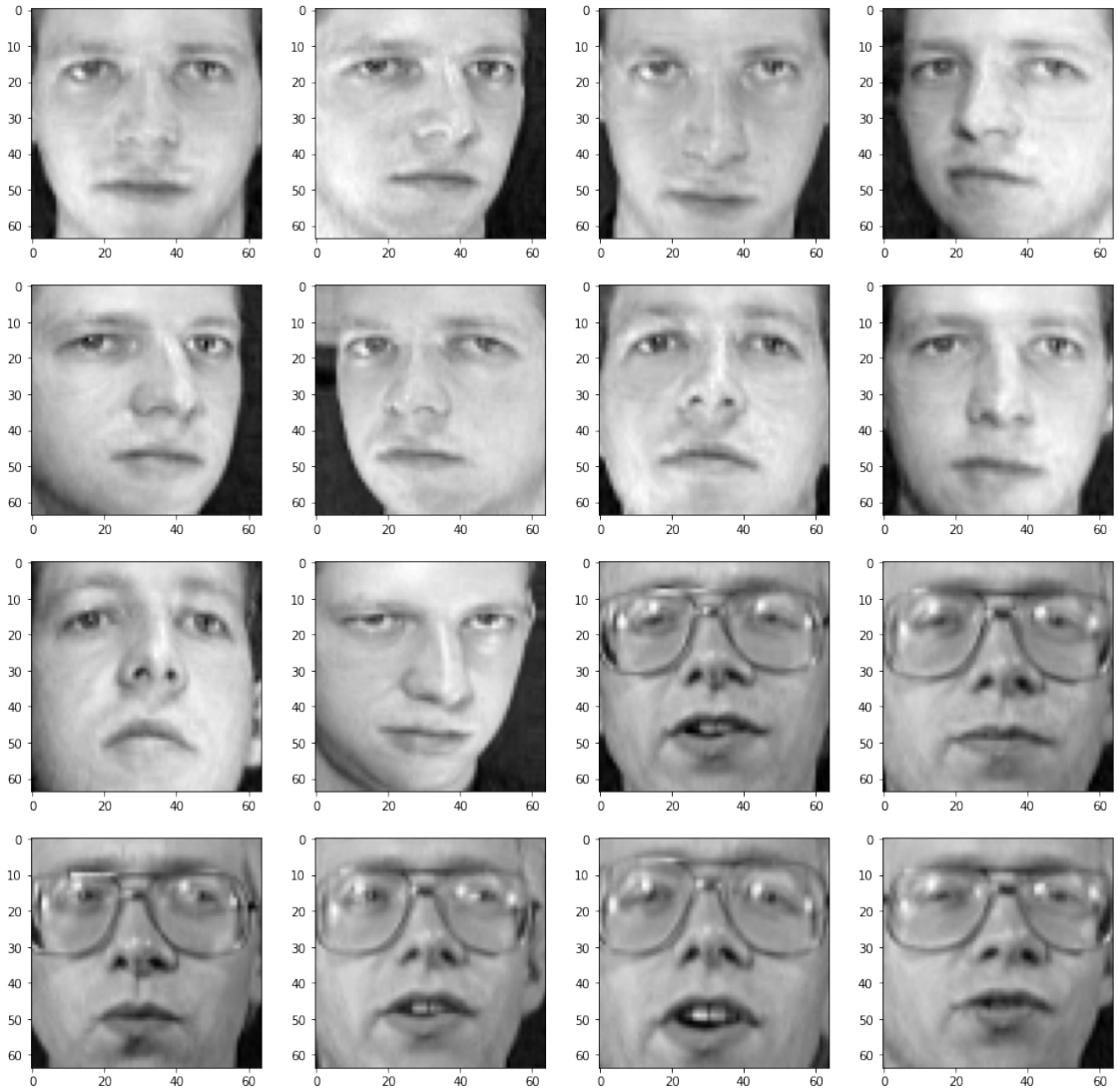
```
Stopień kompresji to: 16.384
```

1.3 Część 2

Przekształcenie odwrotne

[6]:
```
reconstructed = newpca.inverse_transform(df_after_pca)
```

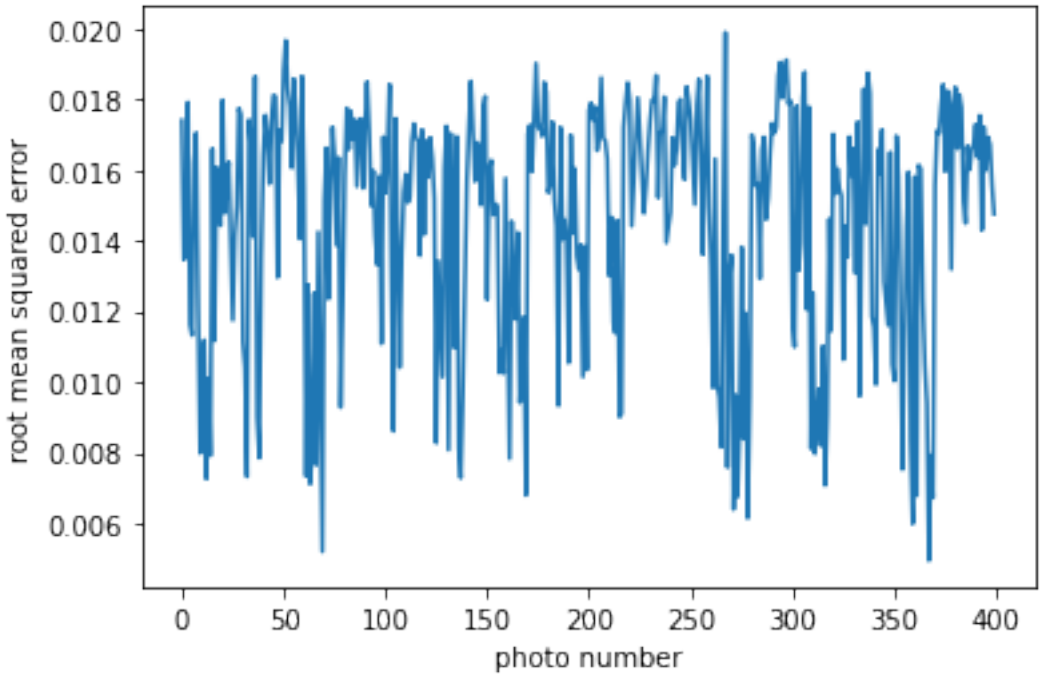
```
draw_faces(reconstructed)
```



Obrazy zrekonstruowane wyglądają bardzo podobnie, różnica jest raczej niezauważalna, jedyne co można powiedzieć, to mała strata jakości.

[7]:
```
from sklearn.metrics import mean_squared_error
scores = []

for i in range(len(df)):
    scores.append(np.sqrt(mean_squared_error(df[i], reconstructed[i])))

plt.plot(scores)
plt.xlabel("photo number")
plt.ylabel("root mean squared error")
```

```
plt.show()

print("Baseline: {}".format(np.sqrt(mean_squared_error(df[0], [0.5 for i in␣
 ↪range(4096)]))))
```



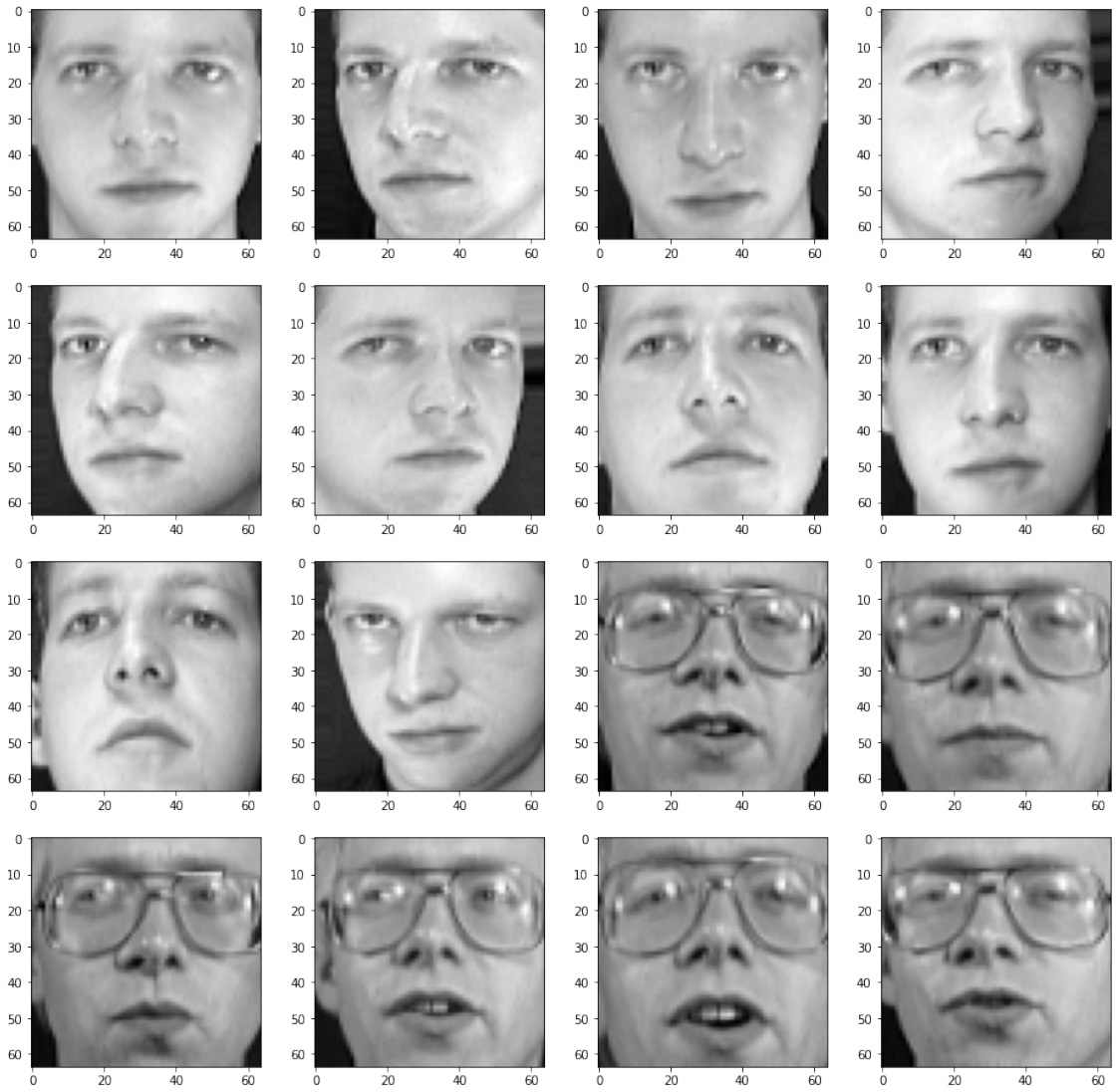

```
Baseline: 0.2025971662871269
```

Jak widać wartość błedu jest niewielka, nie przekracza 0.02, podczas gdy baseline, jest 10x większy.

1.4 Część 3

Przekształcenia niektórych obrazów

[8]:
```
flipped = []
# vertical flip

for i in range(len(df)):
    flipped.append(np.flip(df[i].reshape(64, 64), axis=1).reshape(1, 4096))

draw_faces(flipped)
```



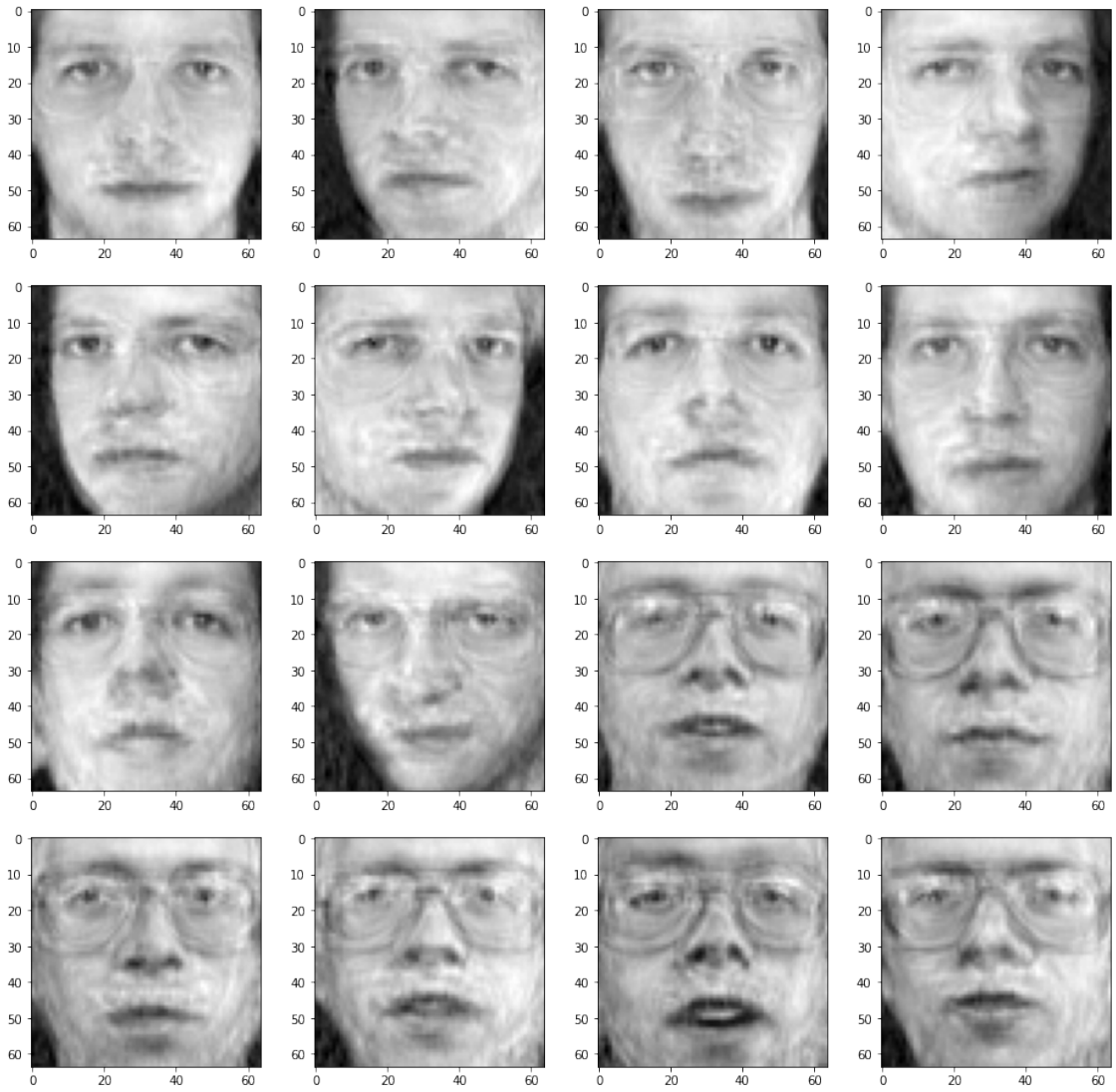
[9]:
```
#rotation
rotated = []

for i in range(len(df)):
    rotated.append(np.rot90(df[i].reshape(64, 64)).reshape(1, 4096))
draw_faces(rotated)
```

[10]:
```
#darkening
darkened = []

for i in range(len(df)):
    darkened.append(darken(df[i], 0.5).reshape(1, 4096))
draw_faces(darkened)
```

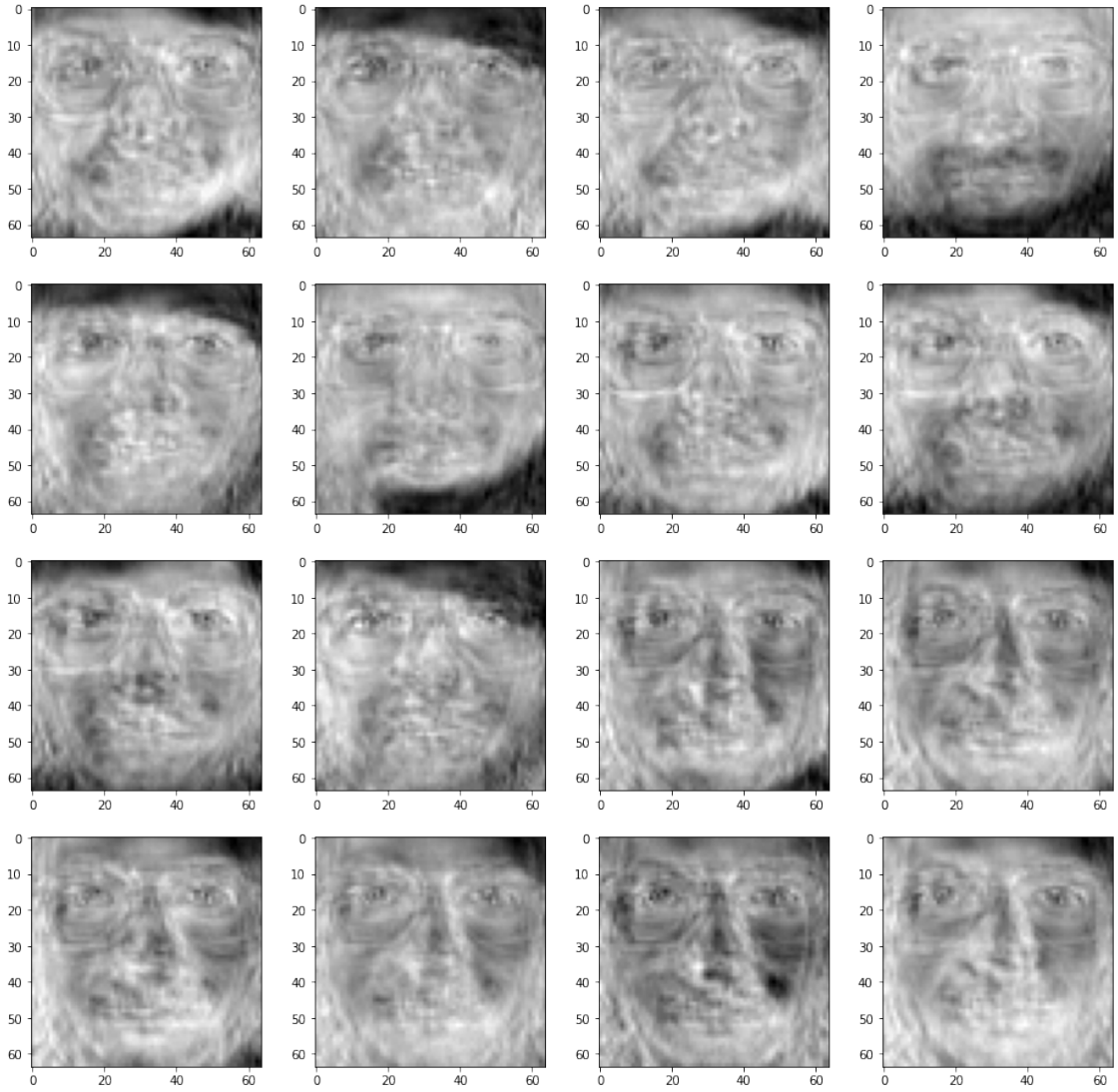
1.5 Część 4

PCA i PCA odwrotne

[11]:
```
flipped_arr = np.array(flipped).reshape(len(df), 4096)
transformed_flipped = newpca.transform(flipped_arr)
reversed_flipped = newpca.inverse_transform(transformed_flipped)

draw_faces(reversed_flipped)
```



[12]:
```
rotated_arr = np.array(rotated).reshape(len(df), 4096)
transformed_rotated = newpca.transform(rotated_arr)
reversed_rotated = newpca.inverse_transform(transformed_rotated)

draw_faces(reversed_rotated)
```



[13]:
```
darkened_arr = np.array(darkened).reshape(len(df), 4096)
transformed_darkened = newpca.transform(darkened_arr)
reversed_darkened = newpca.inverse_transform(transformed_darkened)

draw_faces(reversed_darkened)
```

Widać, że odtworzone twarze są bardzo zniekształcone, w szczególności te, które były poddane rotacji.

[14]:
```
errors = []

for i in range(len(df)):
    errors.append(np.mean(np.sqrt(mean_squared_error(flipped_arr[i],␣
 ↪reversed_flipped[i]))))

flipped_error = np.mean(errors)

errors = []

for i in range(len(df)):
```

```
    errors.append(np.mean(np.sqrt(mean_squared_error(rotated_arr[i],␣
 ↪reversed_rotated[i]))))

rotated_error = np.mean(errors)

errors = []

for i in range(len(df)):
    errors.append(np.mean(np.sqrt(mean_squared_error(darkened_arr[i],␣
 ↪reversed_darkened[i]))))

darkened_error = np.mean(errors)

[flipped_error, rotated_error, darkened_error]
```

[14]: `[0.047216013, 0.0799303, 0.03408763805750515]`

Widać, że obrazy zmodyfikowane dostają większe wartości błedu niż nawet najgorsze obserwacje niezmodyfikowane. Można z tego wnioskować, że PCA może w niektórych przypadkach służyć do wykrywania anomaliów.