Karim_Manisha__CAP_6673_004

In [1]:
```
import numpy as np
import pandas as pd
from scipy.io import loadmat
import matplotlib.pyplot as plt
from PIL import Image as im
from scipy.linalg import svd
```

1. Get the data file allFaces.mat from Canvas (you can read .mat files both in Matlab and Python). Load the file.

In [2]:
```
data = loadmat(r"allFaces.mat")
data.keys()
```

Out[2]:
```
dict_keys(['__header__', '__version__', '__globals__', 'faces', 'n', 'm', 'sub',
'nfaces', 'person'])
```

In [3]:
```
faces = data['faces']
nfaces = data['nfaces']
n= data['n']
m = data['m']
```

In [4]:
```
faces.shape
```

Out[4]:
```
(32256, 2410)
```

In [5]:
```
nfaces
```

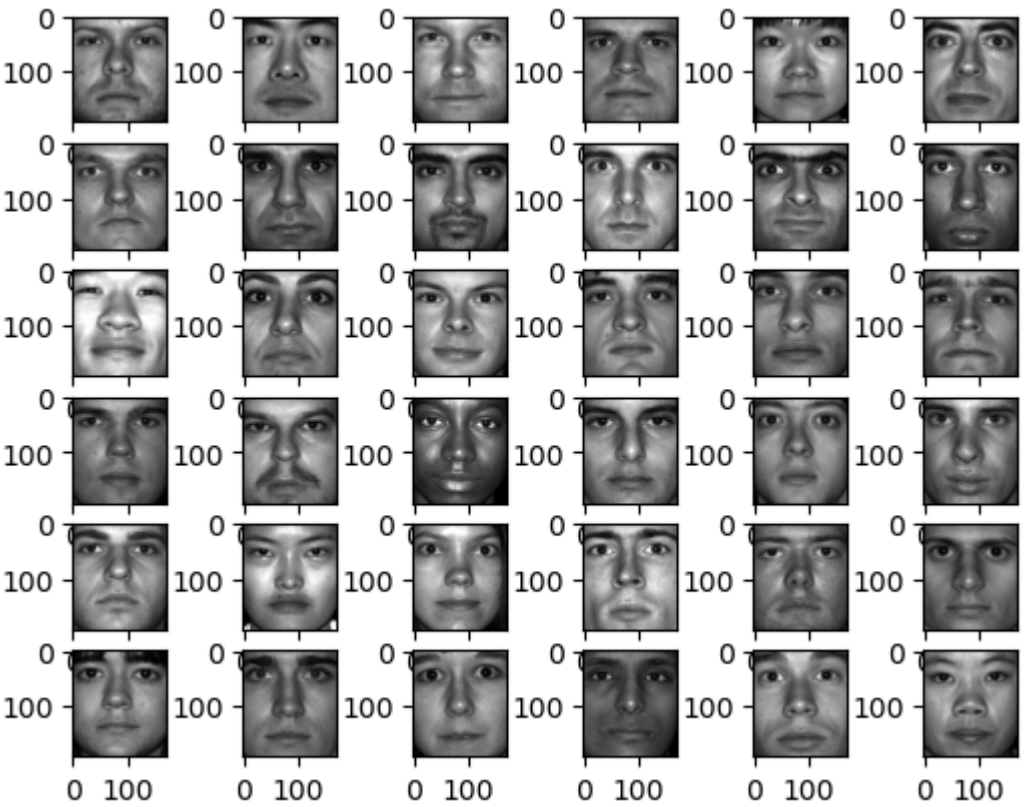
Out[5]:
```
array([[64, 62, 64, 64, 62, 64, 64, 64, 64, 64, 60, 59, 60, 63, 62, 63,
        63, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64,
        64, 64, 64, 64, 64, 64]], dtype=uint8)
```

In [6]:
```
faces = faces.T
```

In [7]:
```
Global_Image_Dict = {}

count = 0

for i in range(nfaces.shape[1]):

    # create an empty array for each person
    Global_Image_Dict[i]  = []

    # read each image for each person
    image = faces[count: count + nfaces[0, i]]

        # append the image to the dictionary key
    Global_Image_Dict[i]=(image)
    count = count + nfaces[0,i]
```

> A. Reading first pictures of person # 1 to person # 36:

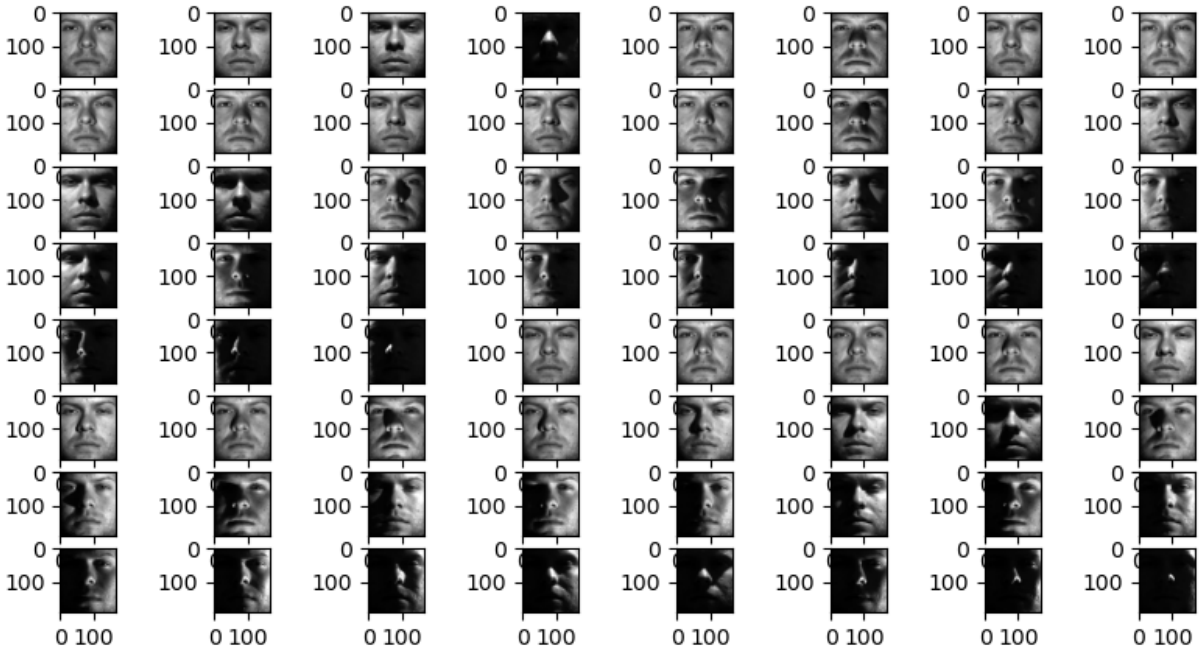
In [8]:
```
for i in range((len(Global_Image_Dict)-2)):
    j = i+1
    plt.subplot(6,6,j)
    plt.imshow(((Global_Image_Dict[i][0]).reshape(168,192).T), cmap='gray')
```



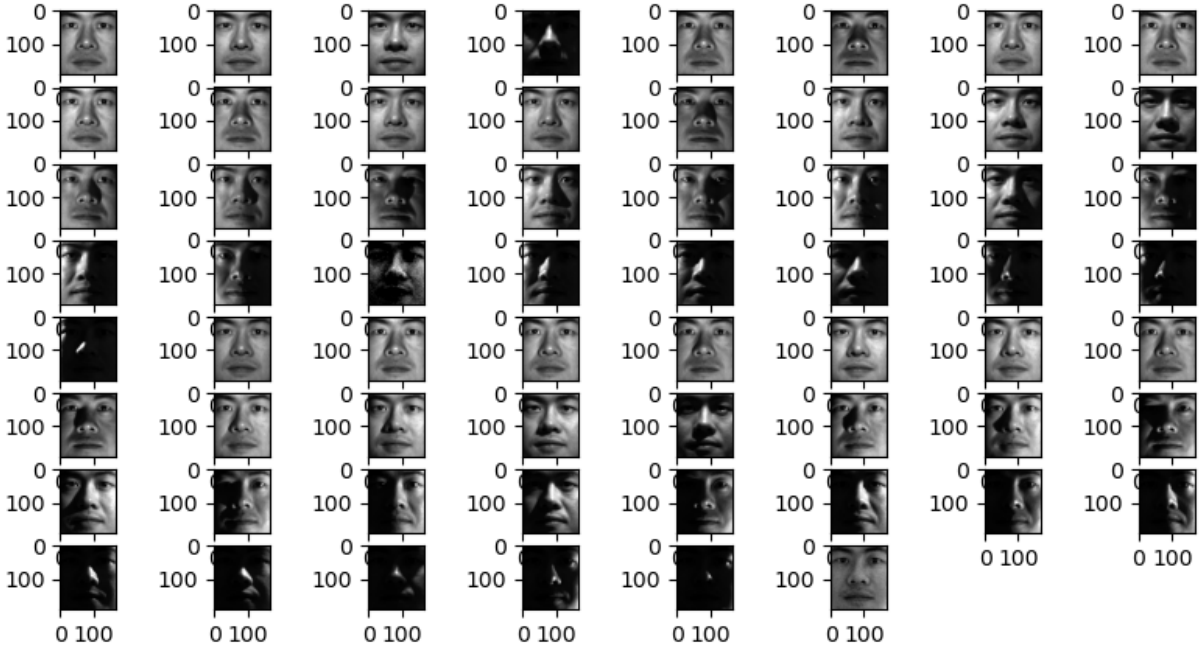
Reading 64 different pictures of Person #1, then Person # 2 ....

In [9]:
```
for i in range(len(Global_Image_Dict)):

    plt.figure(figsize = [10,5])
    print('Person: ', 0 + i)

    for j in range(len(Global_Image_Dict[i])):
        pos = j+1

        plt.subplot(8,8,pos)
        plt.imshow(((Global_Image_Dict[i][j]).reshape(168,192).T), cmap='gray')
    plt.show()
```

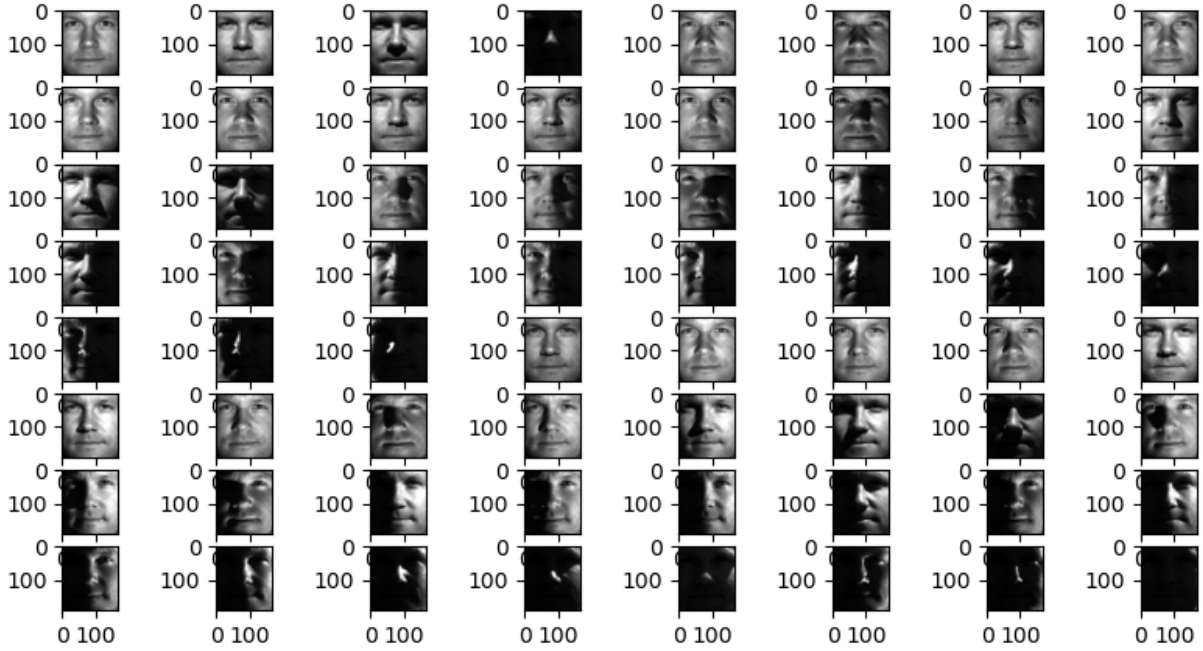
```
Person:  0
```



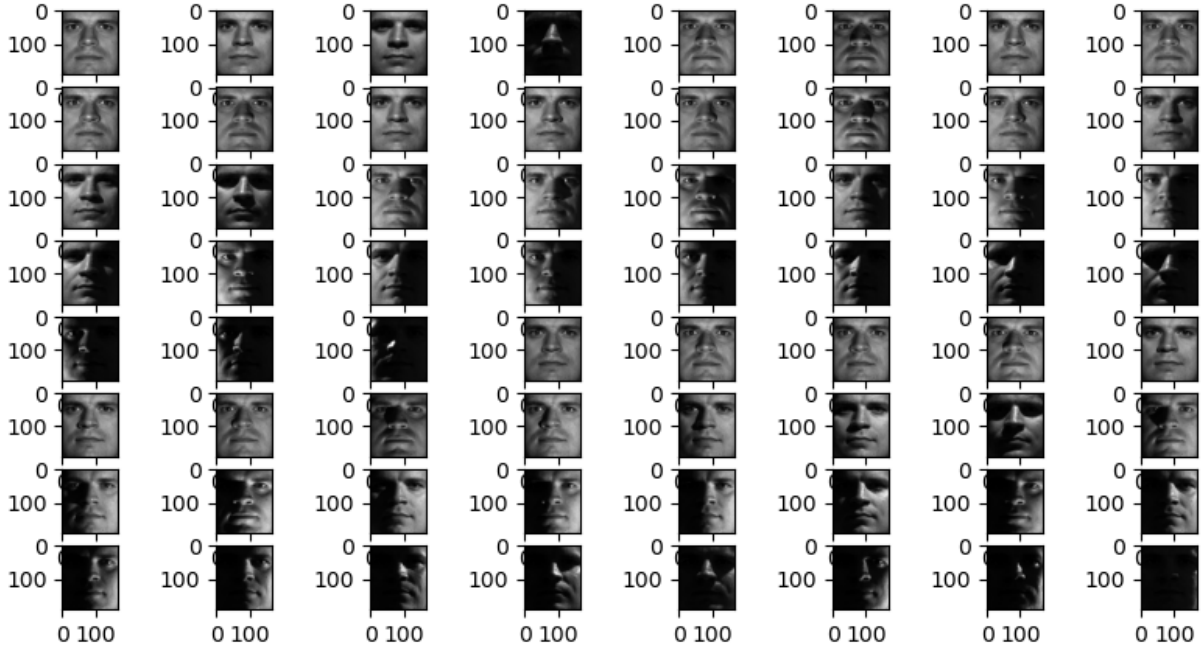

```
Person:  1
```



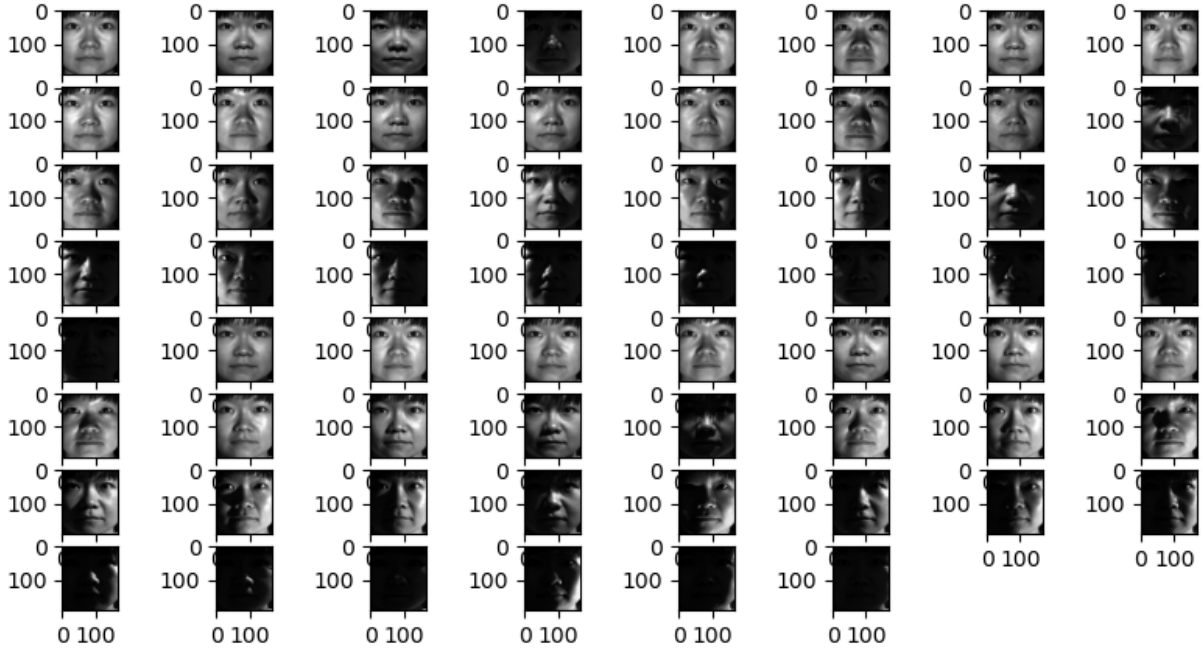

```
Person:  2
```



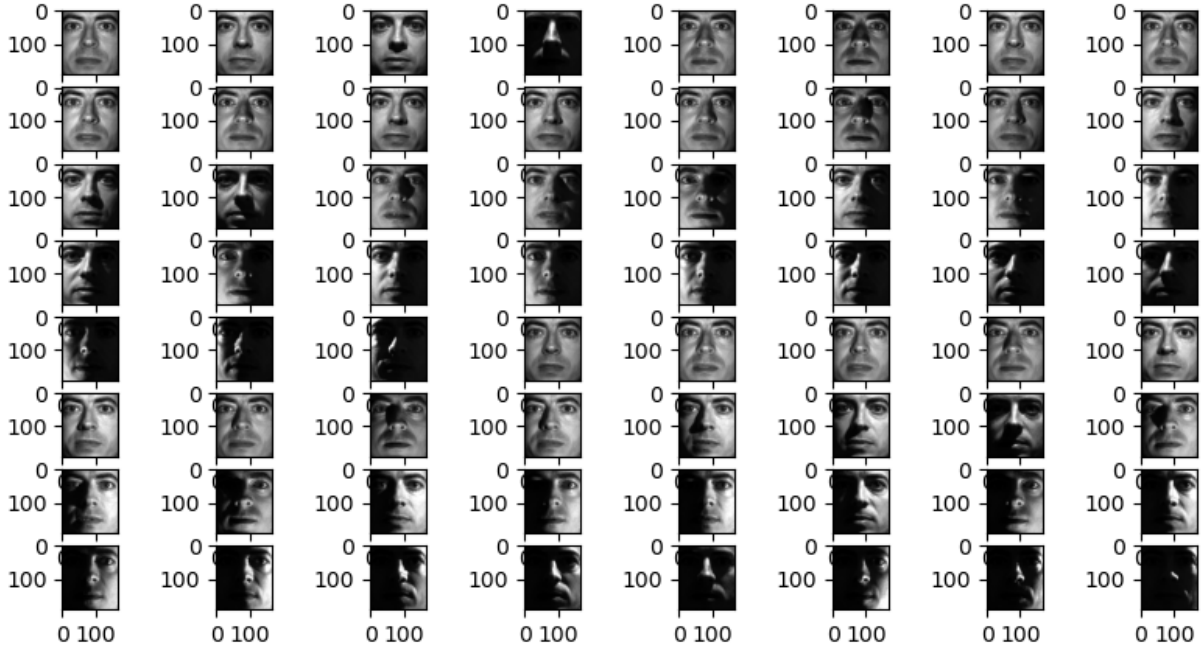
Person: 3



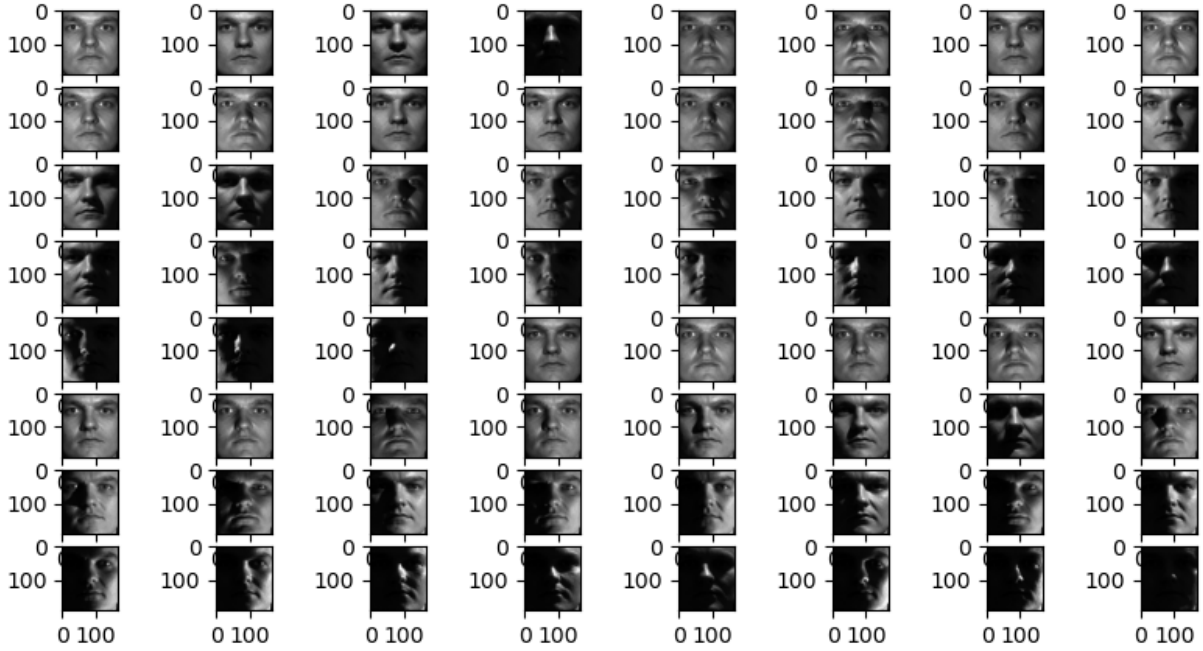
Person: 4



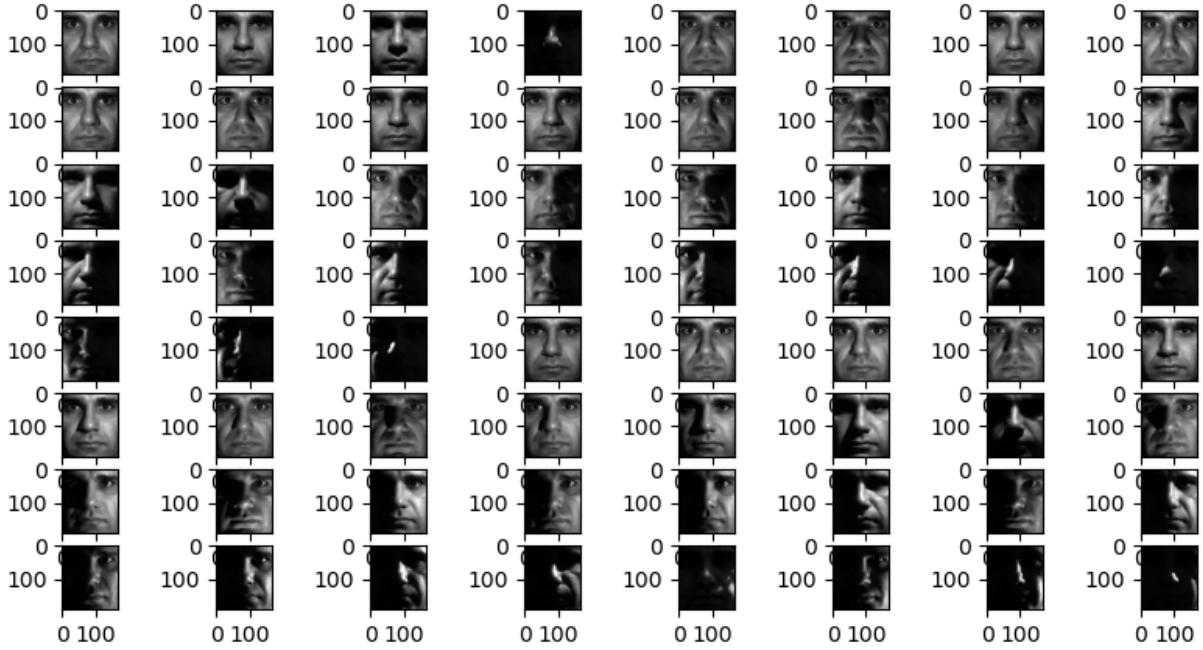
Person: 5



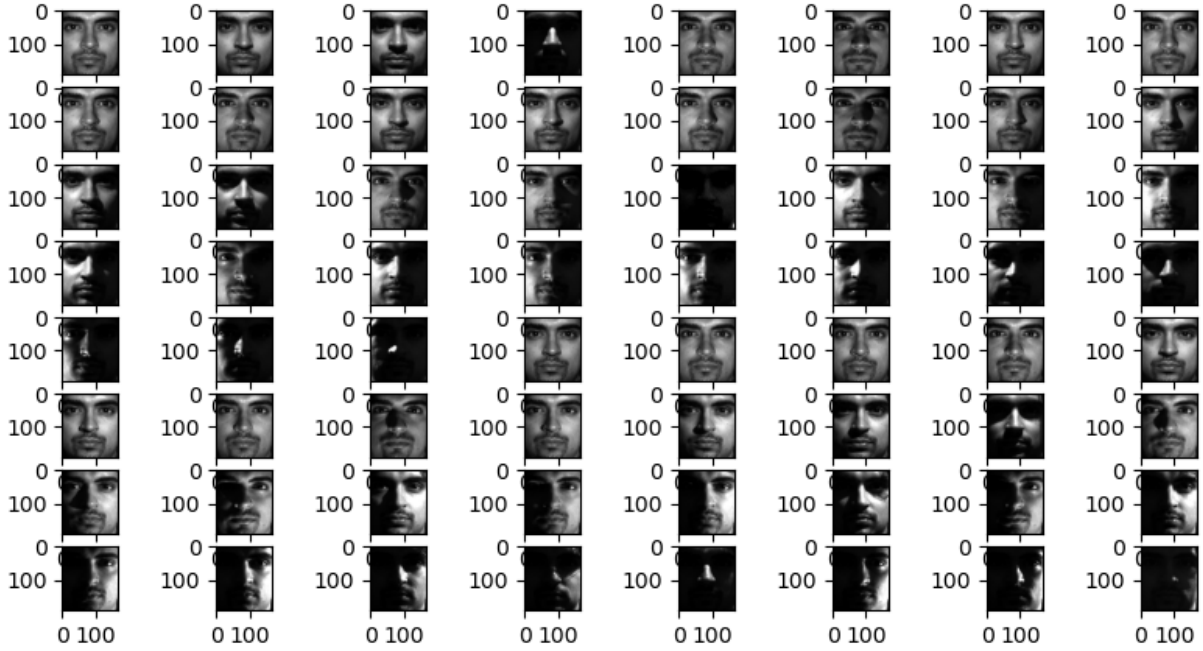
Person: 6



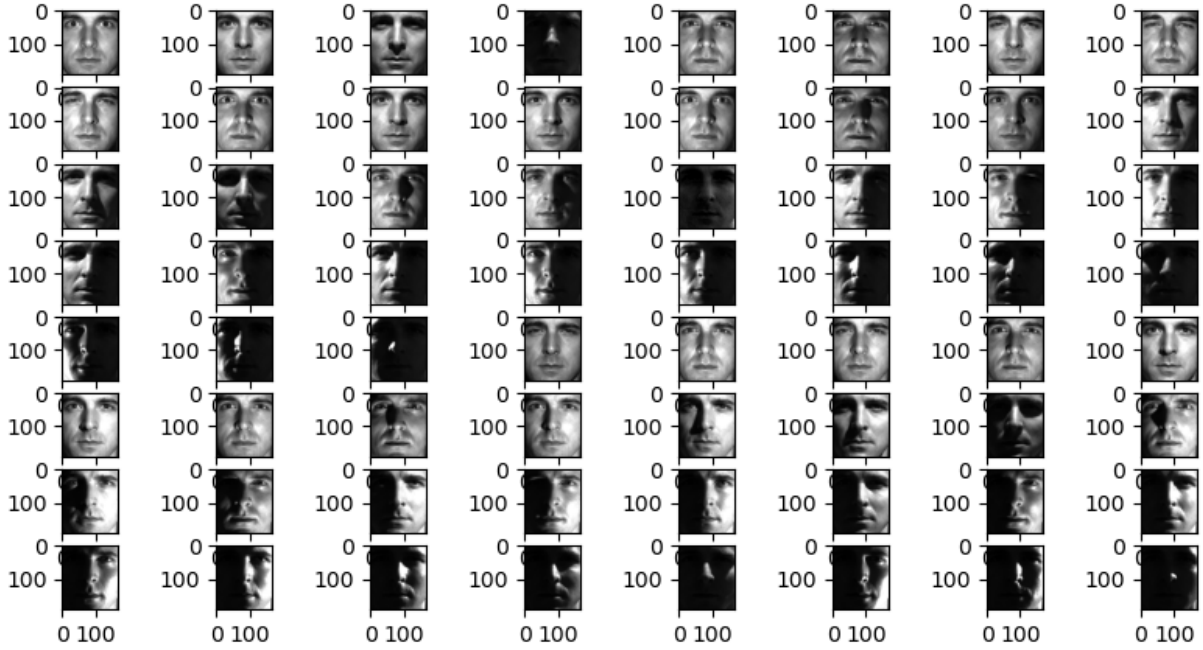
Person: 7



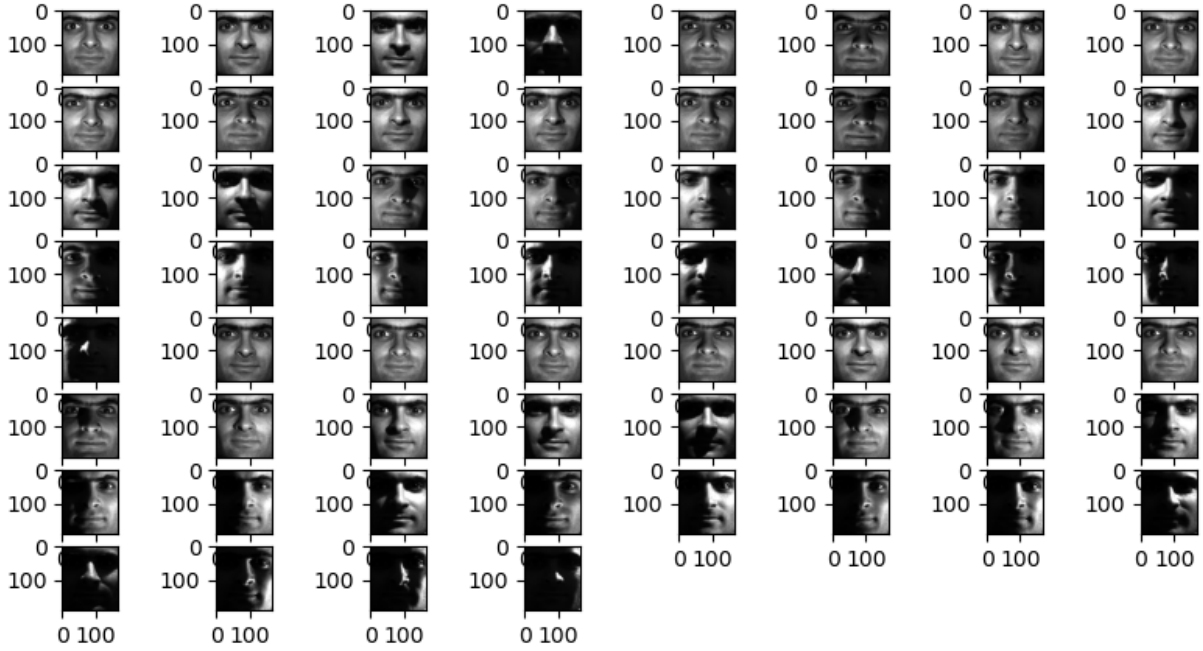
Person: 8



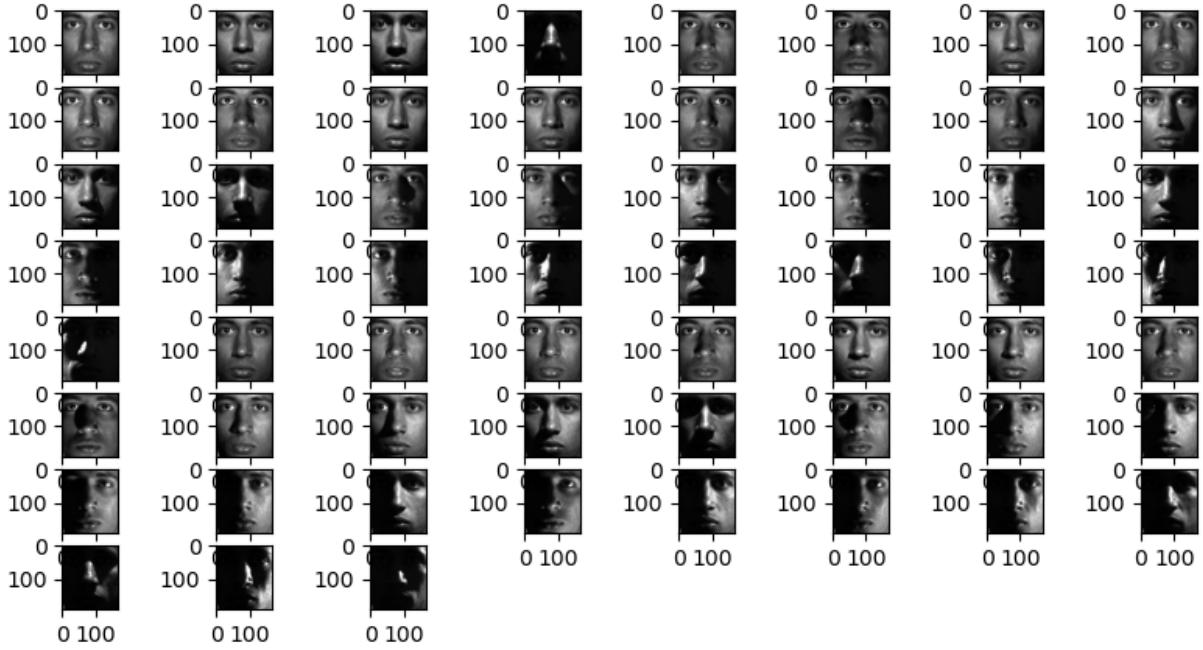
Person: 9



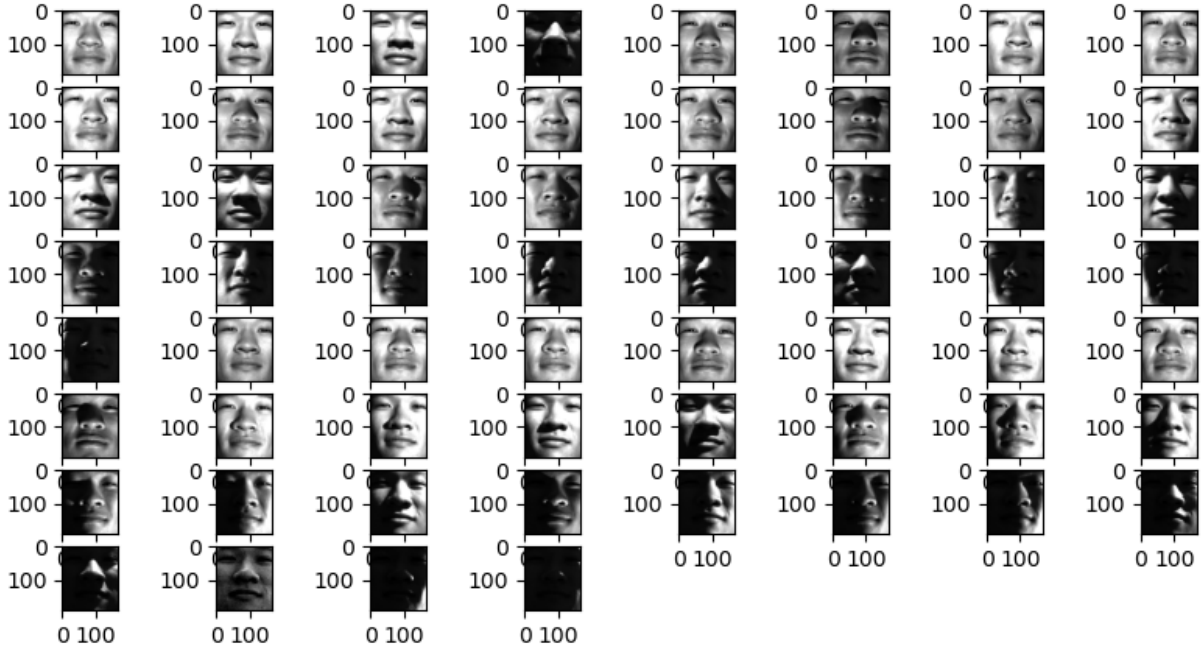
Person: 10



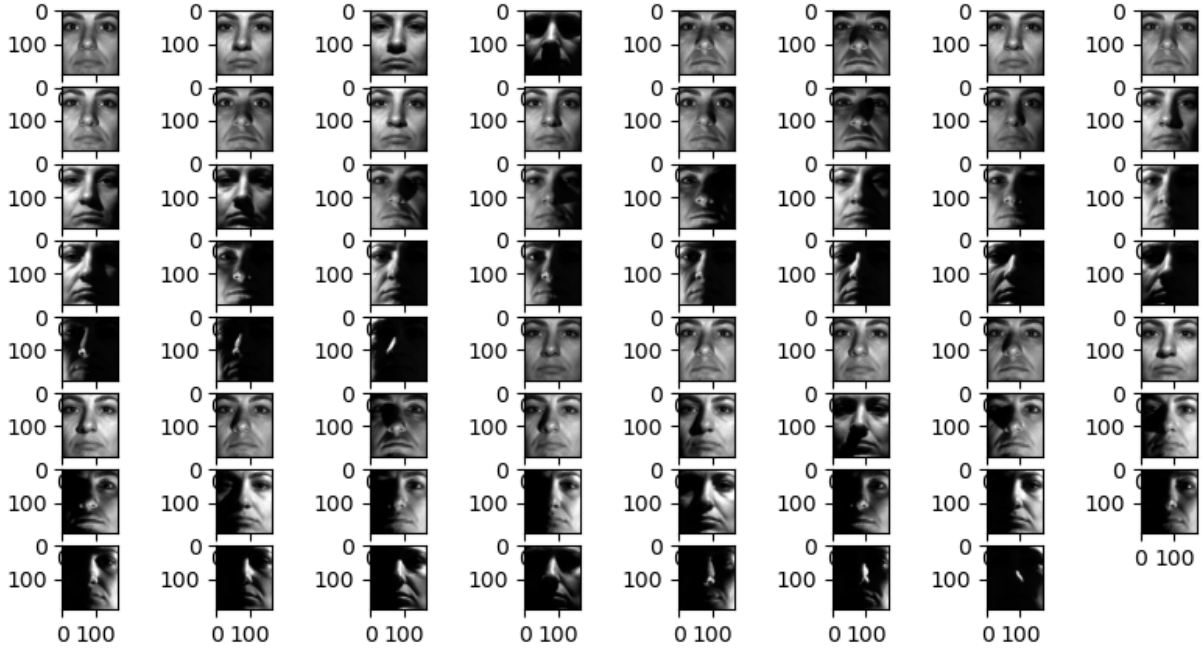
Person: 11



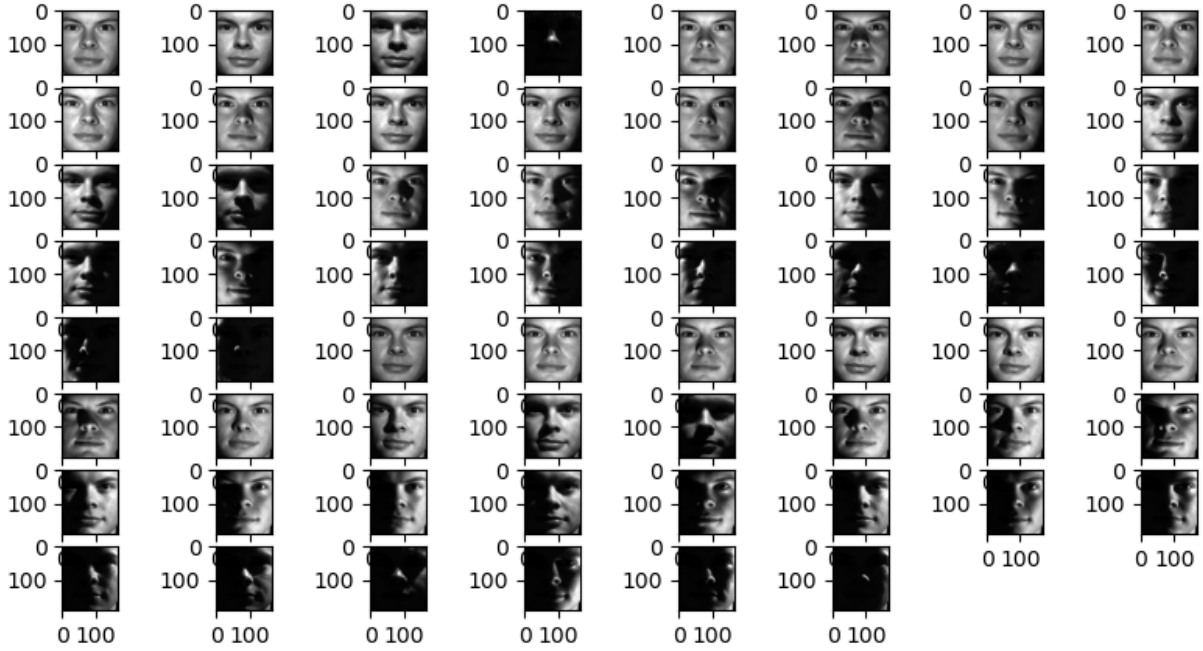
Person: 12



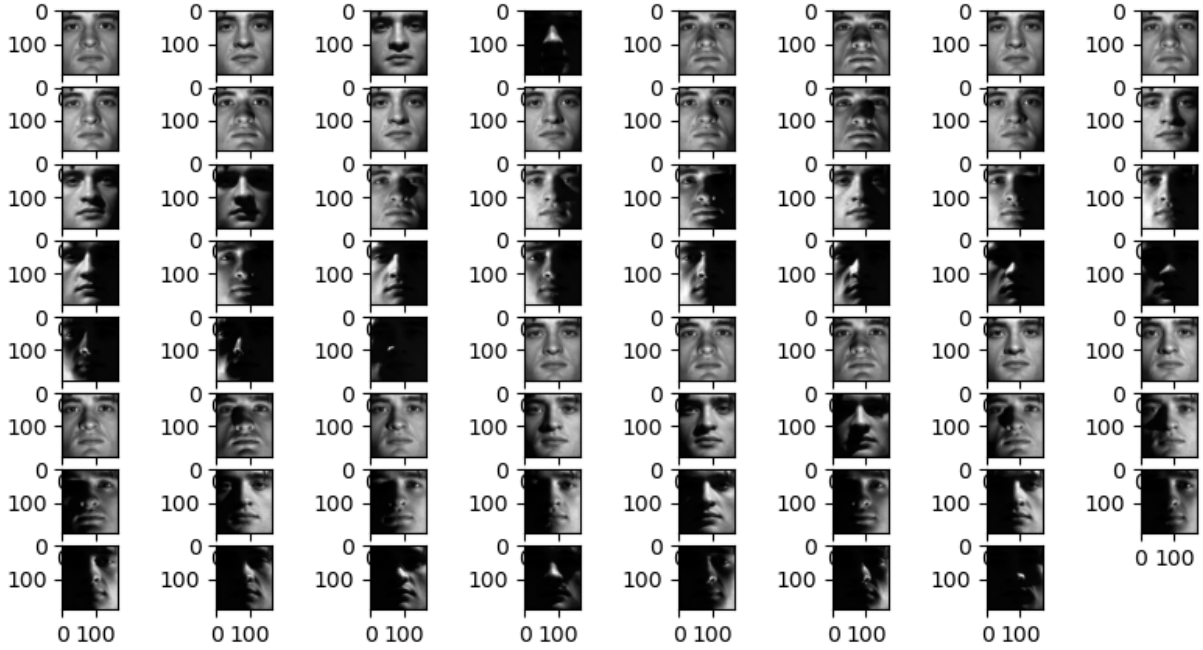
Person: 13



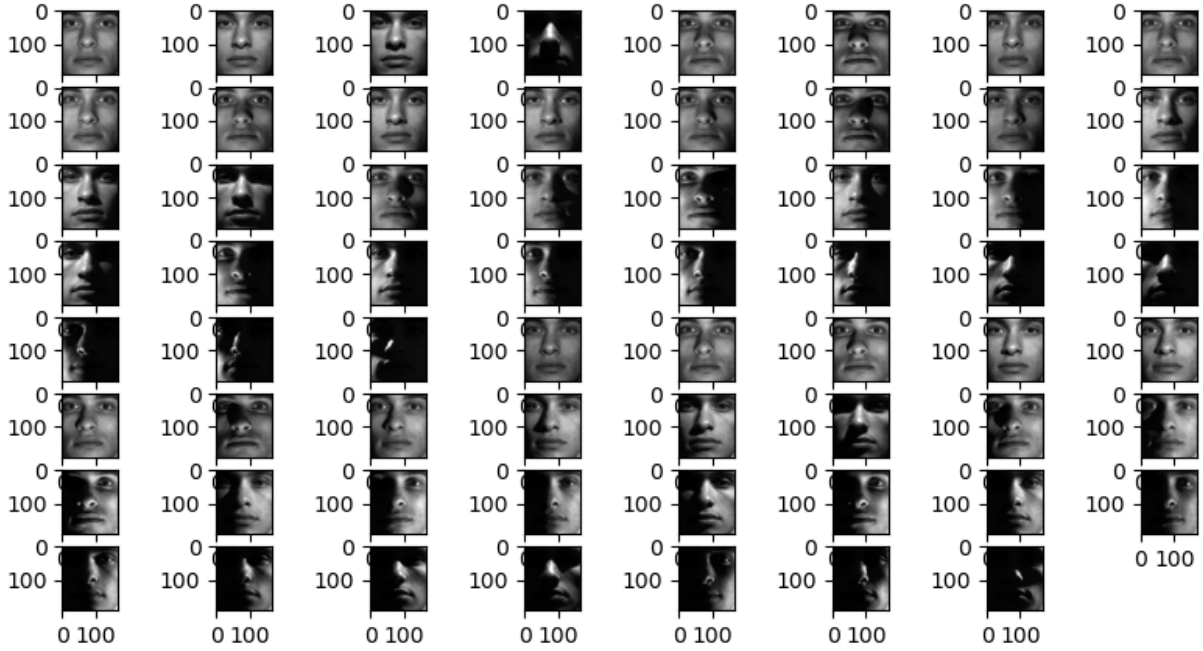
Person: 14



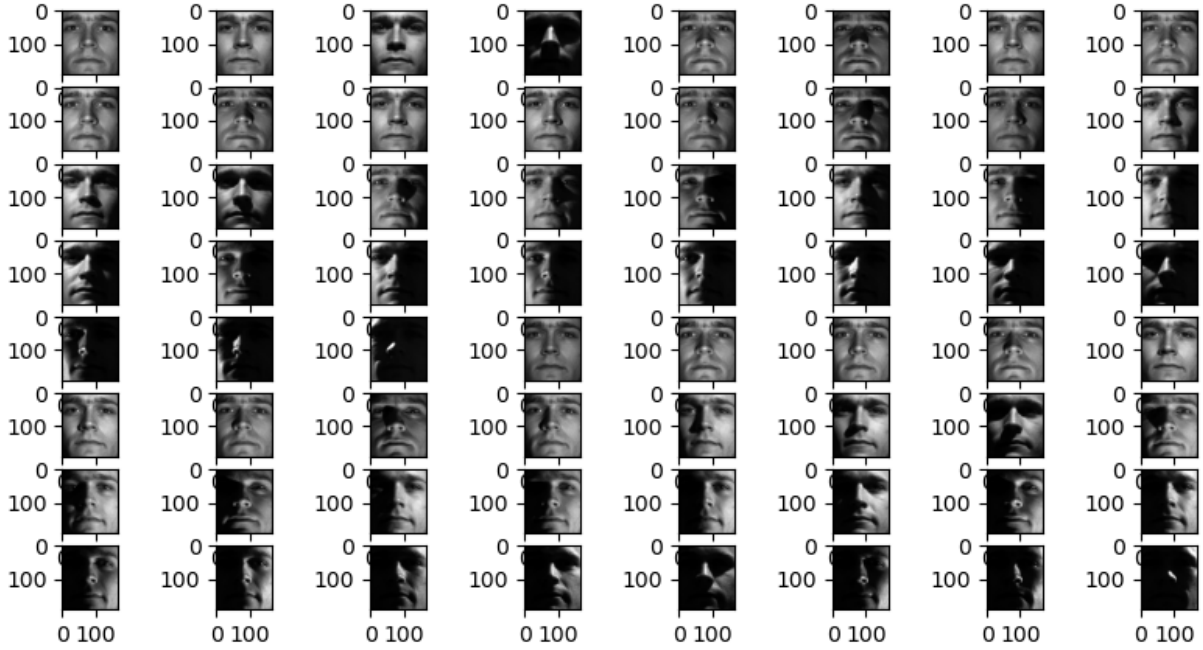
Person: 15



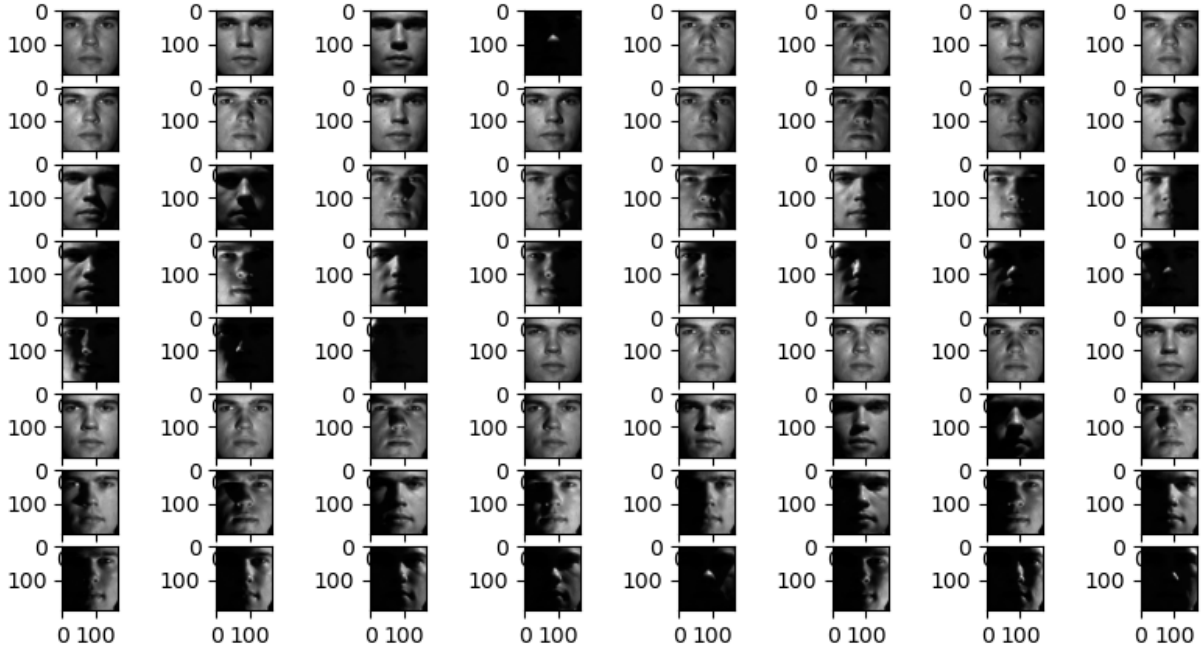
Person: 16



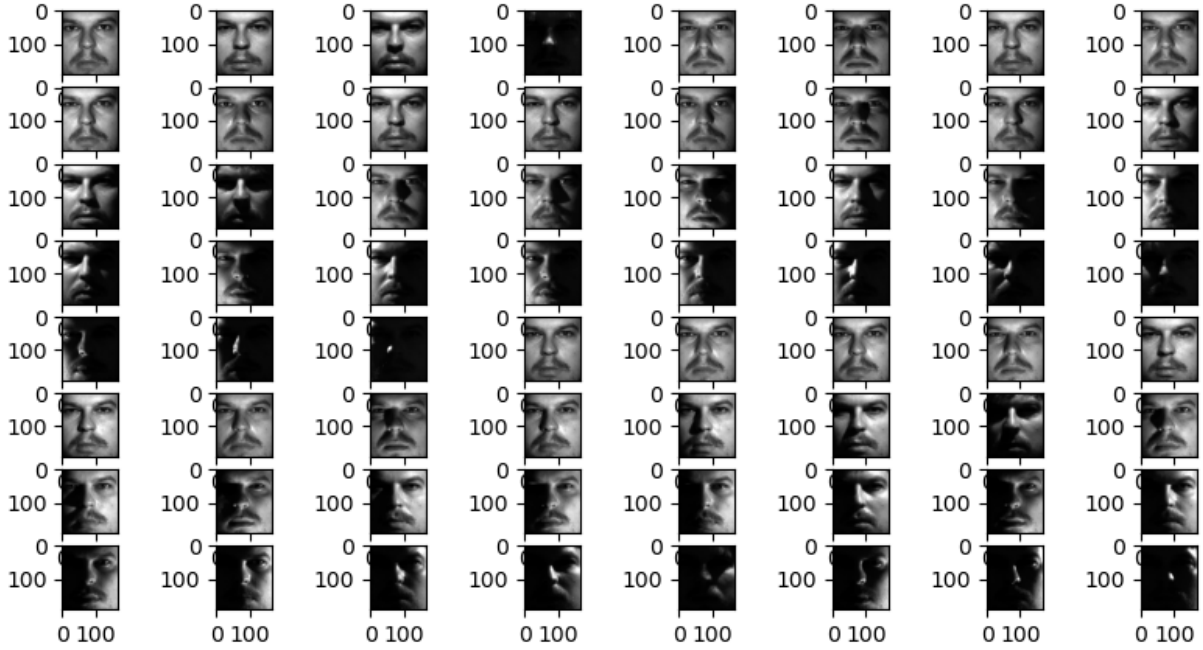
Person: 17



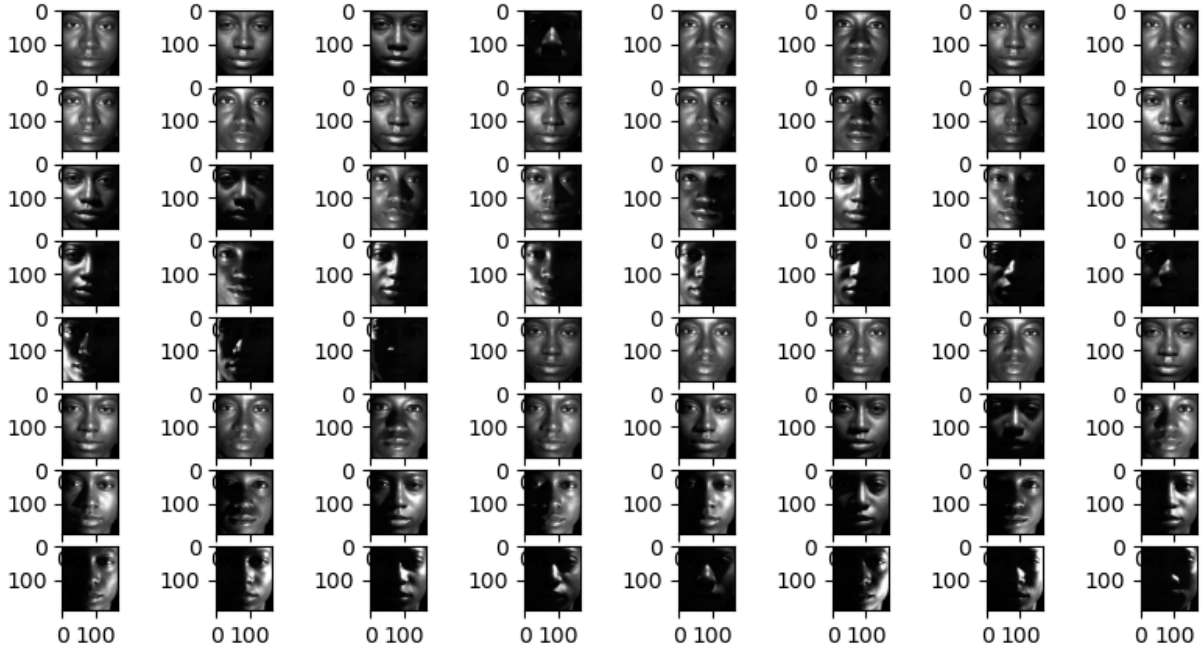
Person: 18



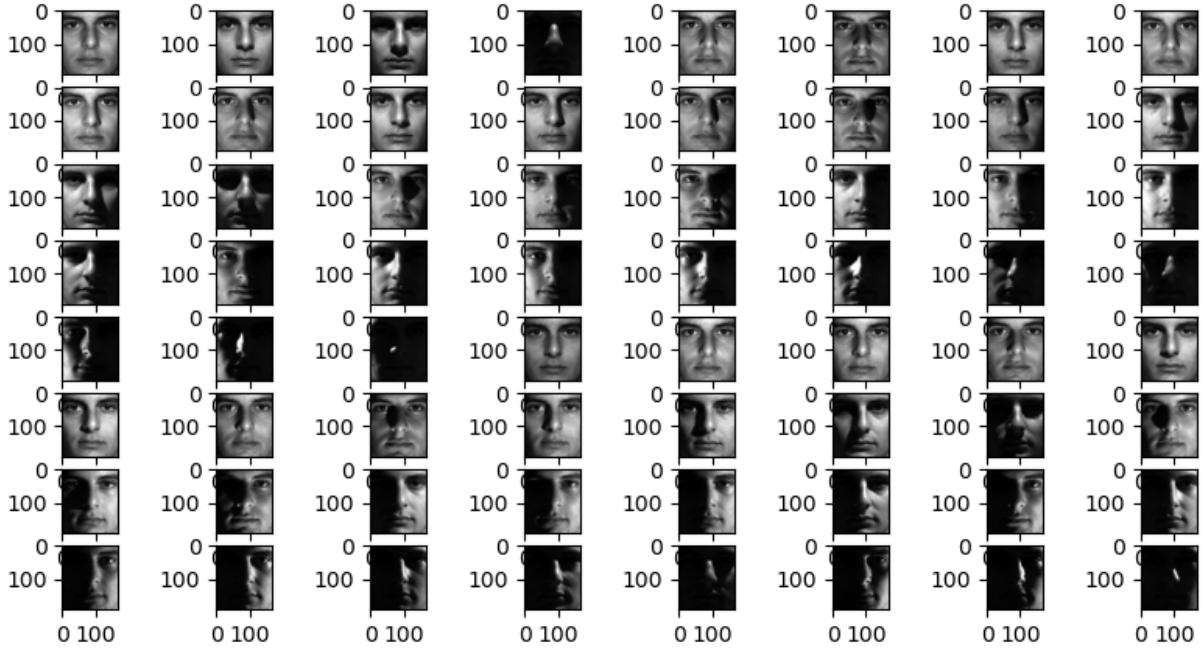
Person: 19



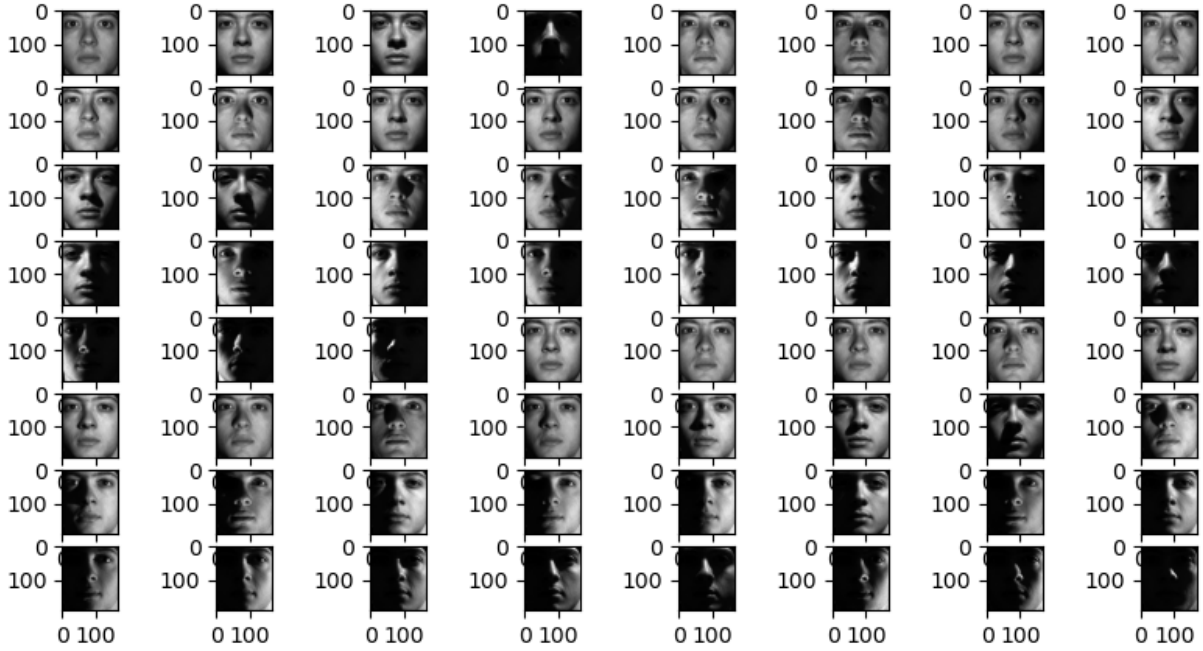
Person: 20



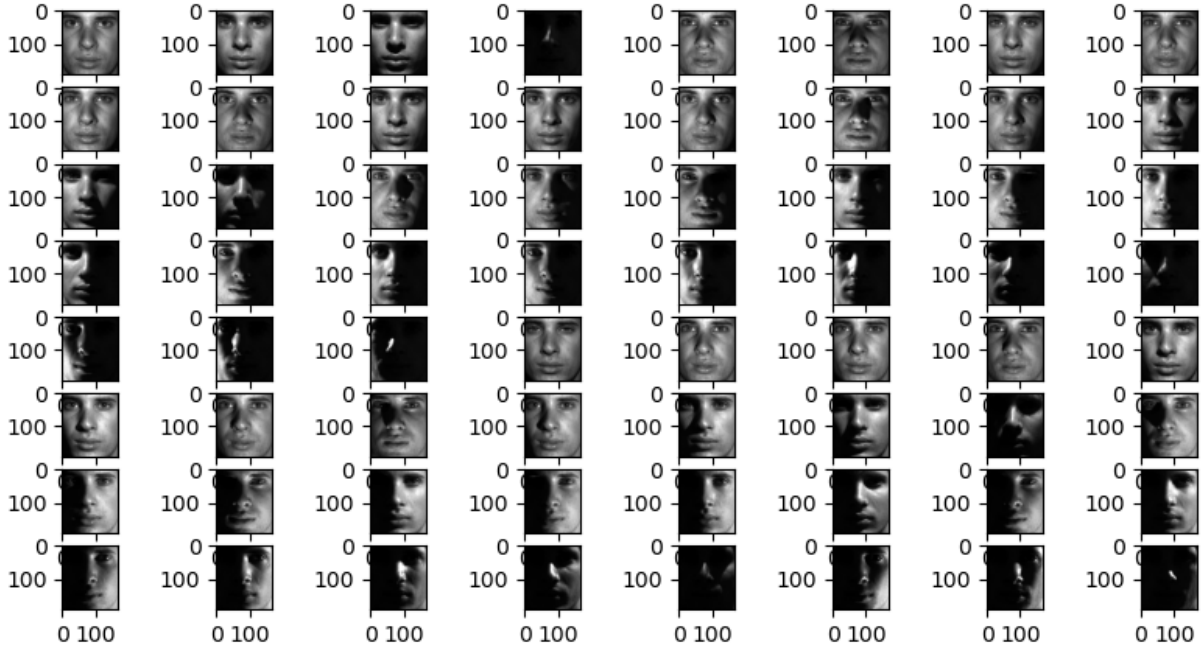
Person: 21



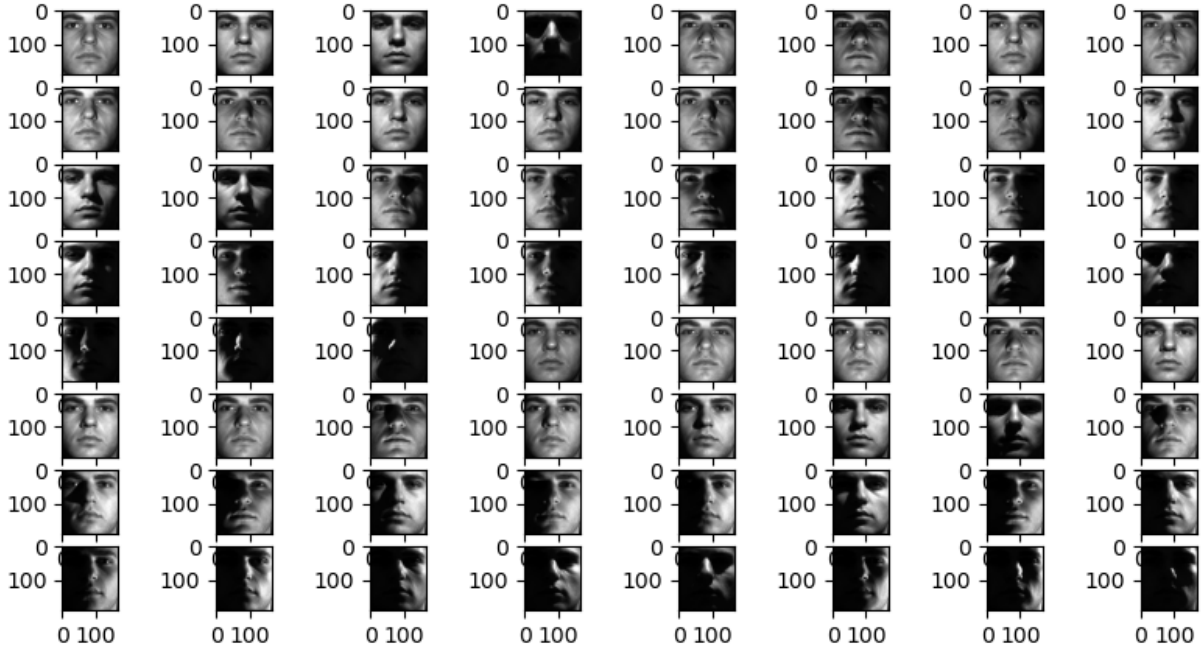
Person: 22



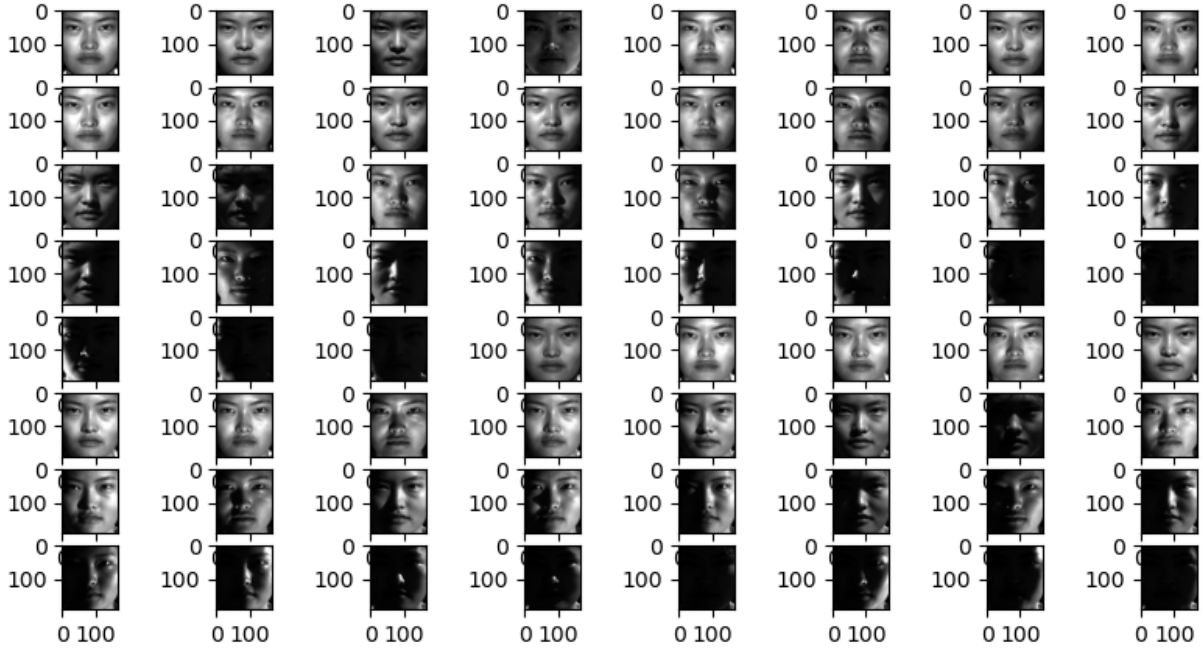
Person: 23



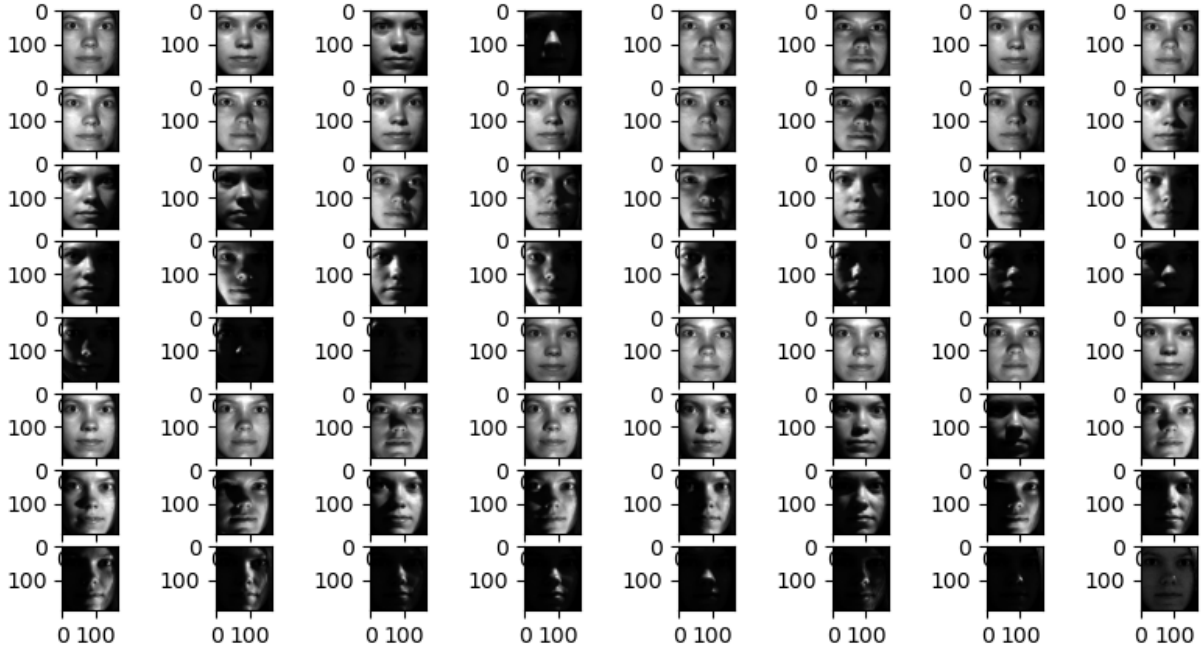
Person: 24



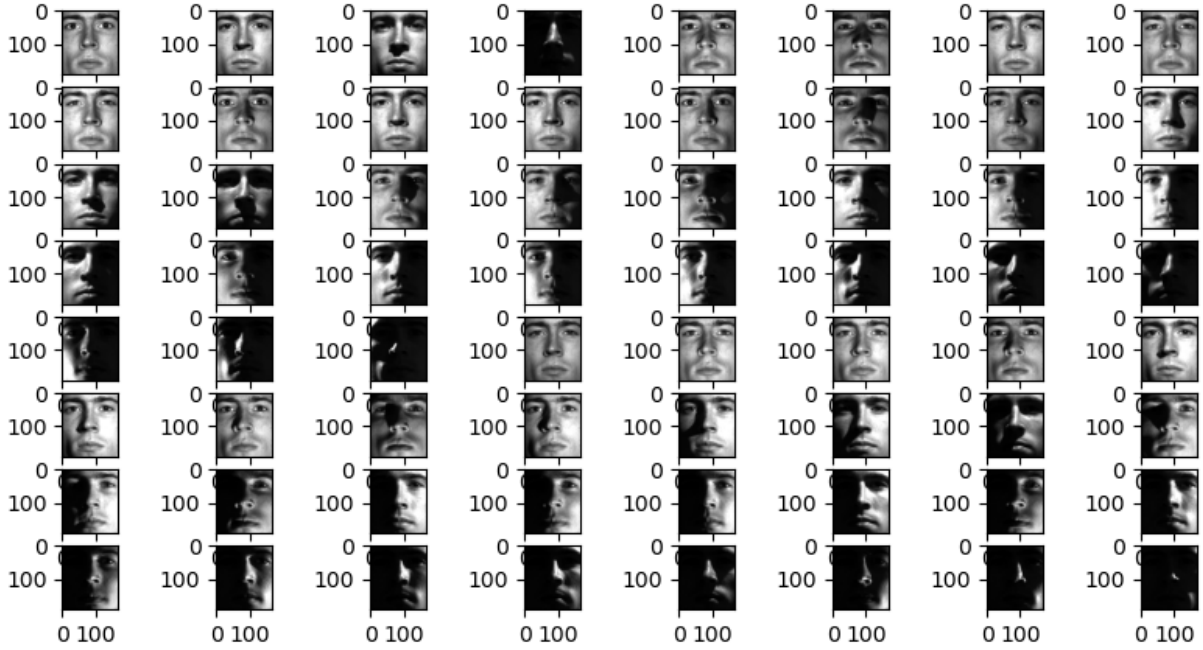
Person: 25



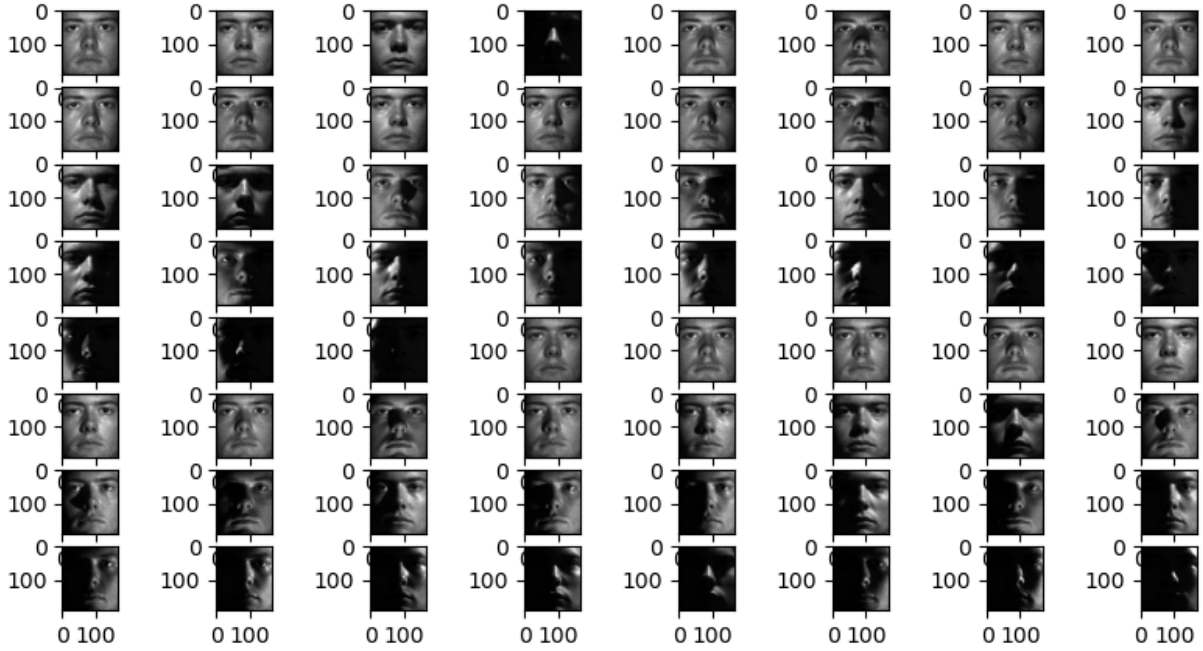
Person: 26



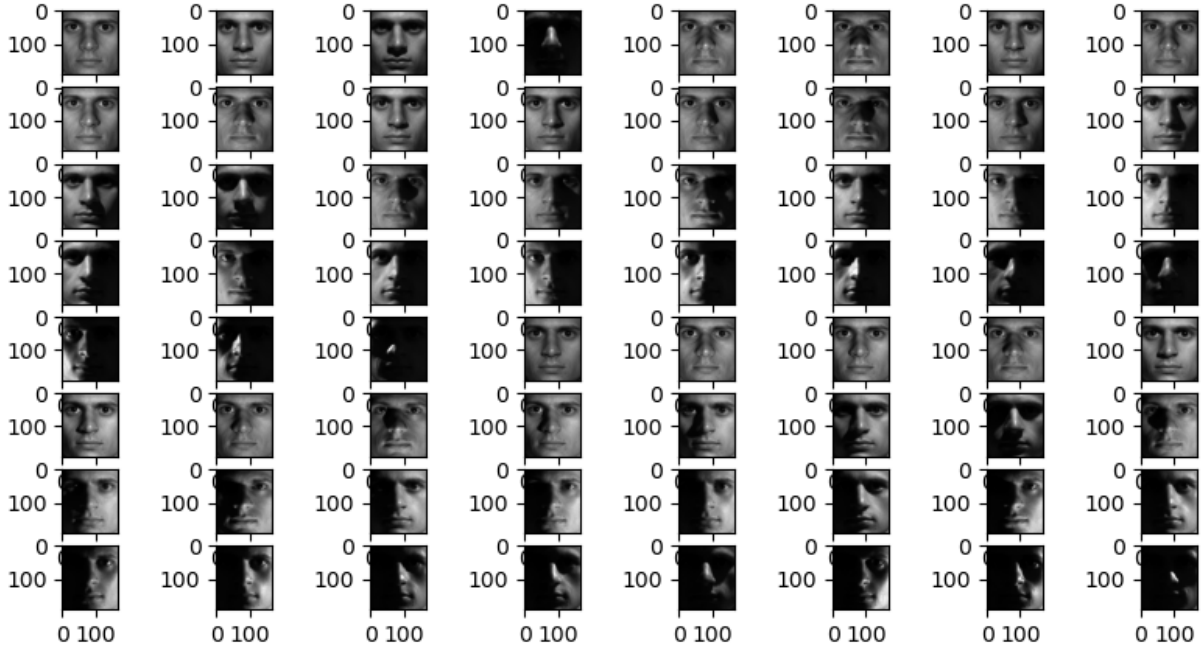
Person: 27



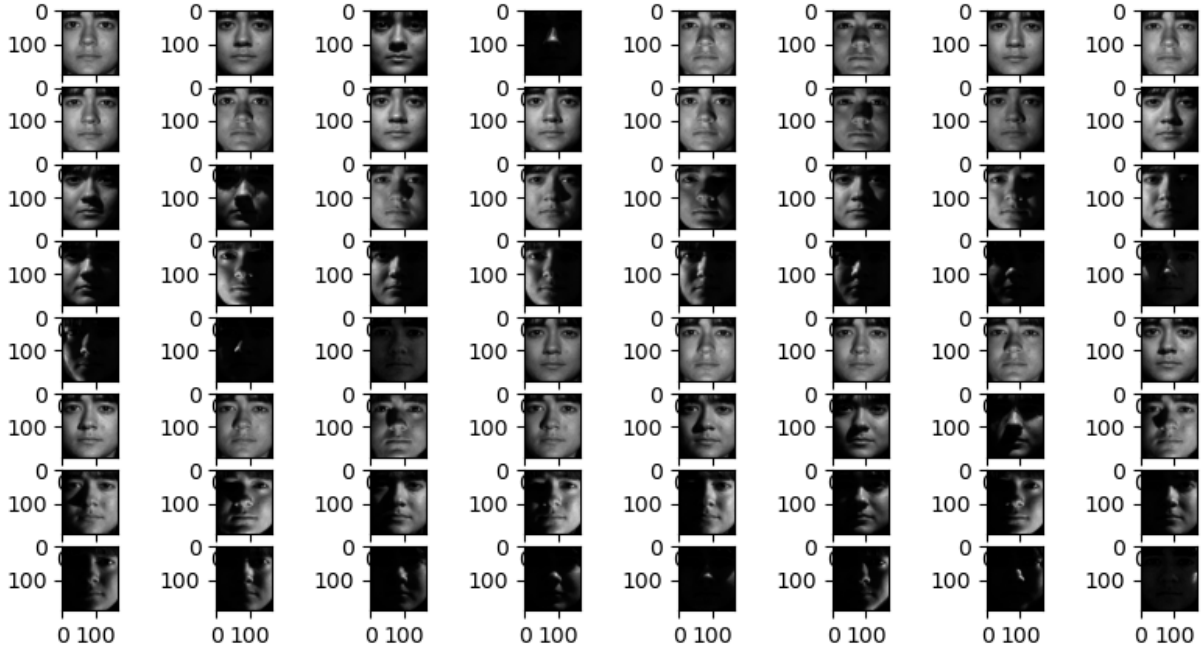
Person: 28



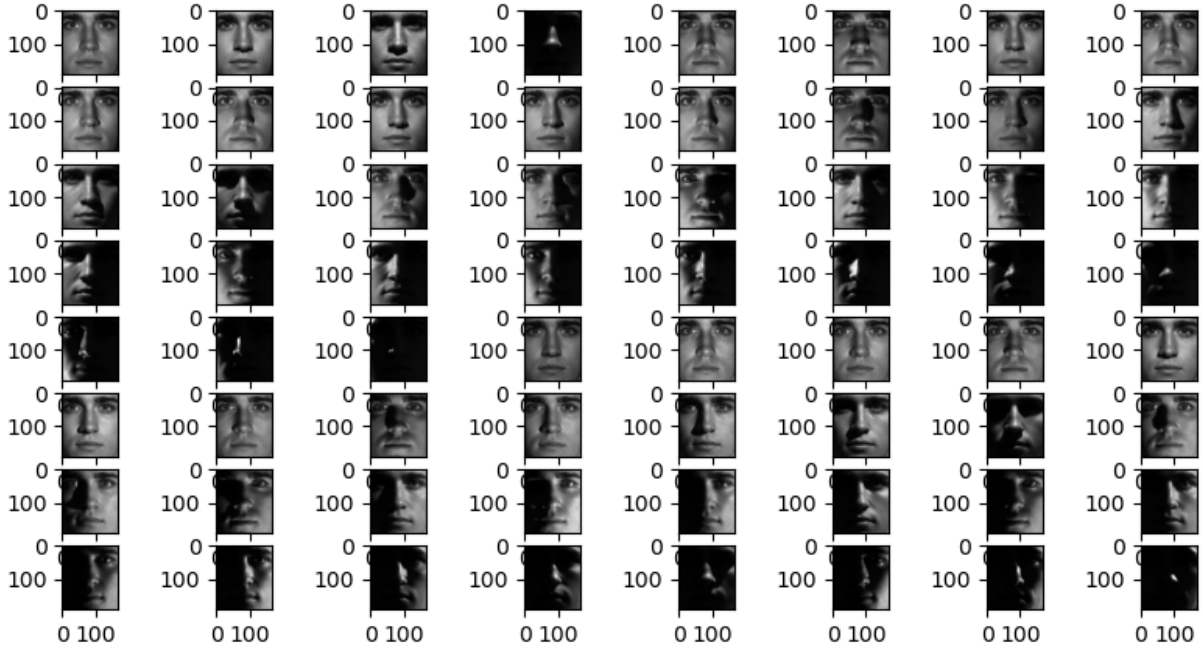
Person: 29



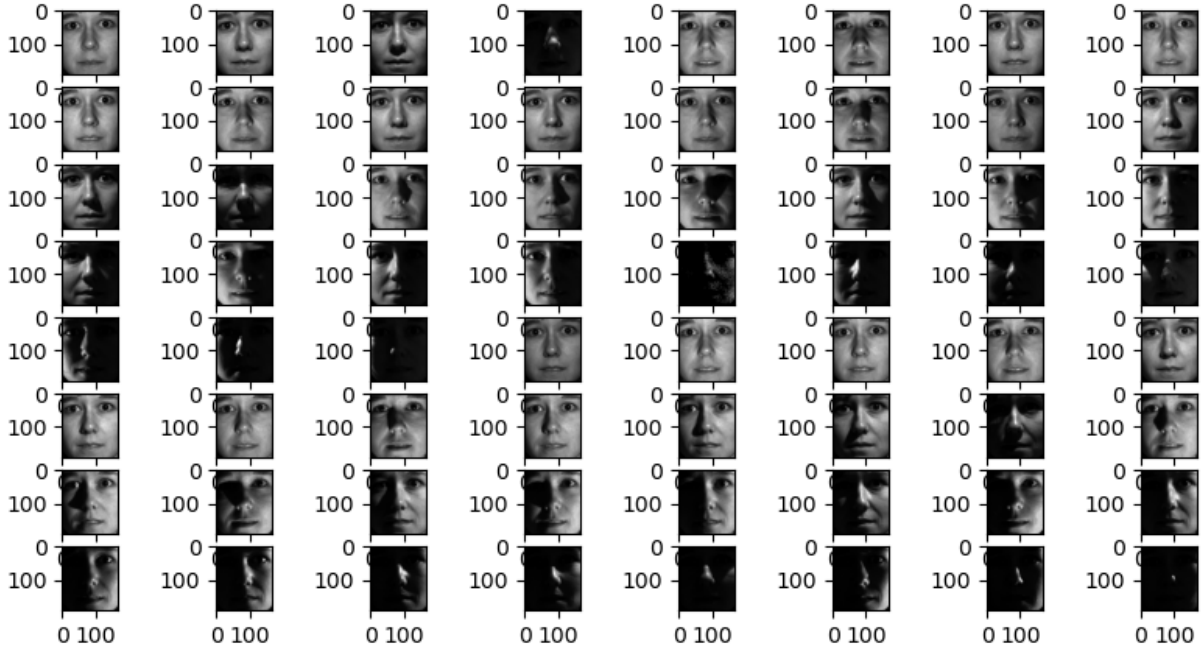
Person: 30



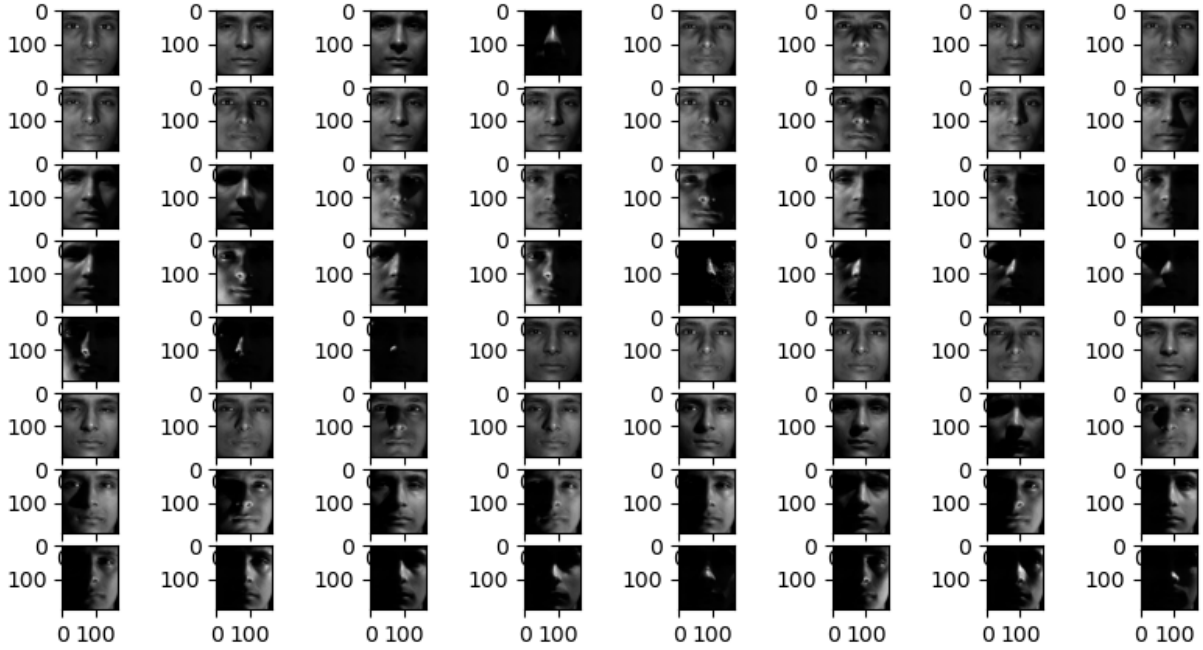
Person: 31



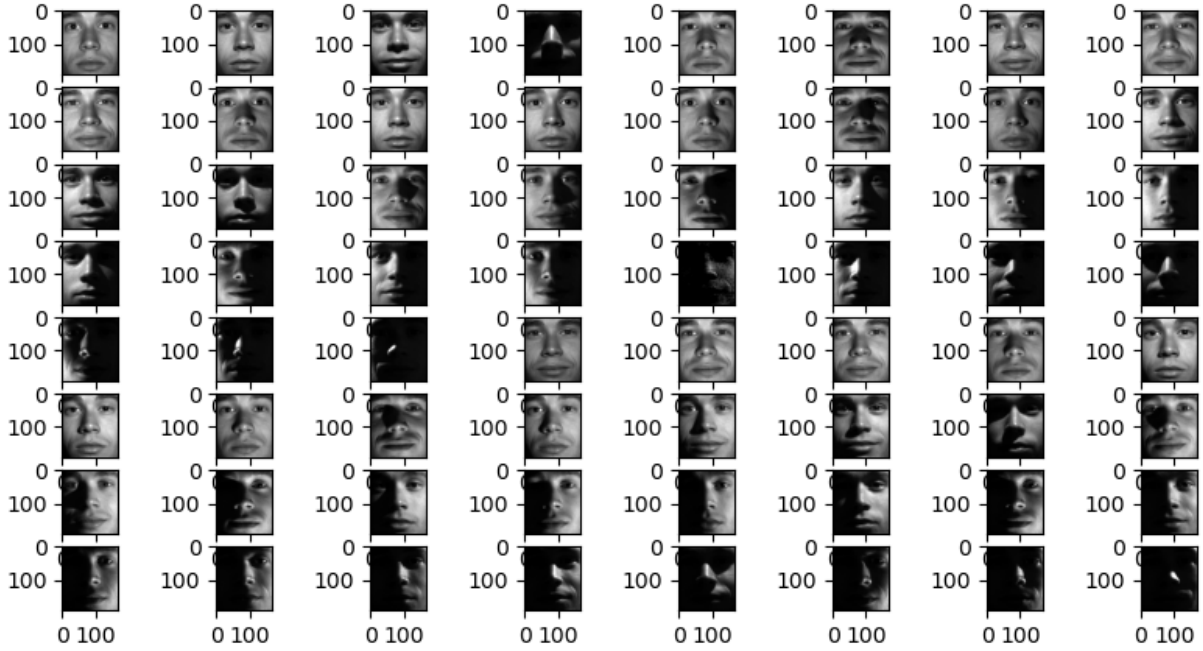
Person: 32



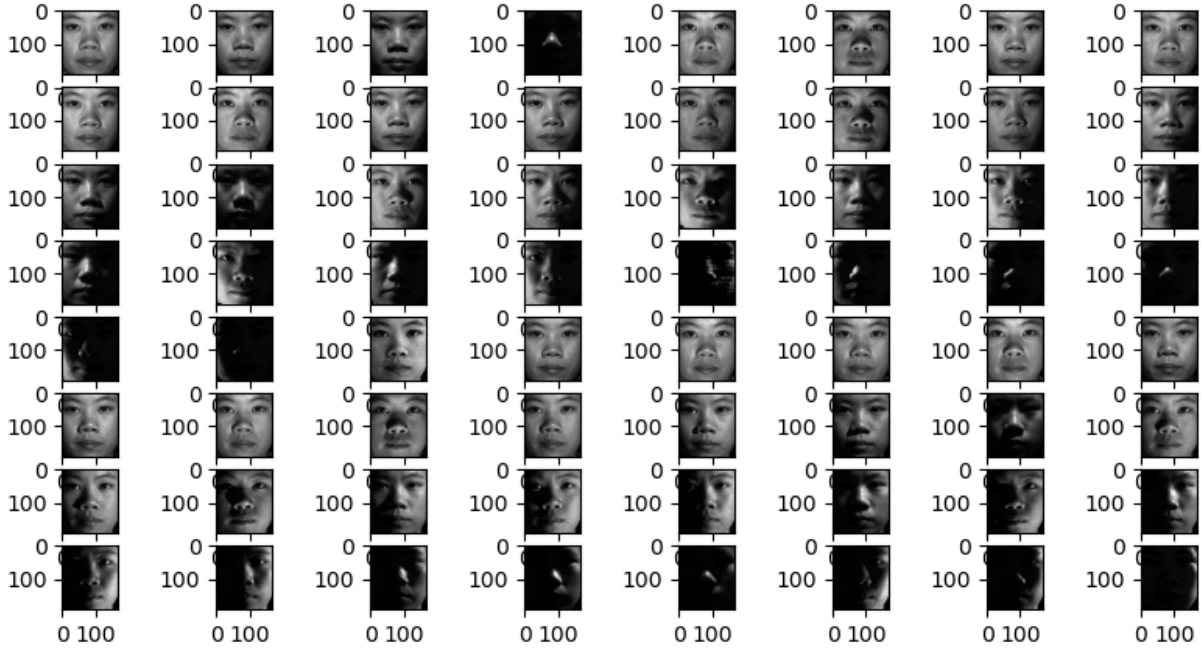
Person: 33



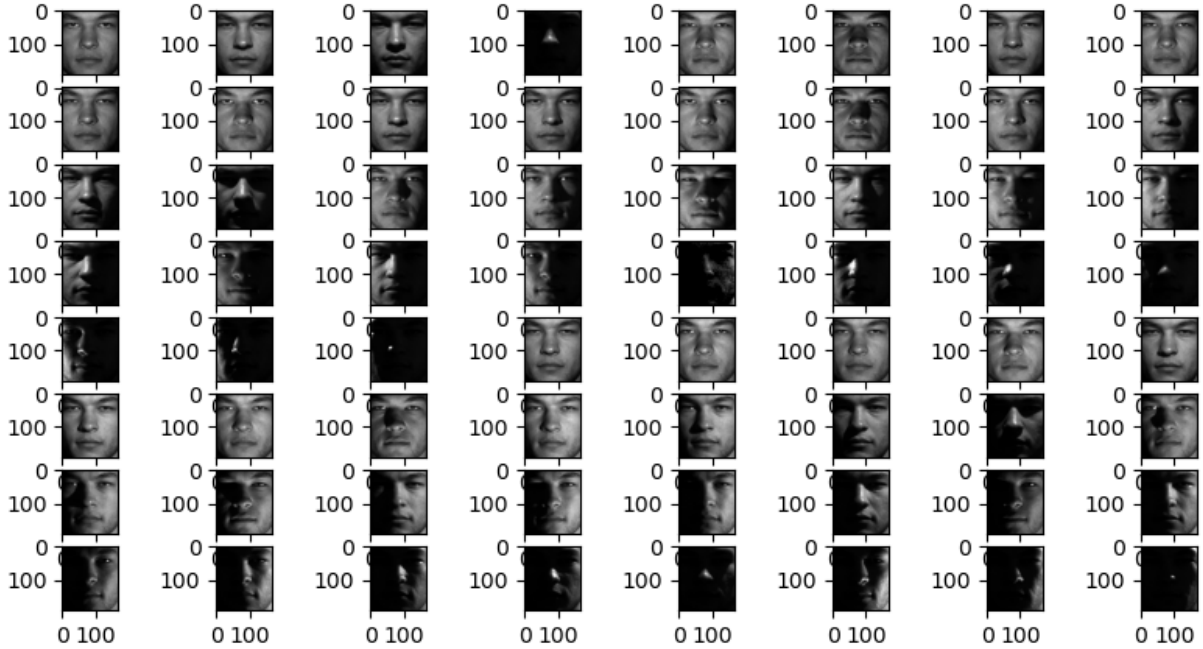
Person: 34



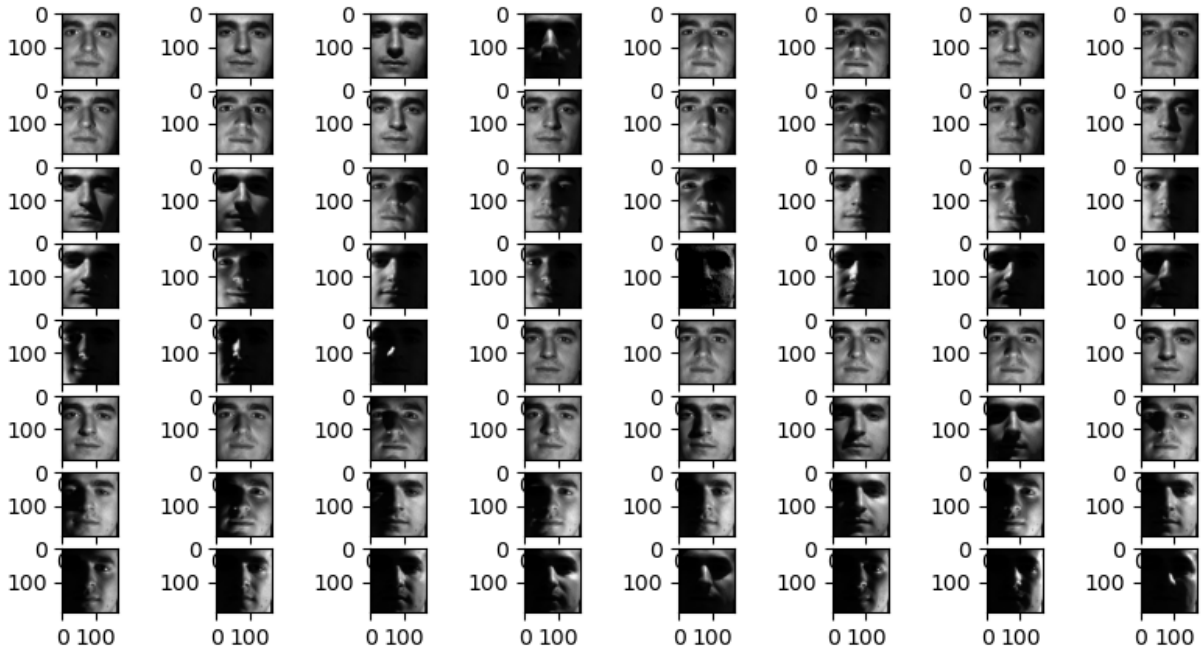
Person: 35



Person: 36



Person: 37

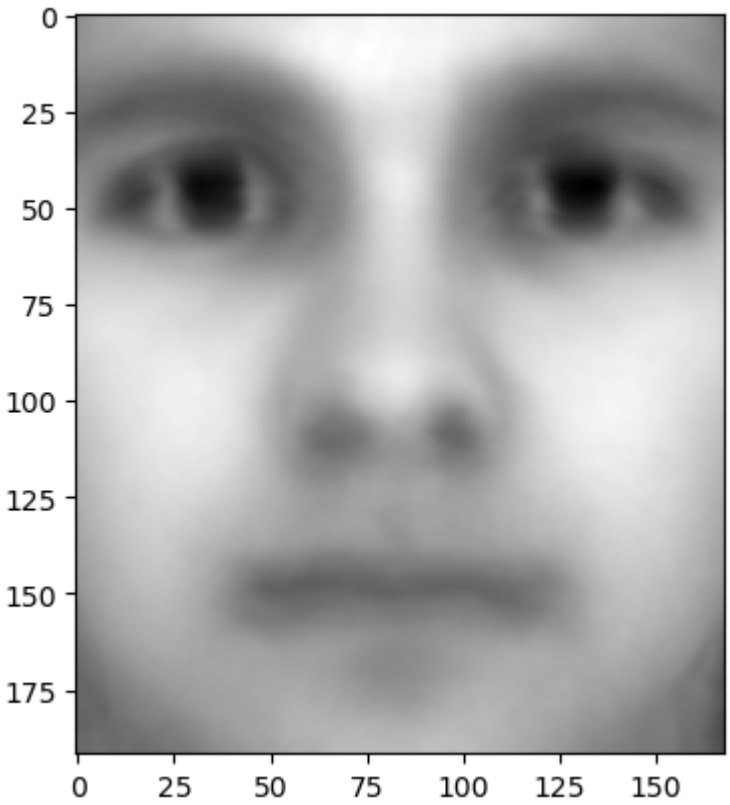


2. Load all faces with all different lightings from person # 1 to person # 36. Calculate the average face picture. Subtract that average from all pictures. Store these average subtracted faces in matrix X.

In [10]:
```
num = nfaces[0,0:36].sum()
```

In [11]:
```
avg = faces[0:num].mean(axis =0)
plt.imshow((avg.reshape(168,192).T), cmap='gray')
```

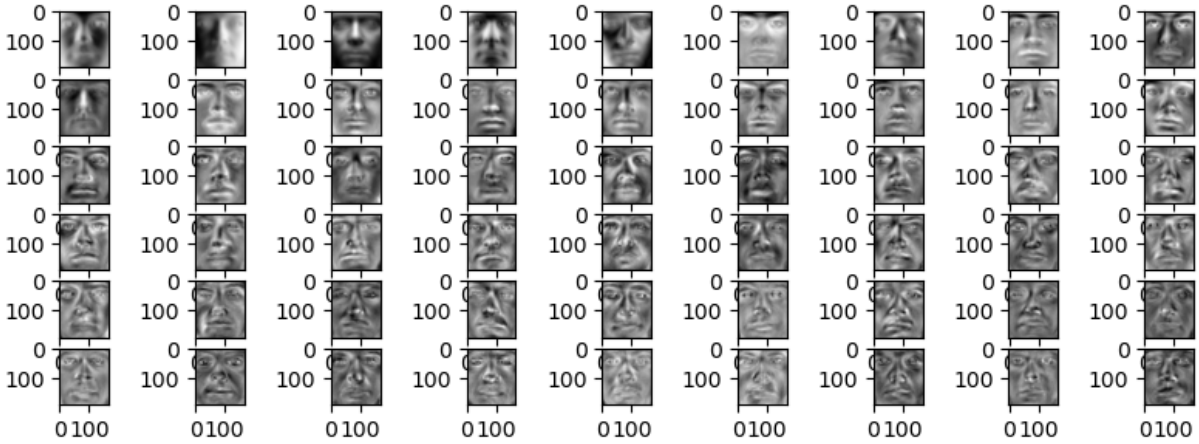
Out[11]:
```
<matplotlib.image.AxesImage at 0x7f2762eac6a0>
```



In [12]:
```
X = []
for j in range(len(faces[0:num])):
    sub = faces[j] - avg
    X.append(sub)

X = np.array(X)
```

In [13]:
```
U,S,V = svd(X.T)
```

In [14]:
```
plt.figure(figsize = [10,5])

for i in range(54):
    pos = i+1


    plt.subplot(9,9,pos)
    plt.imshow((U[:,i].reshape(168,192).T), cmap='gray')
```



> Every eigen face is searching for a distinct characteristic. The initial eigen faces attempt to identify characteristics found in every picture.

> The first and fifth eigen faces' dot product is 4.7704895589362195e-17, while the tenth and fifteenth eigen faces' dot product is 1.3010426069826053e-17. Thus, the 1st and 5th, as well as the 10th and 15th, are orthogonal.

In [15]:
```
print(U[0] @ U[4])
print(U[9] @ U[14])
```

```
4.597017211338539e-17
4.9439619065339e-17
```

4. Now let's decompose the first picture of person # 37 to these eigen faces.

In [16]:
```
person_37 = np.array(Global_Image_Dict[36])
person_37
```

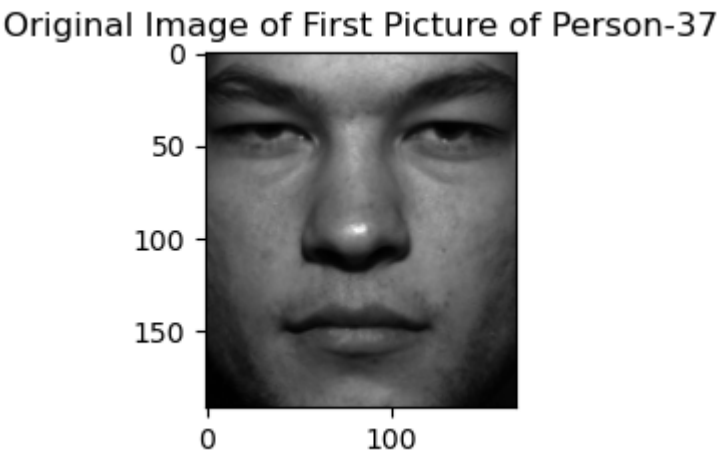
Out[16]:
```
array([[ 33,  33,  32, ...,  56,  59,  58],
       [ 30,  31,  32, ...,  10,   9,   9],
       [ 21,  23,  26, ...,  10,  11,  10],
       ...,
       [  3,   2,   2, ..., 134, 140, 144],
       [  2,   2,   2, ..., 162, 165, 164],
       [  3,   3,   3, ..., 103,  99, 103]], dtype=uint8)
```

In [17]:
```
person_37.shape
```

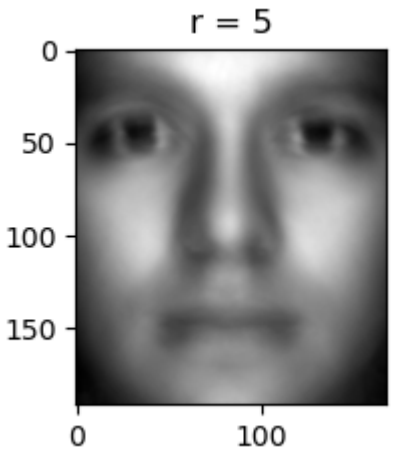
Out[17]:
```
(64, 32256)
```

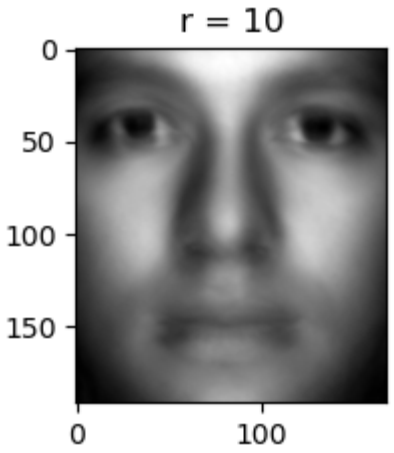
In [18]:
```
plt.figure(figsize=[2,3])

plt.imshow((person_37[1].reshape(168,192).T), cmap='gray')
plt.title('Original Image of First Picture of Person-37')
```

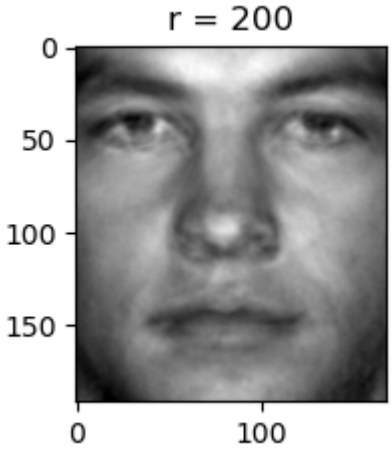
Out[18]:
```
Text(0.5, 1.0, 'Original Image of First Picture of Person-37')
```

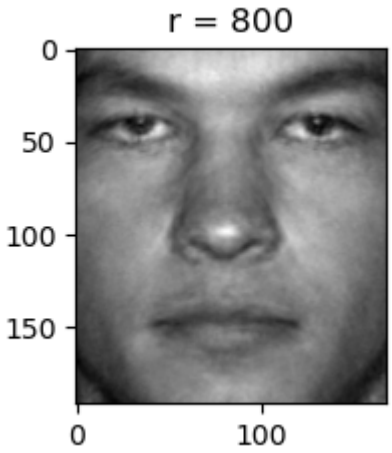


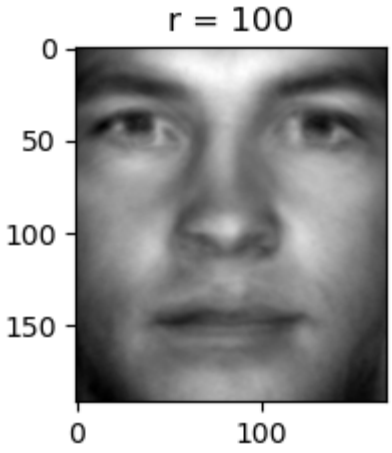
In [19]:
```
X_37 = person_37[0] - avg
r_list = [5, 10, 200, 800, 100]

for r in r_list:
    reconstructed = avg + U[:,:r]  @ (U[:,:r].T @ X_37)
    plt.figure(figsize=[2,3])

    img = plt.imshow(reconstructed.reshape((168,192)).T, cmap='gray')
    plt.title(f'r = {r}', )
    plt.show()
```











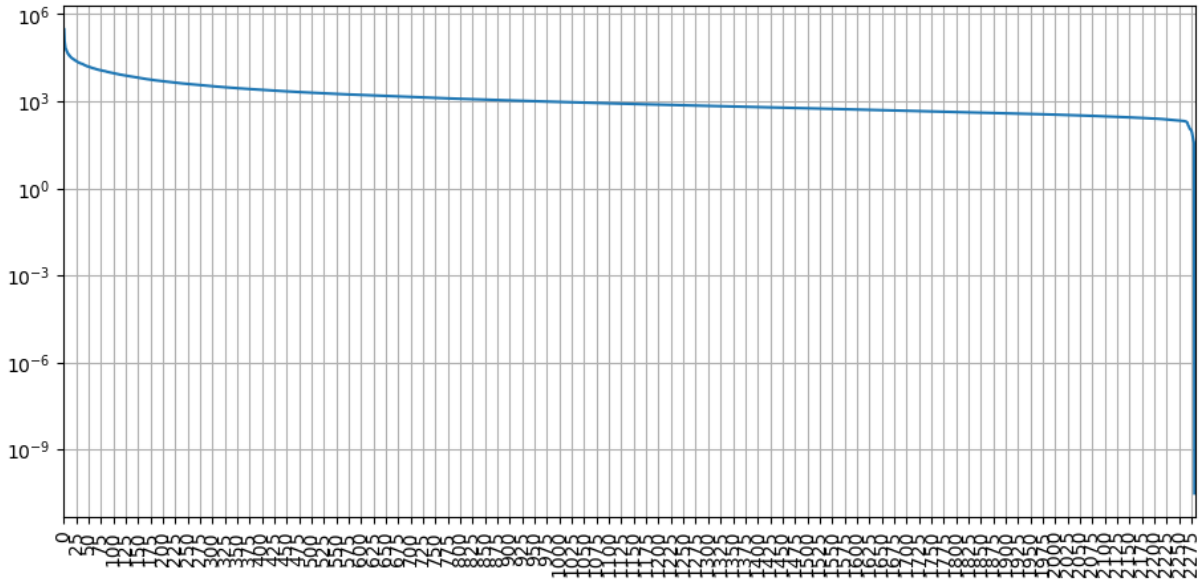
> **At r=100, the reconstructed face has good clarity. Thus, with r=100, it is possible to obtain a reasonable approximation of the image.**

5. Plot singular values (in a semi-logarithmic scale, horizontal axis representing the index, vertical axis representing the value of the singular value, vertical axis is scaled logarithmically). Do you see a good point for truncation?

> **The "elbows" or "knees" in the singular value distribution, which may indicate the change from singular values that reflect noise to those that represent significant patterns, are a common way to determine the truncation value. However, this is ineffective. The majority of the time, the plot does not make a clear change that would reveal the elbow.

The singular value plot that is shown below demonstrates this as well. Thus, it is not possible to select a good truncation point from this figure.**

In [20]:
```
plt.figure(figsize = [11,5])
plt.semilogy(S)
plt.xlim([0, 2282])
plt.xticks(np.arange(0,2282,25), rotation ='vertical')
plt.grid()
```



6. Plot a 2-dimensional graph in which the horizontal axis is the 5th eigen face and vertical axis is the 6th eigen face.

In [21]:
```
P1 = Global_Image_Dict[1]
P2 = Global_Image_Dict[6]

X1 = []
for j in range(P1.shape[0]):
    sub = P1[j] - avg
    X1.append(sub)

X2 = []
for j in range(P2.shape[0]):
    sub = P2[j] - avg
    X2.append(sub)


eigen_faces = [5, 6] # Project onto PCA modes 5 and 6
```

```
PCACoordsP1 = U[:,eigen_faces-np.ones_like(eigen_faces)].T @ P1.T
PCACoordsP2 = U[:,eigen_faces-np.ones_like(eigen_faces)].T @ P2.T


plt.figure(figsize = [5,4])

plt.plot(PCACoordsP1[0,:],PCACoordsP1[1,:],'d',color='k',label='Person 2')
plt.plot(PCACoordsP2[0,:],PCACoordsP2[1,:],'^',color='r',label='Person 7')

plt.legend()
plt.grid()
plt.xlabel('PCA5')
plt.ylabel('PCA6')
plt.show()
```