Face Recognition using PCA

Import some libraries

In [ ]:
```
import os
import cv2
import random
import matplotlib.pyplot as plt
import numpy as np
import  matplotlib.image as mpimg
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

Part 1 - Visualizing the Face Images

1. Visualize randomly selected 16 faces in a 4×4 grid [4 rows and 4 columns].

In [ ]:
```
fig = plt.figure(figsize=(15, 15)) # Creates the canvas and specifies the size
col = 4
row = 4
folder = 'face_data/'
index = 1
dirnames = random.sample(os.listdir(folder), 16) # Randomly selected 16
for dirname in dirnames:
    # Find the file path
    filename = random.choice(os.listdir(folder + dirname))
    path = '%s/%s/%s' % (folder, dirname, filename)
    # Open the file
    with open(path, 'rb') as image:
        img = cv2.imdecode(np.frombuffer(image.read(), np.uint8), cv2.IMREAD_GRAYSCALE)
    fig.add_subplot(row, col, index)
    plt.imshow(img, cmap="gray")
    index = index + 1
plt.show()
```

2. Report the face image size, number of images and number of classes.

In [ ]:
```
print("The image size: " + str(img.shape))

list_class = os.listdir(folder)
print("Number of classes: " + str(len(list_class)))

number_each_class = len(folder)
number_image = len(list_class) * number_each_class
print("Number of images: " + str(number_image))
```

```
The image size: (112, 92)
Number of classes: 40
Number of images: 400
```

Part 2 - Train Test Split

1. Generate Train and Test set from the dataset in the following manner.

2. Select the first nine images of each subject for the Train set.

3. Select the last image of each subject for the Test set.

4. Flatten each image into 1D vector so that the dataset size is N ×L, where N is the number of samples in the train/test set and L is the length of flattened image (L = 92 ×112 = 10304)

5. Report the number of images in Train set and Test set.

In [ ]:
```
trainset = np.zeros((40 * 9, 112 * 92)) # Store the 1D data of the trainset, the image size is
trainset_number = np.zeros(40 * 9).astype(np.int8) # Store the index of trainset
testset = np.zeros((40 * 1, 112 * 92)) # Store the 1D data of the testset
testset_number = np.zeros(40 * 1).astype(np.int8) # Store the index of testset

original_train = [] # The original data of the trainset
original_test = [] # The original data of the testset

for i in range(1, 41):
    people_num = i
    for j in range(1, 11):
        # Select the first nine images of each subject for the Train set
        if j <= 9:
            filename = folder+'/s'+str(people_num)+'/'+str(j)+'.pgm'
            img = mpimg.imread(filename)
            original_train.append(img) # Store original data
            img = img.flatten().astype(np.float) # Flatten each image into 1D vector
            trainset[(i - 1) * 9 + (j - 1),:] = img
            trainset_number[(i - 1) * 9 + (j - 1)] = people_num
        # Select the last image of each subject for the Test set
        else:
            filename = folder+'/s'+str(people_num)+'/'+str(j)+'.pgm'
            img = mpimg.imread(filename)
            original_test.append(img) # Store original data
            img = img.flatten().astype(np.float) # Flatten each image into 1D vector
            testset[(i - 1) * 1 + (j - 9) - 1,:] = img
            testset_number[(i - 1) * 1 + (j - 9) - 1] = people_num

# The number of images in train set and test set
print("Number of images in Train set: " + str(len(trainset_number)))
print("Number of images in Test set: " + str(len(testset_number)))
print("Length of flattened image: " + str(len(trainset[0])))
```

```
Number of images in Train set: 360
Number of images in Test set: 40
Length of flattened image: 10304
```

Part 3 - Apply PCA to Get Eigenfaces

1. Apply PCA using scikit-learn on the Train set

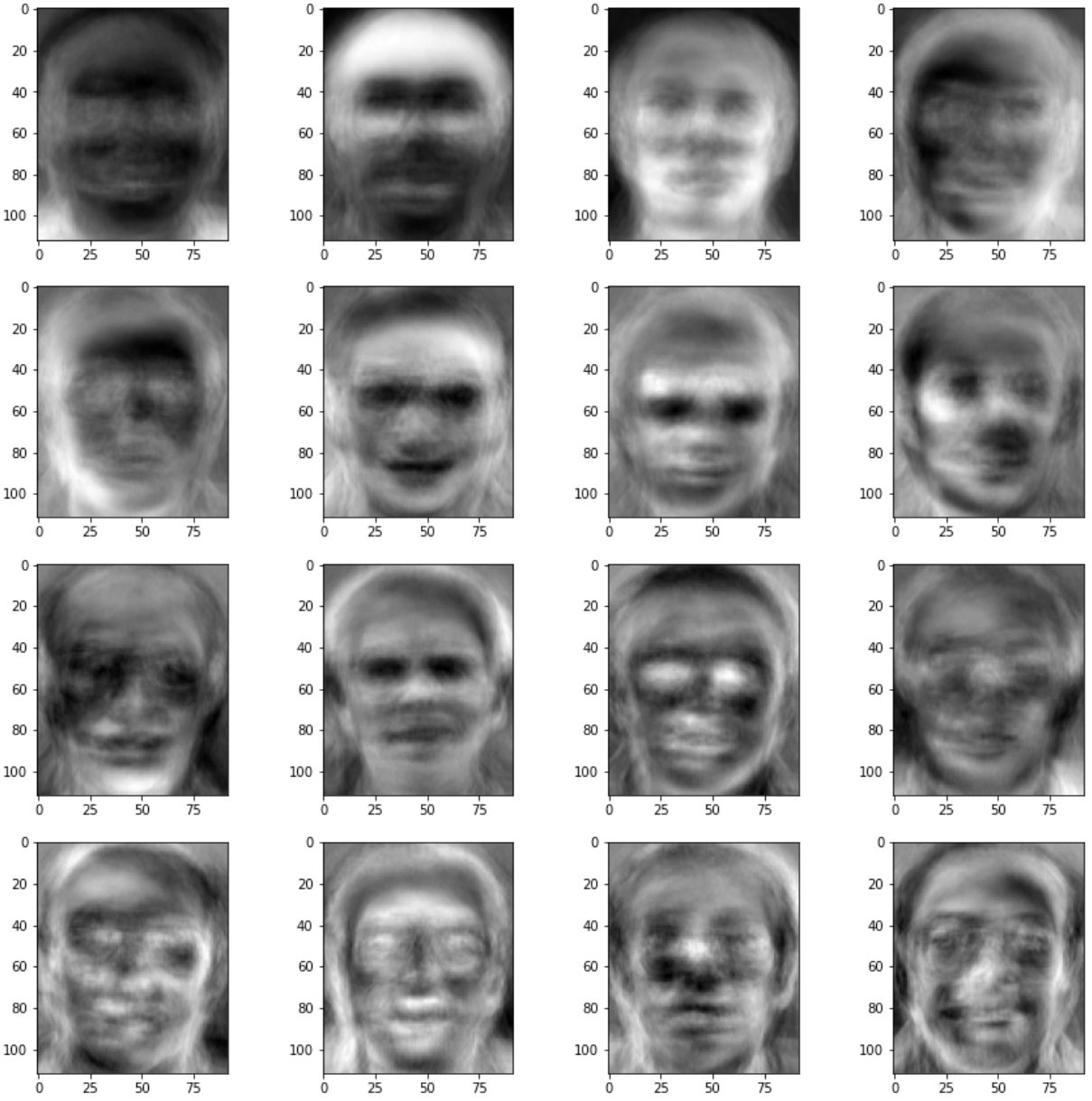
2. Take first 20 principal components in the feature space. These are known as eigenfaces.

3. Visualize the first 16 eigenfaces. For visualization, reshape the flattened vector to original image shape.

In [ ]:
```
scaler = StandardScaler()
trainset = scaler.fit_transform(trainset)
testset = scaler.transform(testset)

n_components = 20 # Take first 20 principal components
print(
    "Extracting the top %d eigenfaces from %d faces" % (n_components, trainset.shape[0]))
# Apply PCA using scikit-learn on the Train set
```

```
pca = PCA(n_components=n_components, svd_solver="randomized", whiten=True).fit(trainset)

eigenfaces = pca.components_.reshape((n_components, 112, 92)) # Reshape the flattened vector to

# Visualize the first 16 eigenfaces
fig = plt.figure(figsize=(15, 15))
for i in range(1, 17):
    img = eigenfaces[i - 1]
    fig.add_subplot(row, col, i)
    plt.imshow(img, cmap="gray")
plt.show()
```

```
Extracting the top 20 eigenfaces from 360 faces
```



Part 4 - Face Recognition

1. Calculate the weights of the training samples using the given formula

In [ ]:
```
train_sample_weights = np.matmul(pca.components_, (trainset - pca.mean_).T)
print(train_sample_weights)
```

```
[[-45.62468198 -75.76333343 -57.22871554 ... -45.85413572 -31.34342282
  -51.772056  ]
 [ -0.56960128  28.36150174  30.36251114 ...  -9.23509151  -7.05075694
   -5.23600699]
 [ 58.33641633  -1.15203468  37.29814071 ... -29.0886094  -39.42523483
  -21.44701117]
 ...
 [-14.94270909  -7.70848003  -1.08623953 ...  -1.5245741  -11.53347257
  -11.4464049 ]
 [  4.03229223  -3.12676997   8.98164922 ...  12.53255706   5.78079654
   -7.67564984]
 [  5.80716919  22.12090272  10.73370916 ...   2.30615288 -13.39734207
    7.40929429]]
```

2. For each test image, calculate weights similarly

In [ ]:
```
test_sample_weights = np.matmul(pca.components_, (testset - pca.mean_).T)
print(test_sample_weights)
```

```
[[-6.08550014e+01 -6.96755417e+00 -2.66122854e+01 -2.40322159e+01
  -7.13330573e+01 -7.51722349e+01 -2.78150702e+01 -3.88964533e+01
  -1.27082721e+01 -3.33397430e+01  3.59624622e+01 -4.87352153e+01
  -4.80122390e+01  5.40599119e+01 -5.81428790e-01 -1.46921500e+01
  -1.49981979e+01 -7.63541036e+01  3.01124461e+00  4.68219113e+01
   2.05669491e+01  8.80687964e+01 -1.75987917e+01  3.26481082e+01
  -3.21700594e+01 -2.85970043e+01  9.33548076e+00  1.33867183e+00
   6.57348240e+01  3.35201724e+01  3.32507374e+01 -1.47172027e+01
   7.70796808e+01  8.07546078e+01 -3.51351788e+01 -1.08948703e+01
  -1.69903359e+01  1.24319999e+01  8.48309625e+01 -2.06637955e+01]
 [ 2.44078356e+01  3.63205157e+01 -7.28525114e+00 -2.31539742e+01
  -5.32964078e+00 -7.14885858e+01 -9.58101602e+00 -2.07780880e+01
  -3.64706824e+01 -3.84429459e+01 -9.18372504e-01 -7.05502685e+00
   2.67135539e+01  5.59606248e+01 -1.29146547e+01 -1.53606288e+01
  -3.98667041e+01 -8.86084491e+00  5.21126178e+01 -4.16540030e+01
  -3.24484653e+01  3.10633066e+01 -4.34516300e+01 -1.51343764e+01
  -1.01848417e+01  8.32965106e+00  4.52933336e+01  6.23984282e+01
  -1.87161732e+01 -3.96361885e+01  3.25772818e+01  2.77127748e+01
  -2.68333271e+01  2.50397090e+01 -1.33320613e+00  4.93385357e+00
   4.44759591e+01 -4.41750153e+01 -5.61068803e+00 -1.98526442e+00]
 [ 2.85417289e+01  2.85277626e+01 -9.14135172e-01  3.59155636e+00
  -1.85844834e+01 -1.66725903e+01  1.75257449e+01  5.47838289e+01
  -1.40031616e+01  4.01165276e+01  3.68989511e+01  1.07471811e+00
  -2.90260321e+01 -2.93297695e+00 -2.87835360e+01  4.62752992e+01
   4.05926346e+01 -1.51017626e+01  2.54283982e+01 -5.07450846e+00
  -2.19260733e+01 -1.28403517e+01  3.19396913e+00  3.77732936e+01
  -1.93538886e+01 -3.88542189e+01  3.86914524e+01 -3.92452210e+01
  -1.57092170e+01  3.86818191e+00 -7.58248108e+00  3.09927557e+01
  -1.65522610e+01  1.15549489e+01 -2.26200780e+01  4.36086706e+01
  -3.99602420e+01 -7.72065472e+00 -1.40516112e+01 -5.01449445e+01]
 [ 3.92841756e+01  5.74244619e+00  6.84784629e+00  1.50521684e+01
   6.92472772e+00  3.21435658e+01  3.14100630e+01 -4.06742455e+01
   1.38759570e+01 -3.90437047e+01 -5.04496587e+01 -1.74107577e+01
  -2.26597970e+00 -1.98054413e+01 -5.35029925e+00 -2.24641716e+00
   3.23234032e+01 -1.21385266e+00 -2.00005068e+01  7.82576612e+00
   1.78912837e+01 -9.41551086e-01 -2.53374340e+01  5.22596222e+01
  -1.47512897e+01  1.45377622e+01 -3.88807714e+01 -2.45785087e+01
   4.49799642e+00 -1.16722354e+01  3.56773580e+01 -3.47258580e+00
   1.39461517e+01  2.06489988e+01  2.48779276e+01  8.12131597e+00
  -3.50268554e+00 -2.89130354e+01 -3.07949817e-01 -2.85812005e+01]
 [-1.89362858e+01  1.53973254e+01 -4.55633439e+01 -2.52512845e+01
   1.45623793e+01  4.55676539e+01 -3.31956096e+01  5.87243317e-01
  -3.11227795e+01 -3.69007340e+01 -1.57067110e+01 -2.61322747e+01
  -6.64697525e+00 -4.84915411e+01  1.52082549e+01  5.69174093e-01
  -3.33048911e+00  1.02752416e+01  8.91205095e+00 -5.16611619e+00
  -1.73171874e+01  2.47018674e+00 -1.00995156e+01 -2.17381141e+01
  -7.22089152e+00 -4.99564142e+01  2.50139548e+00 -1.35709235e+01
   5.35955611e+00  1.35797856e+01  4.54024074e+01  1.99938271e+01
   3.19872446e+01  2.19079195e+01 -8.53563926e+00 -1.91031828e+01
  -4.41589092e+01 -7.01234698e+00  1.70578836e+01  2.82258467e+01]
 [ 4.24778207e+00  1.50557476e+00  8.97674724e+00  2.23648816e+01
  -3.93929506e+00 -2.56512586e+01 -5.30923843e+00 -1.12360388e+00
   2.18308430e+01 -9.77061017e+00  1.81918147e+00  8.93136031e+00
  -9.76128096e+00 -1.88275389e+01 -3.16179937e+00  1.59845035e+01
   2.86141816e+00  1.30947560e+00  7.81032499e+00  4.16733151e+00
   1.28584352e+00 -1.44946076e+01 -1.22462547e+00 -4.27132957e+00
   3.41679585e+01  6.11341570e+00  1.12841175e+01 -2.50750286e+01
  -1.88572494e+01  2.65001278e+01  3.13984598e+01  5.65150151e+00
  -5.87032899e+00  2.65060851e+01 -7.75995316e+00  9.69239165e+00
  -2.65224835e+01 -7.12080133e+00 -1.67496261e+01 -2.05404181e+01]
 [ 1.53175189e+01  1.66080705e+01 -1.32695841e+01 -8.19545801e+00
  -2.15877019e+01  7.32805059e+00  2.60953787e+00 -1.62380599e+01
   6.30017759e+00 -1.54854205e+01 -1.61923264e+01  1.86046331e+01
  -1.38983010e+01  4.88830775e+00 -1.20771376e+01 -5.18442066e+00
  -8.43083640e+00 -6.91939922e+00 -1.11735767e+01 -8.42991291e+00
  -9.15617151e+00 -8.03356549e+00 -8.07597509e+00 -1.86349790e+01
   2.09285180e+00  1.74372088e+01 -8.93107146e+00 -4.17891578e+00
  -1.22614801e+01 -1.62495246e+00  1.13695625e+00  1.26149110e+01
  -6.94028769e-01 -3.08292722e+00 -2.42595802e+01 -1.02303841e+01
  -8.02418514e+00  1.62339531e+01 -7.76604456e+00  1.72573302e+01]
 [-8.35260500e-01 -1.40599993e+01  1.49905665e+01  1.34236071e+01
```

```
 -1.03827375e+01  9.02719268e+00  6.28317187e+00  1.02464406e+01
  2.39480553e+00  1.09920503e+01  1.97393794e+01 -1.23832757e+01
 -4.86094733e+00 -6.98908550e+00  8.05699709e+00 -3.57119597e-01
 -3.76875297e+01 -6.07682325e+00 -1.90229173e+01  2.52229855e+01
 -1.71268802e+01 -5.84813636e+00 -1.58893317e+01 -1.87826879e+01
  2.13089364e+01 -1.63382063e+00  1.57170817e+00 -2.20789402e+00
  1.03887571e+01  4.92113593e+00 -1.79111305e+00  7.34233379e+00
  2.69429532e+01  1.59509989e+00 -2.28893445e+01 -1.45235814e+01
 -1.55848790e+00  4.12178423e-01 -1.18462306e+01 -1.42561402e+01]
[ 2.35356302e+01  1.84693160e+01  2.81917953e+00 -8.10812600e+00
 -1.19216939e+01 -1.81604917e+01 -1.49713043e+01  3.51358182e+00
 -2.18195628e+00  1.29420826e+01 -2.13657583e+01  1.13493743e+01
  1.80579983e+01 -5.17728808e+00 -9.26281489e-01  2.06917722e+01
 -2.70972878e+01  8.36594197e+00 -8.17742536e+00  1.49575677e+01
 -7.41282453e+00  2.65695590e+00 -2.52117143e+01  2.85655676e+01
  3.14190095e+00 -1.20155833e+01  1.36275118e+01  5.65988273e+00
  8.83622044e+00  2.12688872e+00 -2.05808851e+01 -1.97774870e+01
  1.62446226e+01 -9.20490125e+00 -1.29583206e+01 -1.43358503e+01
  7.23169424e+00 -1.56613469e-01  5.38580450e+00  8.01038365e+00]
[-8.75501047e+00  2.16170782e+01 -1.22247993e+01 -2.52072986e+01
  1.24058378e+01 -2.33685276e+01  2.97569322e+01  3.15730905e+00
 -2.03824092e+01  1.05090921e+01  2.72277519e+00  1.39956372e+01
 -1.57438680e+00 -2.68230732e+01  1.47565137e+01 -1.99312370e+01
 -9.02625346e+00  1.42361458e+01 -1.10415850e+01 -3.51383318e+00
  9.36894205e+00  3.13309789e+00  2.32043465e+00  3.39566883e+00
 -2.06080789e+01 -9.00291145e+00 -1.72212824e+01 -7.45745563e+00
  1.60664499e+01 -2.09346420e+00  1.05813632e+01  1.51528897e+01
 -2.77072861e+00 -1.07010106e+01  1.23161031e+01  3.52075911e+00
 -7.91189098e-01 -3.93692706e+00  1.03475418e+01  4.66928066e+00]
[ 1.03042017e+01  2.61554583e+00  1.15827049e+01  2.09412984e+01
  4.54800654e+00 -7.47637435e+00 -1.19468505e-01  1.62974098e+01
  2.70938201e+01  1.64577027e+01 -1.07006711e+01 -8.36329226e+00
 -1.25481007e+01  2.82896560e+00 -4.06867245e-01 -4.51194029e+00
 -1.89038181e+00 -8.02259218e+00  2.22448939e+00  1.01791294e+01
  1.26397423e+00  3.91792982e+00 -1.52602042e+01 -1.11773607e+00
 -3.80727759e-01  1.47119712e+01 -6.59001855e+00 -2.81717835e-01
  2.66516096e+00 -1.00942744e+01  2.40718884e+01 -2.56089073e+00
 -3.02860403e+01  6.18586699e+00 -5.23741849e+00  6.23736573e+00
  1.07240445e+01 -2.66477979e+01 -8.60602190e+00 -1.17181993e+01]
[-4.80605689e+00  9.26600537e+00 -1.03444462e+01 -9.27517006e+00
  7.63468687e+00  8.32659464e+00  1.93697150e+00  1.14747458e+01
 -1.95600064e+00  8.82138042e+00  3.12159916e+00  8.74489612e+00
 -7.16602985e+00  9.09079433e+00  1.55628885e+01  1.23807328e+01
 -1.60189265e+01  6.63934670e+00 -2.05779998e+00 -2.66606224e+01
  2.41999526e+01  1.28838551e+00 -7.23865755e+00  6.29591167e+00
  8.54272917e+00  9.98075660e-01 -9.67119492e-01 -8.56800854e-01
 -1.85697846e+01  9.40326183e+00  3.29067184e+00 -4.42649646e+00
 -3.94434668e+00  1.58045105e+01  1.00970026e+01 -7.16707801e+00
 -5.24120226e-01 -5.63640125e+00 -2.00975725e+01 -1.62706292e+01]
[ 1.16694998e+00 -6.66398760e+00 -5.19157995e+00  5.34175611e-01
 -2.54247547e+00 -4.50139332e+00 -7.80568491e+00  2.48158181e+01
  8.99337234e+00  7.60314919e+00 -7.67377063e+00  2.68065903e+00
 -9.09506090e+00 -2.85048031e+00  6.00282537e+00 -1.79140386e+01
  1.42348984e+01 -1.05143602e+01  1.14246483e+01 -3.04782729e+00
 -9.91355473e+00  5.12686626e+00 -7.12524073e+00  5.43302392e+00
 -3.30703654e+00  1.30679968e+01 -1.10755705e+01  3.07723276e+00
 -4.71644747e+00  7.12223369e+00 -9.45880408e+00 -1.80250856e+01
  2.86021224e+01 -1.12454291e+01 -1.11708058e+01  8.54144338e+00
 -1.36335779e+01  1.22521415e+00  1.08970445e+01 -6.97211232e+00]
[ 1.19325906e+01  1.69622535e+01 -7.48258805e+00  1.20607250e+01
  5.06098420e+00 -1.58148497e+01  1.03794676e+01  1.22910065e+01
  7.78546145e+00 -3.29984054e-01  1.07335599e+01 -2.09667742e+01
  6.41500118e-01  6.71743224e+00  1.47574471e+00 -9.17695286e+00
  1.13729036e+01 -2.94163079e+00 -1.82565169e+00  8.73910920e+00
 -1.41451370e+00  5.76883311e-01  5.49314296e+00 -1.67074345e+01
 -3.13837661e+00  4.68322832e+00 -1.23135083e+00  1.91034478e+00
 -5.29221304e+00 -5.66752169e+00 -2.09936694e+00 -2.18068050e+01
 -1.53114751e+01  1.29627972e+01  4.34850659e+00 -2.91107708e+00
 -1.48696630e+01  7.04777959e+00 -5.68232502e+00  1.14739266e+01]
[-4.13396730e-01 -1.55822872e+01 -1.12375277e+01 -2.05296569e+01
  9.65375130e+00  1.50980416e+01  1.04641661e+01 -8.94295488e+00
```

```
  3.45421202e+00  1.41973140e+01  6.43956302e+00 -5.80617855e+00
  8.85846741e+00  1.09734564e+00  2.53046197e+00 -1.46023332e+00
 -7.26456517e+00  1.50544459e+00  1.31224288e+01  3.44769312e-01
 -1.06223661e+01  9.22175802e+00 -4.35379763e+00 -3.42631059e-01
 -3.50749877e+00 -2.86620726e+00  1.58895353e-01 -1.53808511e+00
  8.34856972e+00 -1.14035369e+01 -4.07627548e-01 -1.58534365e+01
 -9.26501364e+00  1.09556195e+01 -4.76785792e+00  1.60665299e+01
  1.23439488e+01  4.34949054e+00 -9.92418129e-01 -3.53474934e+00]
[ 6.64047818e+00 -1.95347694e+01 -4.76476092e+00 -1.20310802e+01
 -6.01917231e+00 -5.06251251e+00  5.41769039e+00  1.08302874e+01
  5.18521647e+00  3.47790553e+00 -2.84352753e+01 -1.63834735e+01
  1.58883793e+00  1.69071129e+00  5.11109817e+00 -5.14560983e+00
 -5.98845324e+00 -3.83624109e+00 -1.52014922e+01 -8.19100948e+00
 -4.43551684e+00 -3.89119602e+00  2.31763567e+00 -9.34048714e+00
  3.84360309e+00 -6.44773742e+00  6.26926407e-01 -6.25658347e+00
 -7.51775699e+00  3.18196106e+00  1.54208368e+01 -1.79056162e+01
  7.83338358e+00  9.78865834e-01  2.17850828e+01 -1.31381999e-01
  5.89123344e+00 -8.51173431e-01  9.42209915e+00 -4.69310519e+00]
[ 1.54730625e+00 -4.44530055e+00 -2.16090772e-01  5.57229105e-01
  1.34861462e+00 -1.05787910e+01 -6.78111000e-02 -3.97297521e+00
  1.62886949e+01  1.33906458e+00 -2.39724740e+00 -2.63513501e+00
 -9.23556030e+00  9.23268949e+00  5.83847026e+00  1.16647694e+01
  1.83758305e+01  1.64091183e-01 -1.11313772e+01 -9.59857062e+00
  9.80469044e+00 -1.27178681e+01  5.49517062e+00  1.03936578e+01
 -4.14002995e+00  6.96504534e+00 -2.55254229e+00 -1.74277707e+01
  2.60378205e+00 -2.59955995e+00 -1.17625913e+01 -1.20167736e+00
 -7.86510476e+00 -8.67117505e+00  1.84404428e+01  4.55186608e+00
 -8.31807352e+00  3.79760331e+00  1.20491940e+01  1.81951446e+01]
[ 9.46007719e+00  1.73714898e+00 -8.56684958e+00  1.59938178e+00
 -3.57141713e+00  2.65644587e+00 -6.11233893e+00 -1.01551758e+01
 -1.37457247e+01  5.40831278e-01  5.45760259e+00 -1.45757409e+01
  2.47371796e+00  2.99579645e+00  9.89656245e+00 -1.32893259e+00
 -9.66320185e-01  6.10027047e+00  1.01023056e+01  1.82537810e+00
  1.36664039e+01 -5.34706574e+00  6.58171063e+00  1.15418610e+01
 -6.76540582e+00  1.50203295e+01  1.91339517e+01  5.62204471e+00
  7.25393735e+00  3.41610713e+00 -5.83224691e+00 -4.80819852e+00
 -1.73226624e-01 -8.02521238e+00 -3.78056487e+00 -1.37773927e+01
  6.08779380e+00  5.05819409e+00  6.84775722e+00  1.67042973e+00]
[-1.20126629e+01 -1.39805435e+01  7.84443078e-01 -4.98027235e+00
 -2.90303591e+00 -8.80835851e+00  1.57199443e+01 -9.04581540e+00
  9.01221766e+00 -7.38720979e+00 -9.31048701e-01  1.60590572e+00
 -1.05816952e+01  3.99702837e+00 -2.07159824e+00  1.34698727e+01
 -1.28032997e+01 -7.83506404e+00  1.69727804e+01  5.45428241e+00
 -5.50469293e+00 -5.31165658e+00 -1.24252900e+01  9.44377112e-01
  4.31717606e+00 -5.11092214e+00 -4.62146678e+00  4.86588293e+00
  9.64686265e-01  1.86652434e+00 -1.54484167e+01  1.01693880e+01
 -4.68424506e+00 -3.12074985e+00 -3.57070656e-01  3.50491986e+00
  3.99232559e+00 -1.12211666e+01  5.04375482e+00  1.27923824e-01]
[-1.60372574e+00 -1.35711983e-01 -4.77869273e+00  1.85745934e+00
  1.82229780e+00  1.18052072e+01  1.12220190e+01 -2.72152349e+00
 -8.22114216e+00 -7.17305738e+00  3.28798900e+00 -1.45323586e+01
  4.89076436e-01  7.96046911e+00  1.90549655e+00 -4.40562843e+00
 -4.41578911e+00 -7.28124002e+00 -1.84059011e+01  1.58714244e+01
 -9.26600910e-01 -1.96258143e+00 -6.88309096e-01  2.45756491e+00
 -5.61337482e+00 -1.52919251e+00 -9.48612414e+00  3.07195834e-01
 -6.80549266e+00 -1.80679462e+00 -5.51214420e+00  9.89437495e+00
  4.94252841e+00 -6.15618517e+00 -2.28789169e+00  6.16130973e+00
  4.01717774e+00  3.61880381e+00  1.31804068e+00 -1.20993004e+00]]
```

3. Take the minimum euclidean distance between the test image weight and all the training sample weights to predict the class of the test image

In [ ]:

```
train_sample_weights = train_sample_weights.T
test_sample_weights = test_sample_weights.T

similar_index = []
for i in range(0, 40): # Each test image
    distance = [] # Store Euclidian distance
```

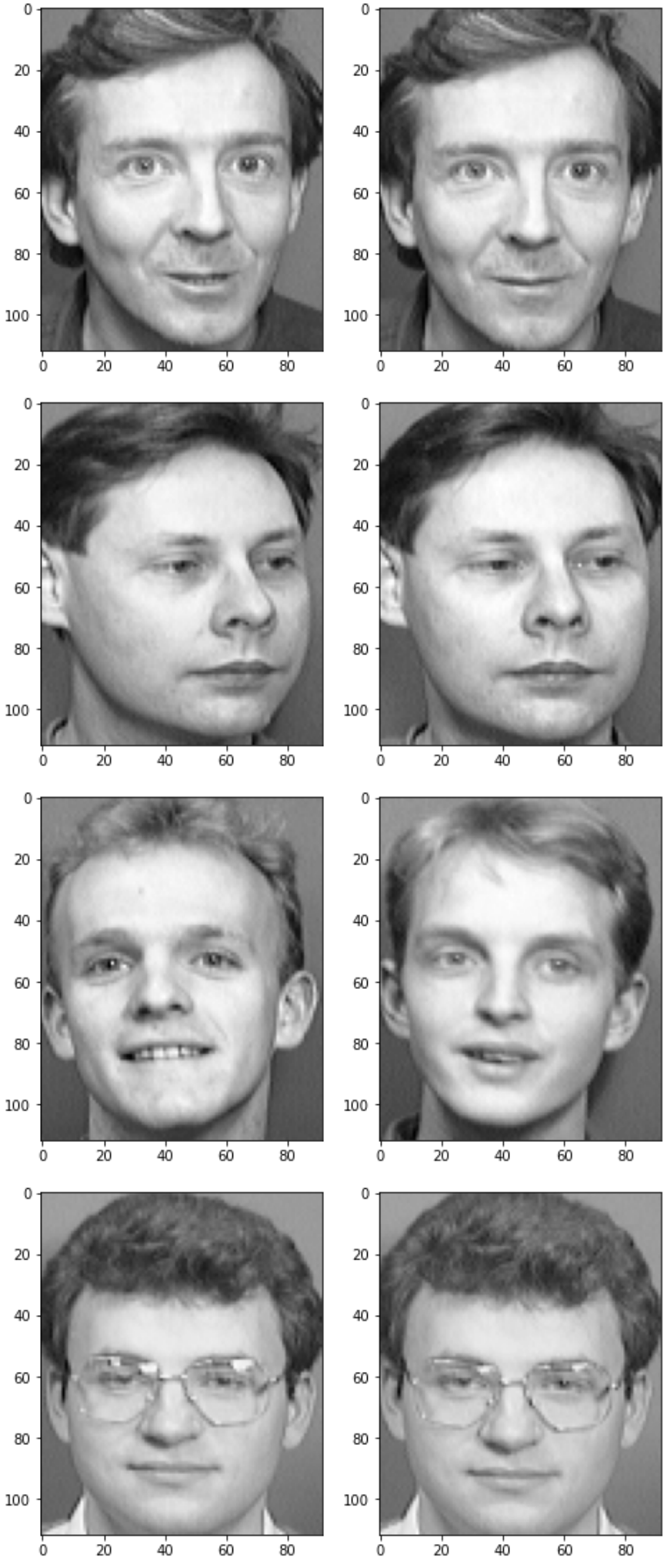
```
    for j in range(0, 360): # Each train image
        minDistance = np.linalg.norm(train_sample_weights[j] - test_sample_weights[i]) # Calcul
        distance.append(minDistance)
    similar_index.append(np.argmin(distance)) # Find the nearest distance

res = [] # Find the class
for each in similar_index:
    res.append(trainset_number[each])
print(res)
```

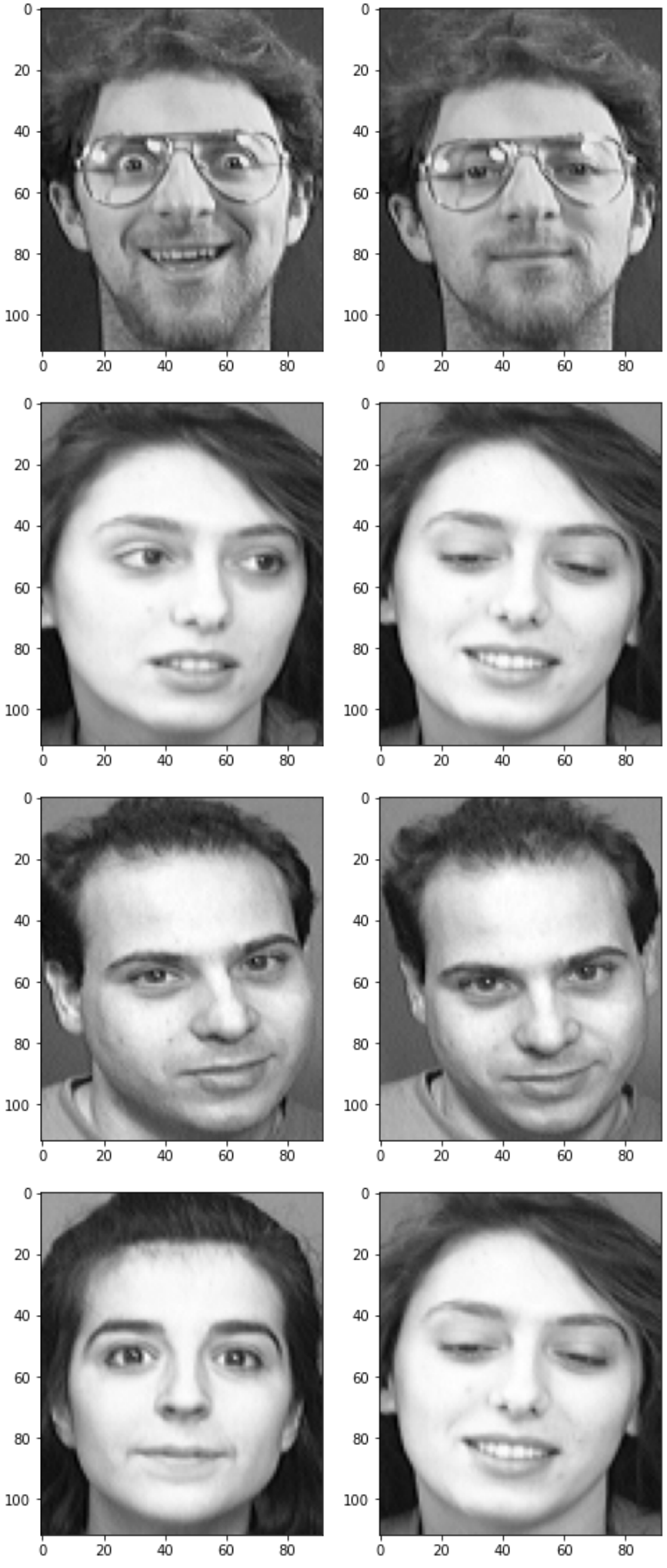
```
[1, 2, 3, 4, 40, 6, 7, 8, 9, 8, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26,
27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40]
```

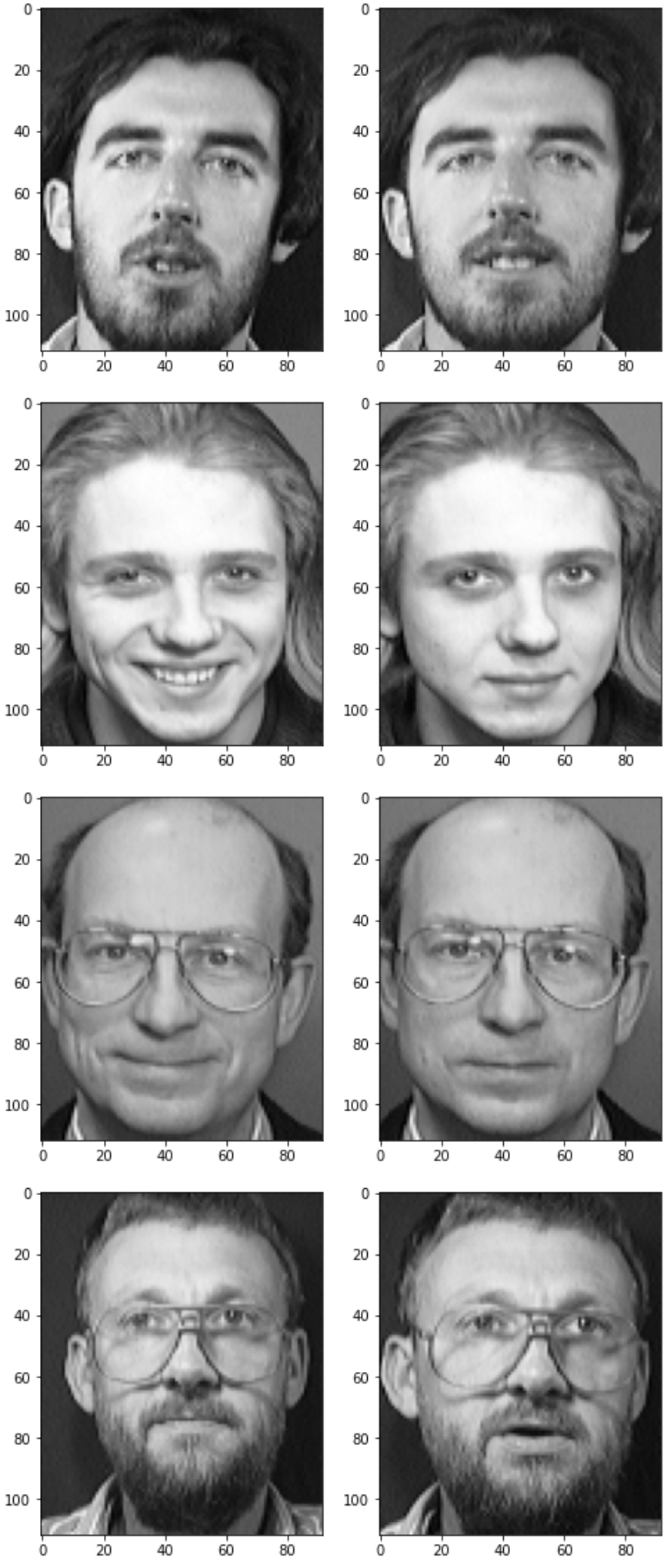
4. For each test image, Visualize the test image and the train image with minimum distance

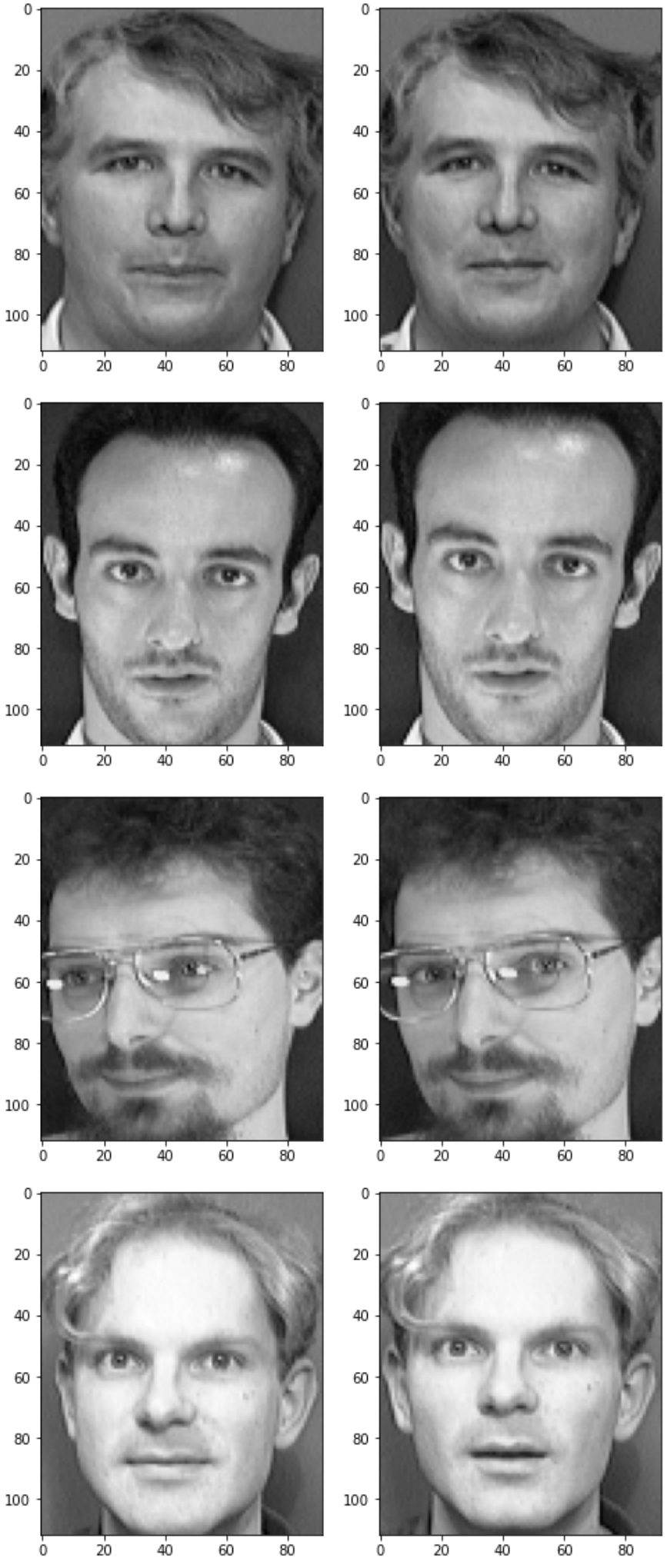
In [ ]:

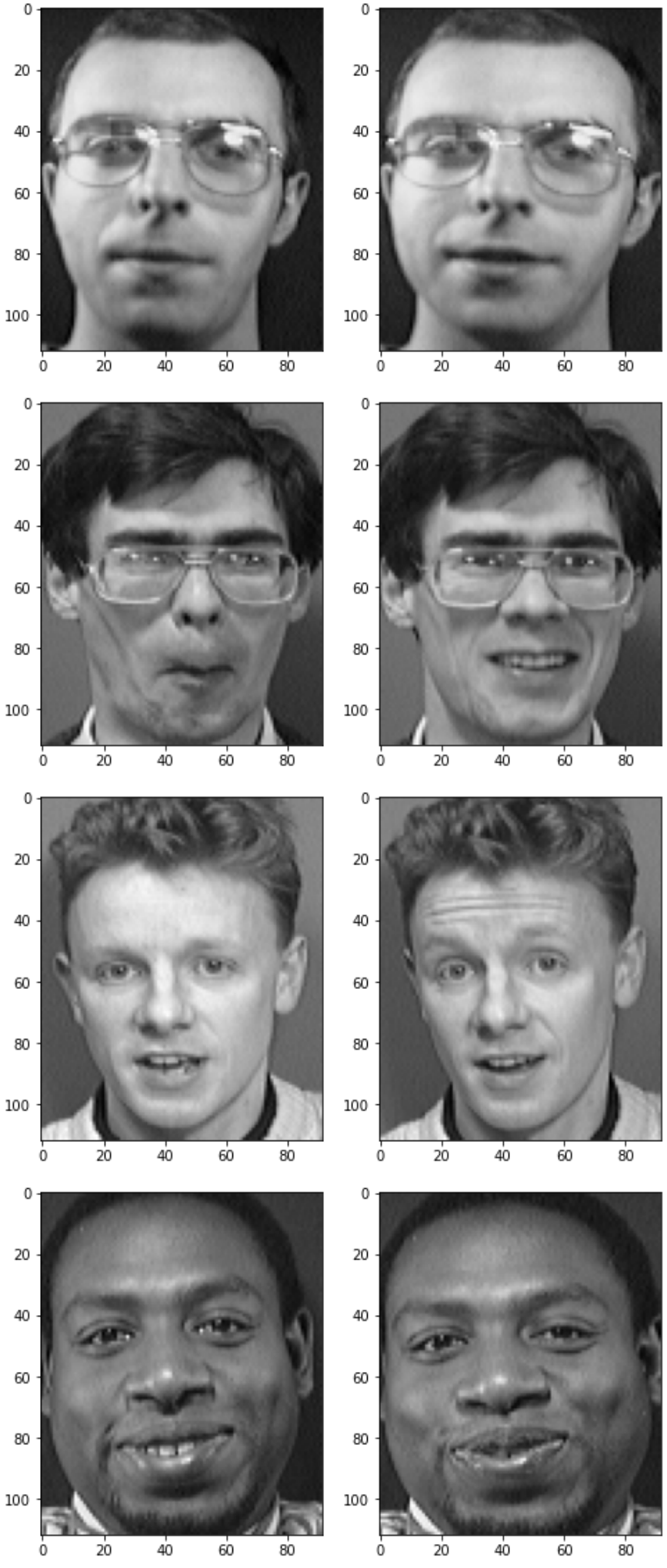
```
for i in range(1, 41):
    fig = plt.figure(figsize=(8, 8))
    fig.add_subplot(1, 2, 1)
    plt.imshow(original_test[i - 1], cmap="gray") # Test image
    fig.add_subplot(1, 2, 2)
    plt.imshow(original_train[similar_index[i - 1]], cmap="gray") # The train image with minimu
    plt.show()
```

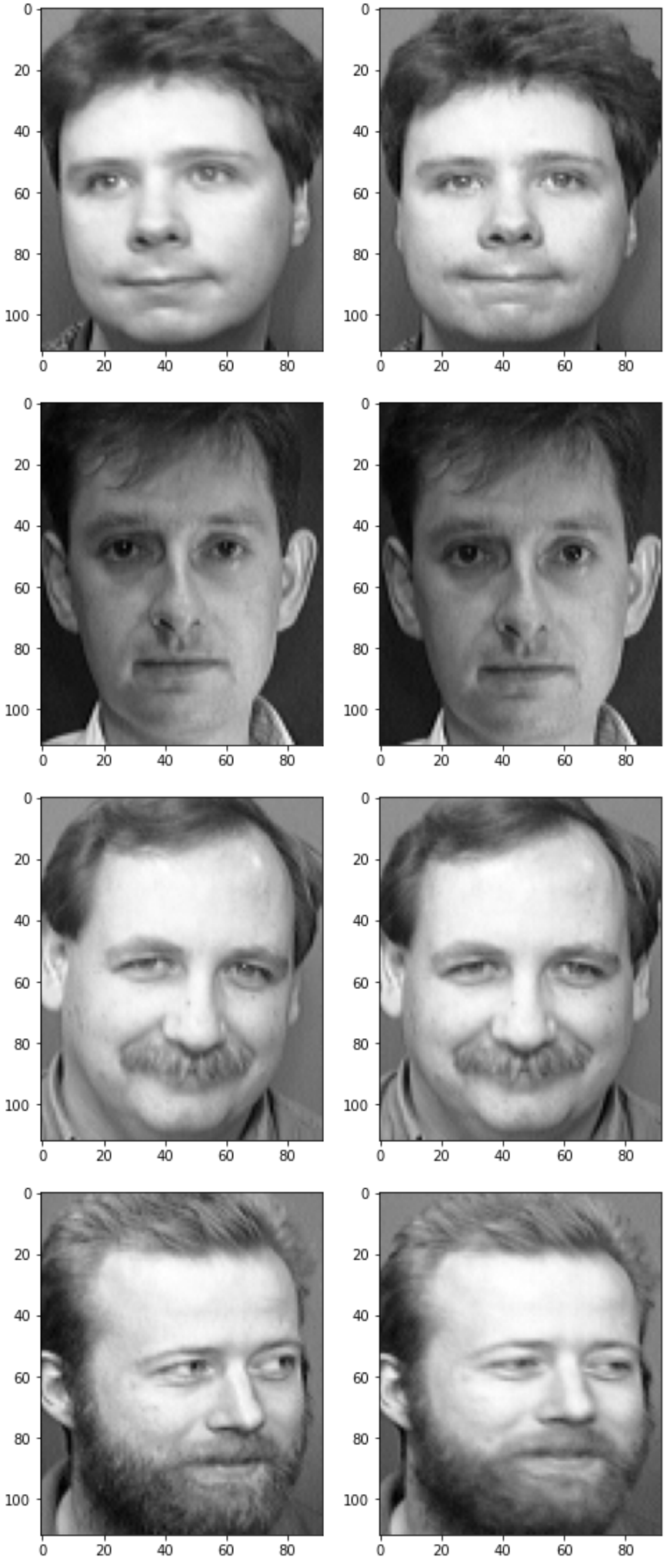

0
20
40
60
80
100
0
20
40
60
80

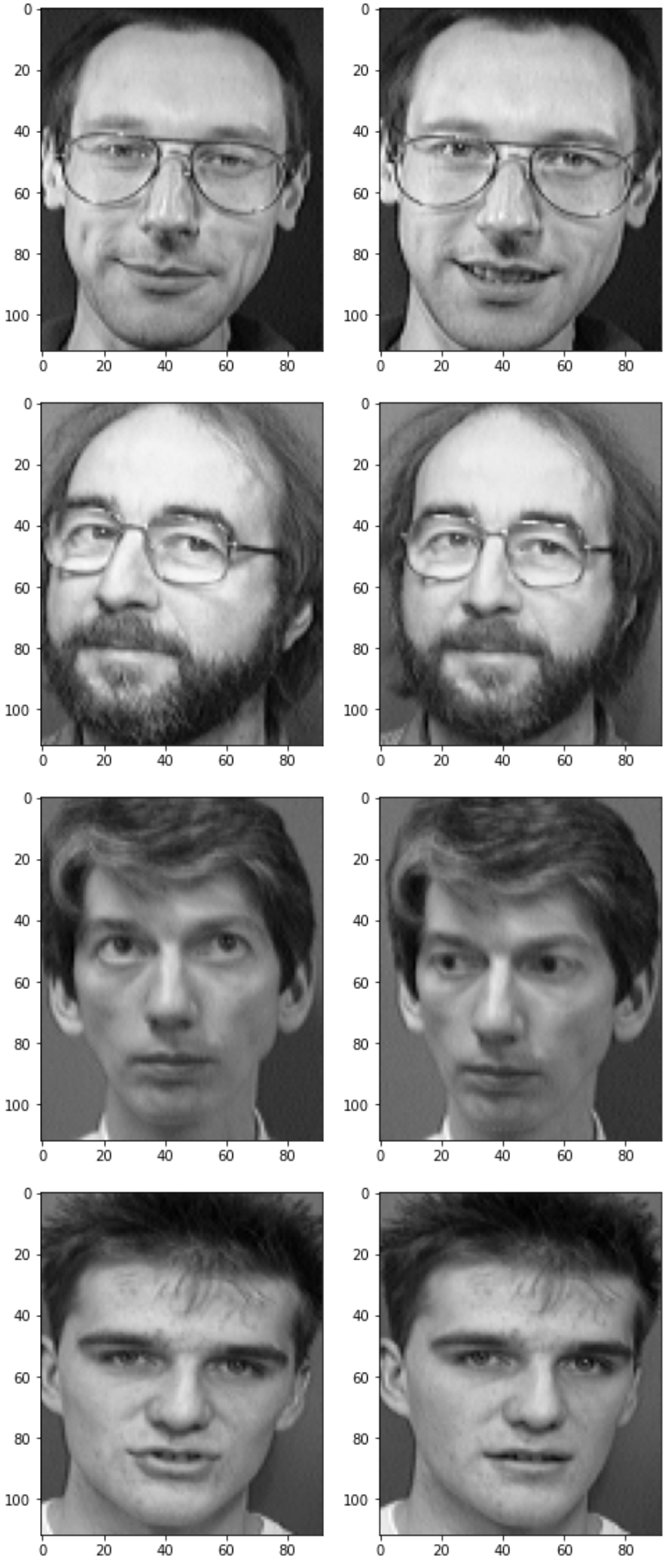

0
20
40
60
80
100
0
20
40
60
80

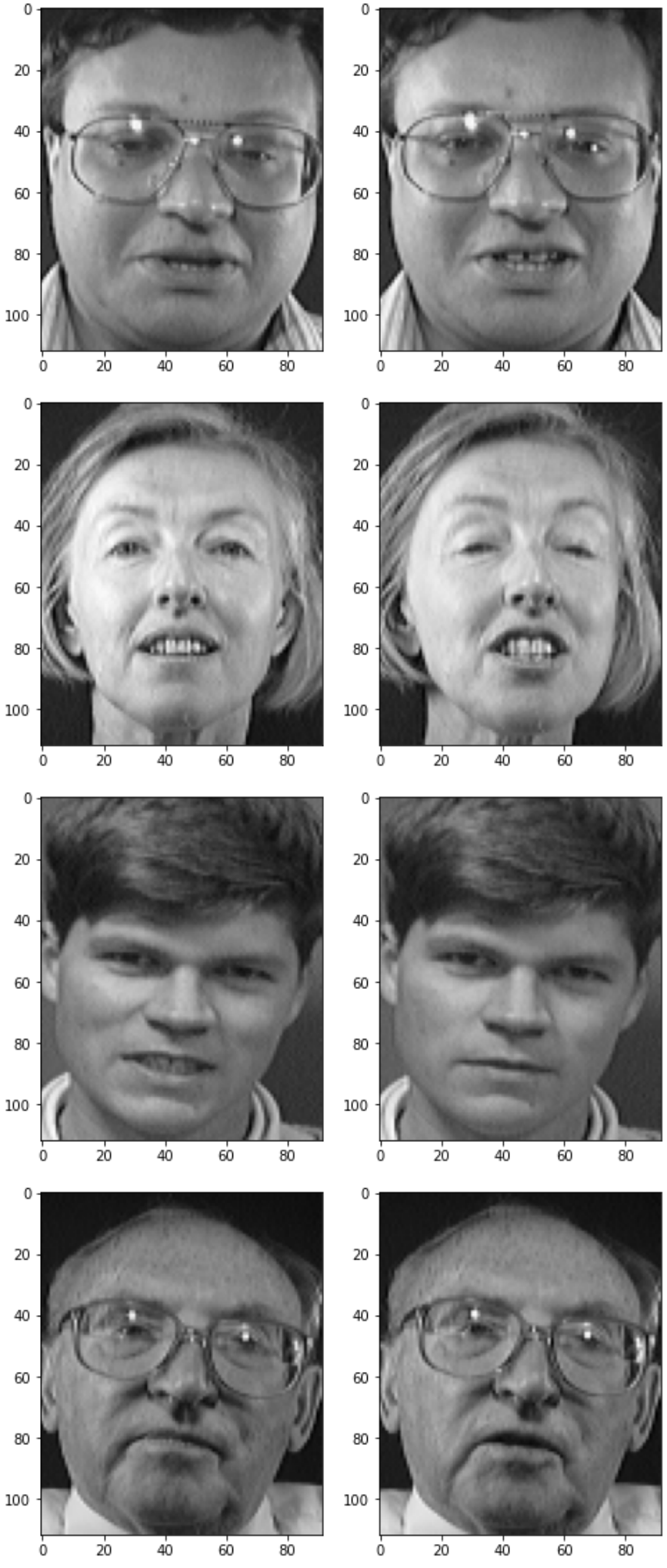

0
20
40
60
80
100
0
20
40
60
80

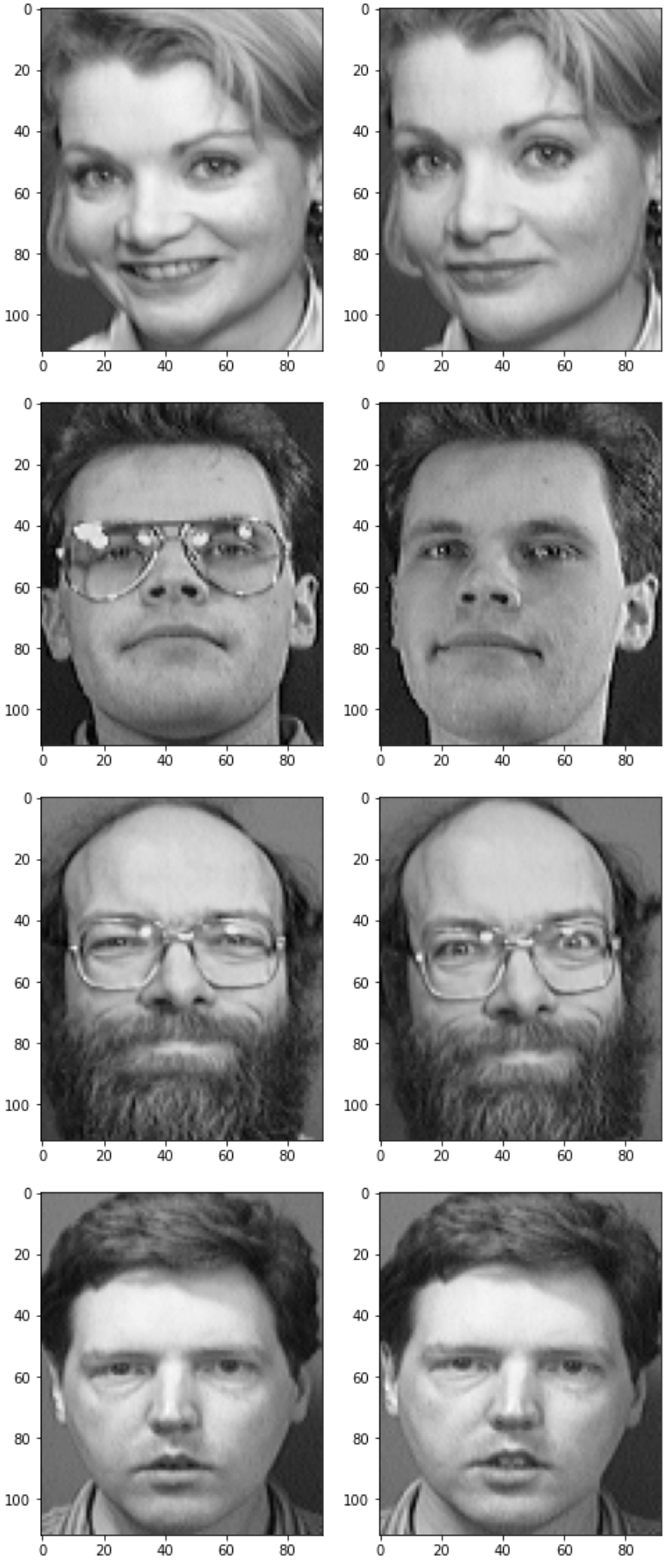

0
20
40
60
80
100
0
20
40
60
80

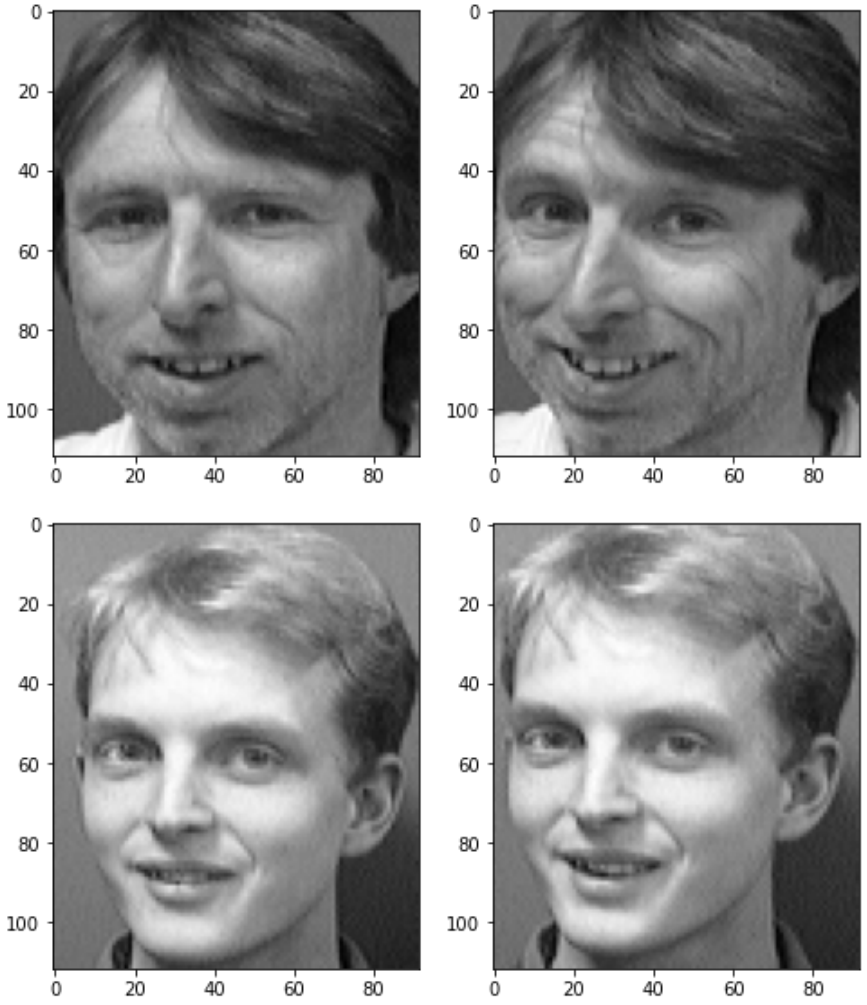

0
20
40
60
80
100
0
20
40
60
80


0
20
40
60
80
100
0
20
40
60
80


0
20
40
60
80
100
0
20
40
60
80


0
20
40
60
80
100
0
20
40
60
80


0
20
40
60
80
100
0
20
40
60
80



5. Report accuracy and total explained variance ratio by the selected components

In [ ]:
```
count = 0 # The number of wrong predict
index = 0
for i in res:
    if i != index + 1:
        count = count + 1
    index = index + 1
print("The accuracy of first 20 principal components:" + str(float(40 - count) / float(40)))
print("The total explained variance ratio of first 20 principal components:" + str(sum(pca.expl
```

```
The accuracy of first 20 principal components:0.95
The total explained variance ratio of first 20 principal components:0.6954736141113826
```

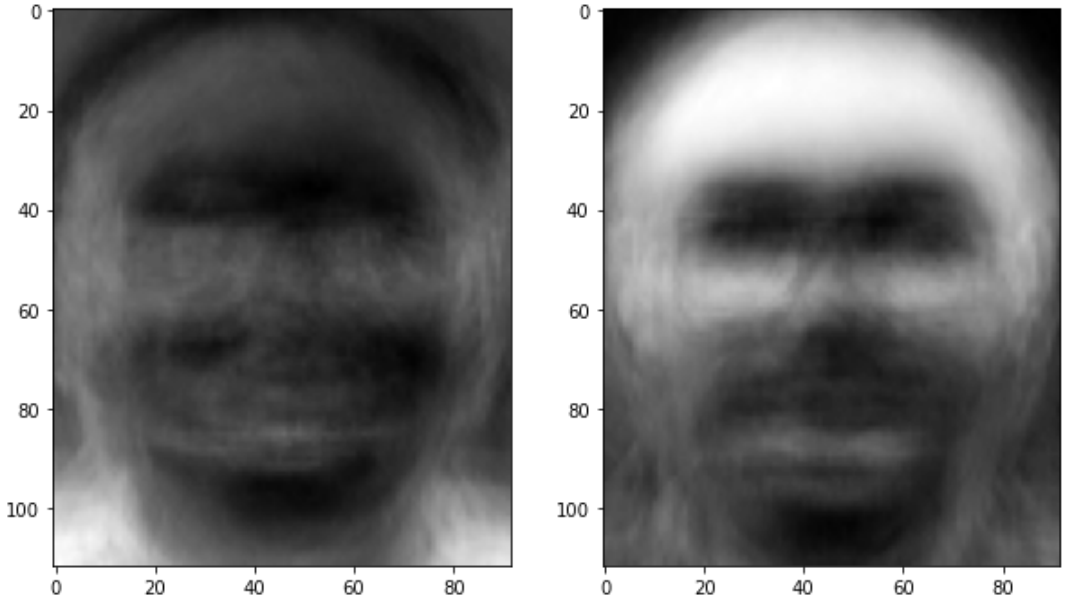
Part 5 - Face Recognition

1. Repeat Part 3 using only first two principal components instead of 20. Visualize the first two eigenfaces

In [ ]:
```
n_components = 2 # Take first 20 principal components
print(
    "Extracting the top %d eigenfaces from %d faces" % (n_components, trainset.shape[0]))
# Apply PCA using scikit-learn on the Train set
pca_2 = PCA(n_components=n_components, svd_solver="randomized", whiten=True).fit(trainset)
# Reshape the flattened vector to original image shape
eigenfaces_2 = pca_2.components_.reshape((n_components, 112, 92))
# Visualize the first 2 eigenfaces
fig = plt.figure(figsize=(10, 10))
for i in range(1, 3):
    img = eigenfaces[i - 1]
```

```
    fig.add_subplot(1, 2, i)
    plt.imshow(img, cmap="gray")
plt.show()
```

```
Extracting the top 2 eigenfaces from 360 faces
```



2. Repeat Part 4

2.1. Calculate the weights of the training samples using the given formula

In [ ]:
```
train_sample_weights_2 = np.matmul(pca_2.components_, (trainset - pca_2.mean_).T)
```

2.2. For each test image, calculate weights similarly

In [ ]:
```
test_sample_weights_2 = np.matmul(pca_2.components_, (testset - pca_2.mean_).T)
```

2.3. Take the minimum euclidean distance between the test image weight and all the training sample weights to predict the class of the test image

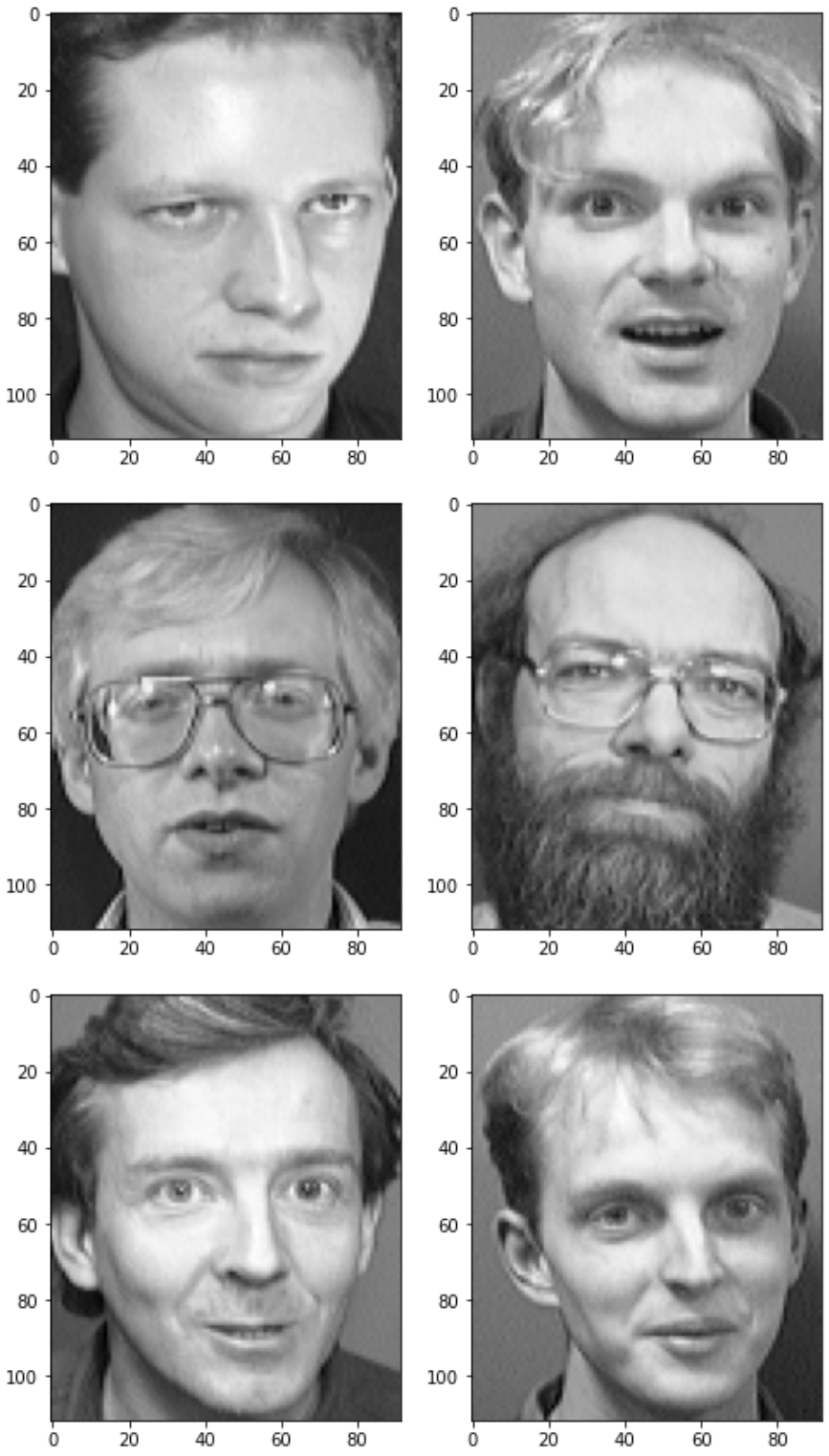
In [ ]:
```
train_sample_weights_2 = train_sample_weights_2.T
test_sample_weights_2 = test_sample_weights_2.T

similar_index_2 = []
for i in range(0, 40):
    distance = [] # Store Euclidian distance
    for j in range(0, 360):
        minDistance = np.linalg.norm(train_sample_weights_2[j] - test_sample_weights_2[i]) # Ca
        distance.append(minDistance)
    similar_index_2.append(np.argmin(distance)) # Find the nearest distance

res_2 = []
for each in similar_index_2:
    res_2.append(trainset_number[each])
print(res_2)
```

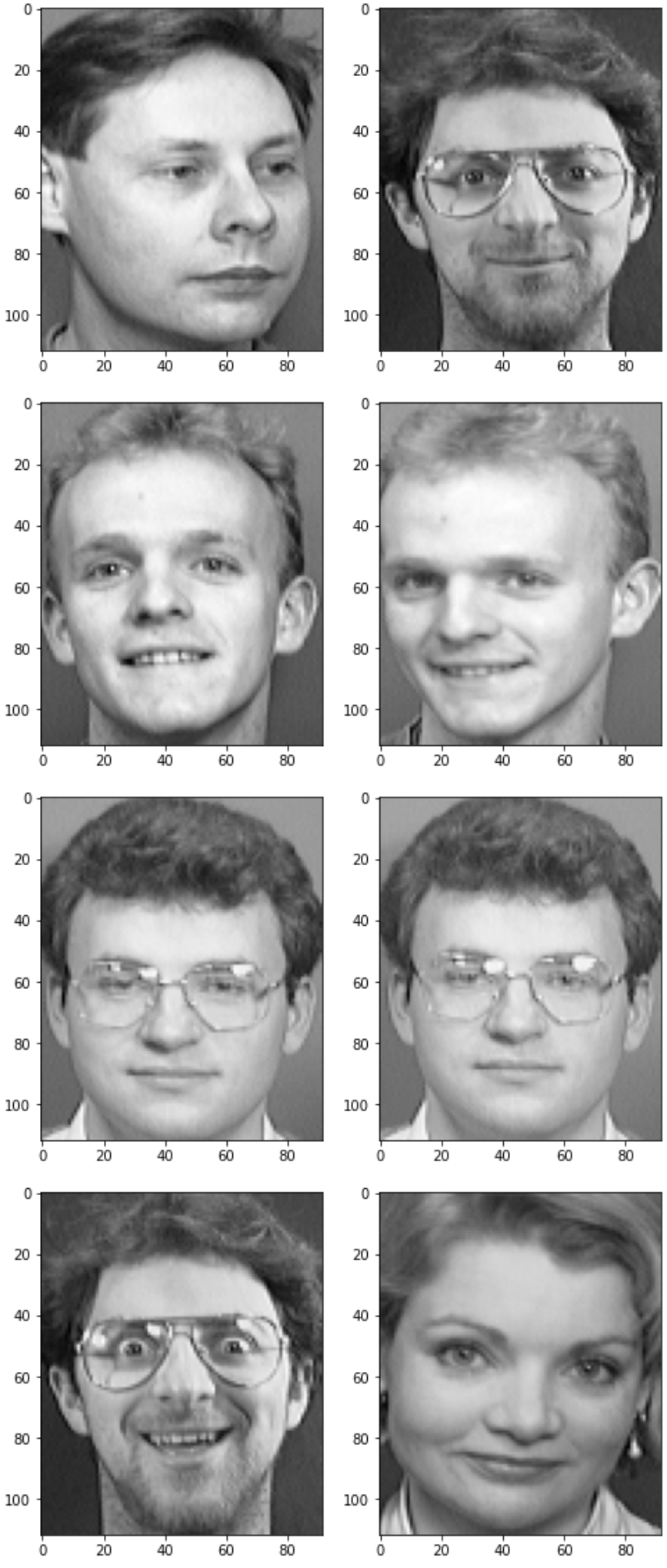
```
[18, 37, 40, 7, 5, 6, 35, 8, 17, 9, 31, 12, 13, 34, 17, 8, 17, 18, 28, 30, 21, 22, 9, 24, 40,
5, 28, 19, 39, 31, 31, 26, 33, 34, 5, 40, 37, 23, 39, 25]
```

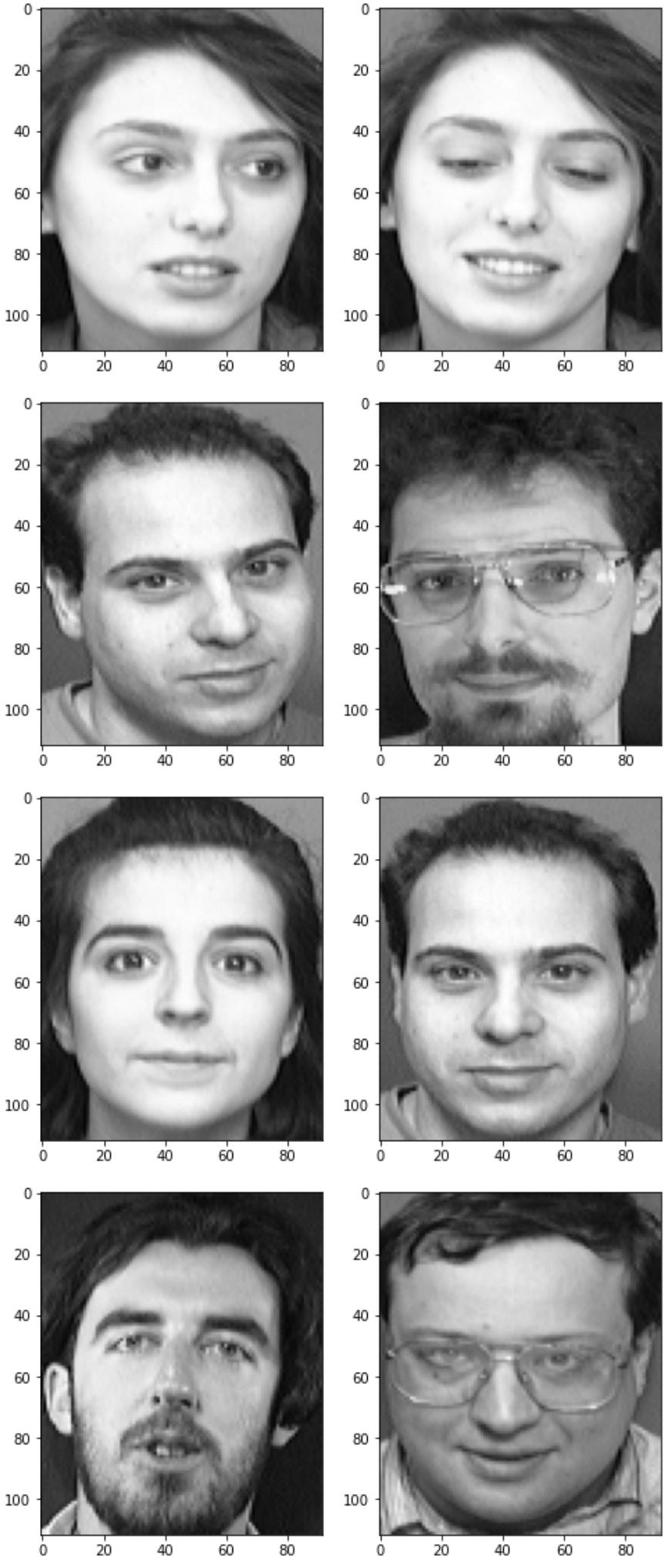
2.4. For each test image, Visualize the test image and the train image with minimum distance

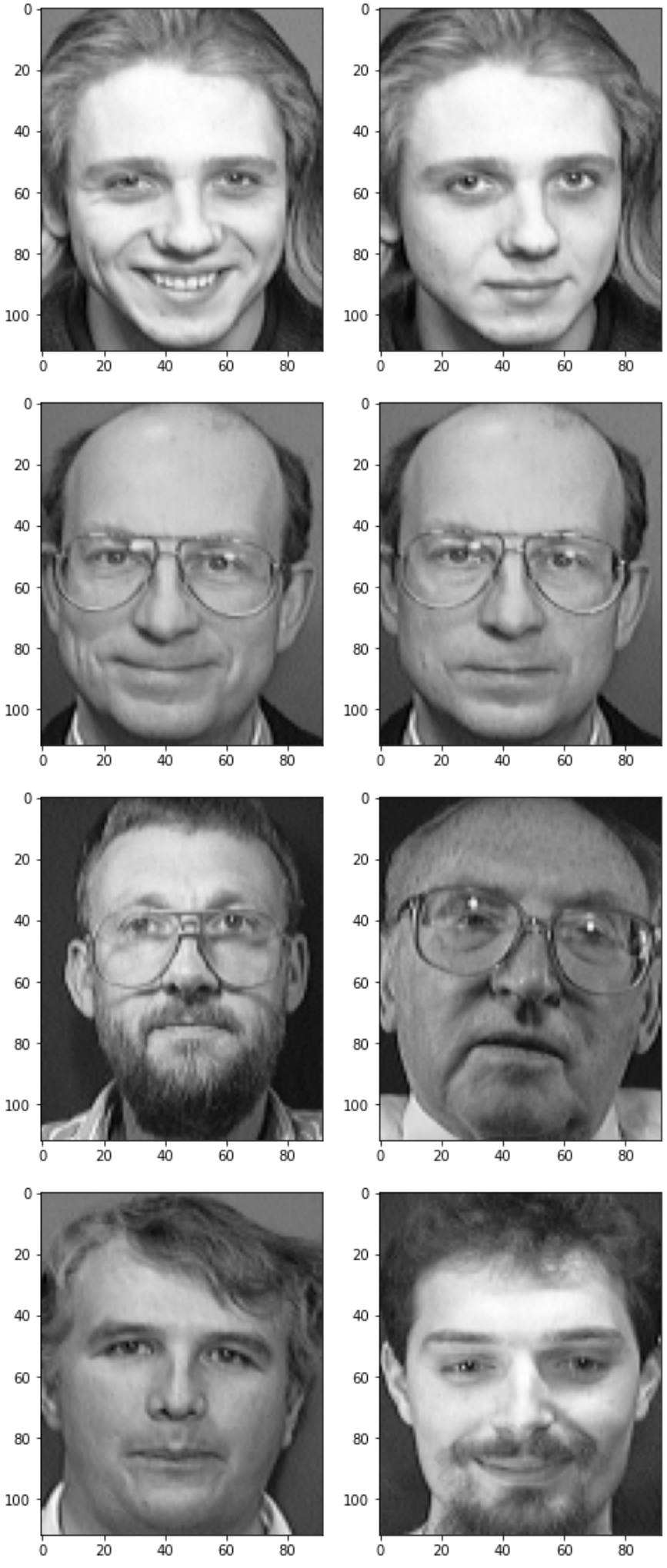
In [ ]:
```
for i in range(1, 41):
    fig = plt.figure(figsize=(8, 8))
    fig.add_subplot(1, 2, 1)
    plt.imshow(original_test[i - 1], cmap="gray") # Test image
    fig.add_subplot(1, 2, 2)
```

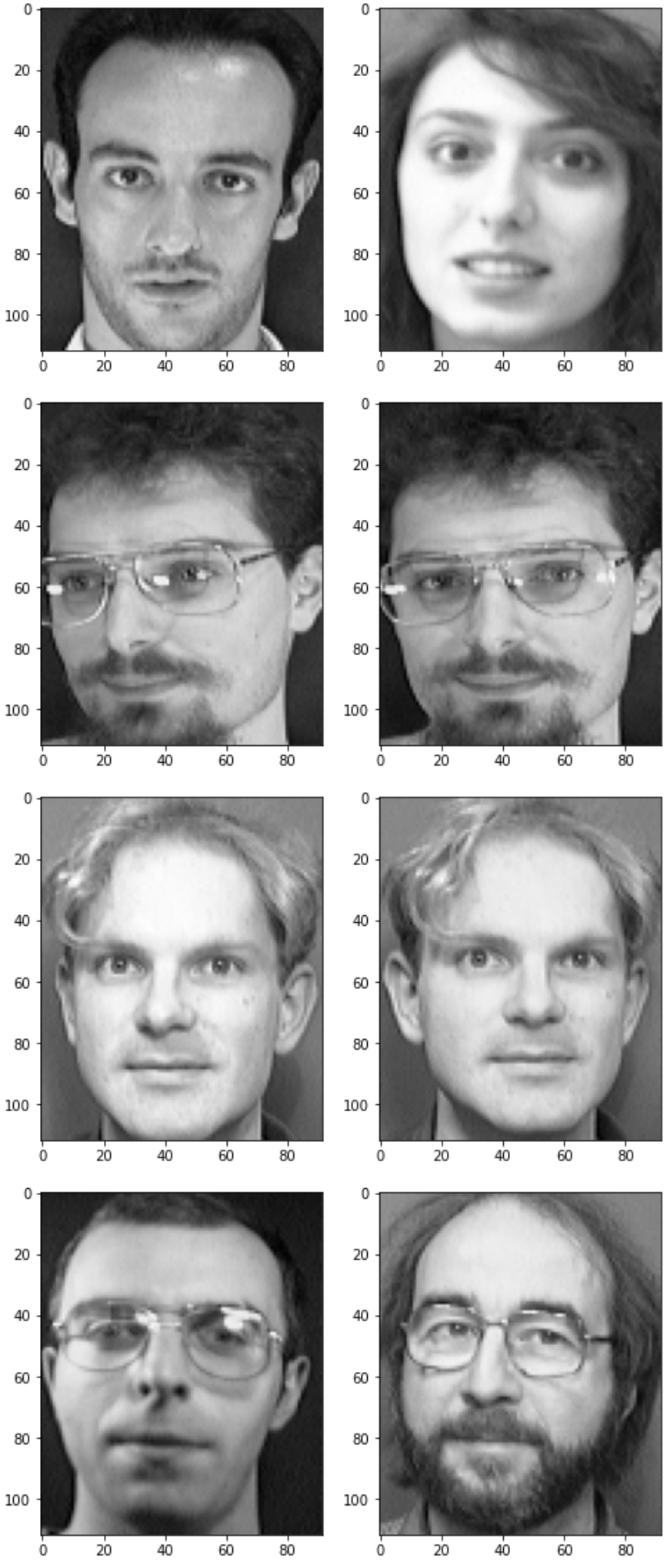
```
    plt.imshow(original_train[similar_index_2[i - 1]], cmap="gray") # The train image with mini
    plt.show()
```

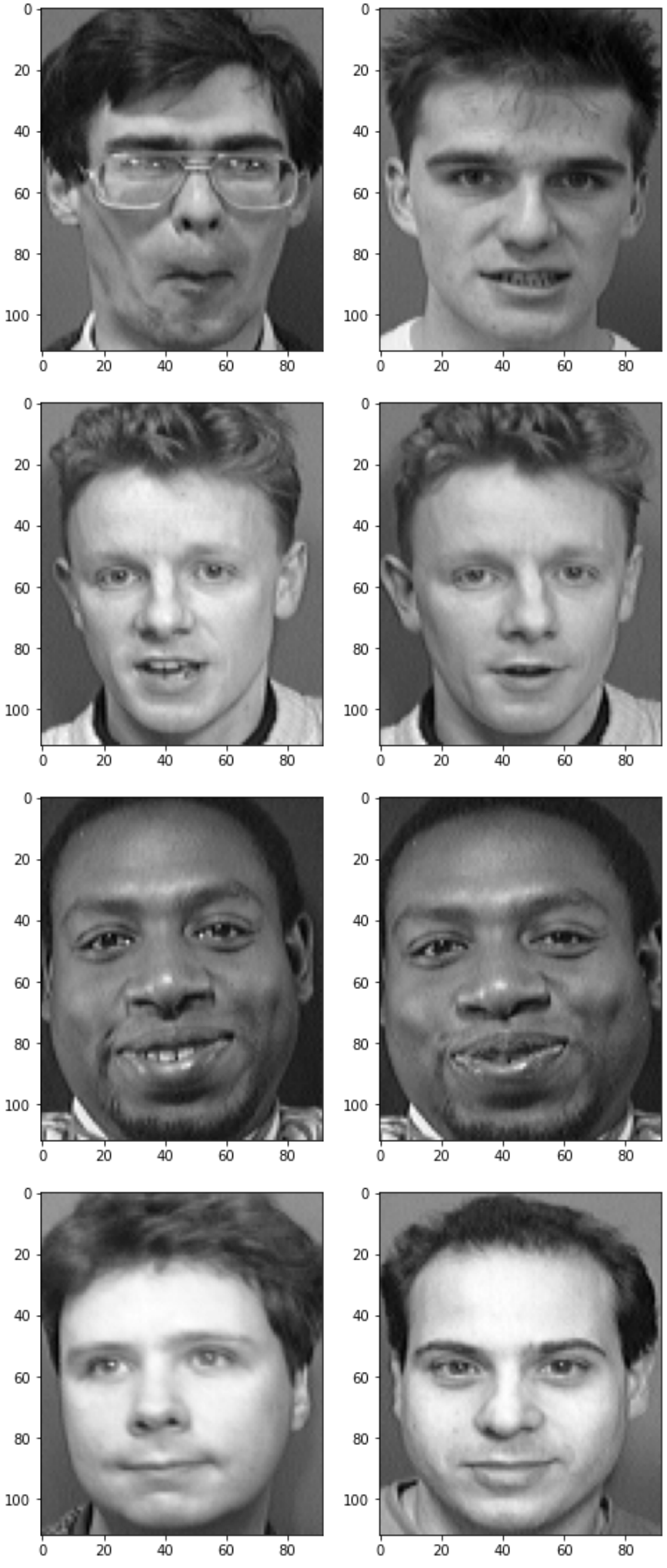


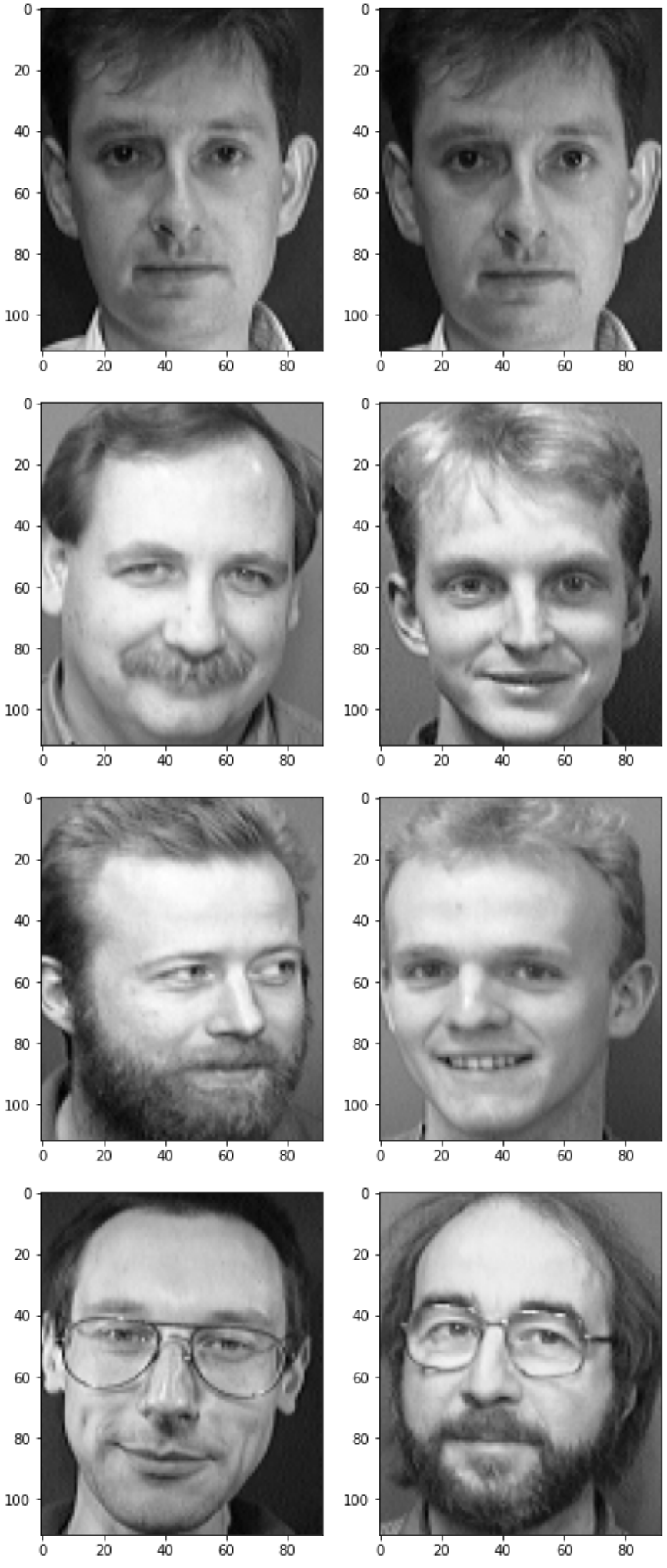

0
20
40
60
80
100
0
20
40
60
80

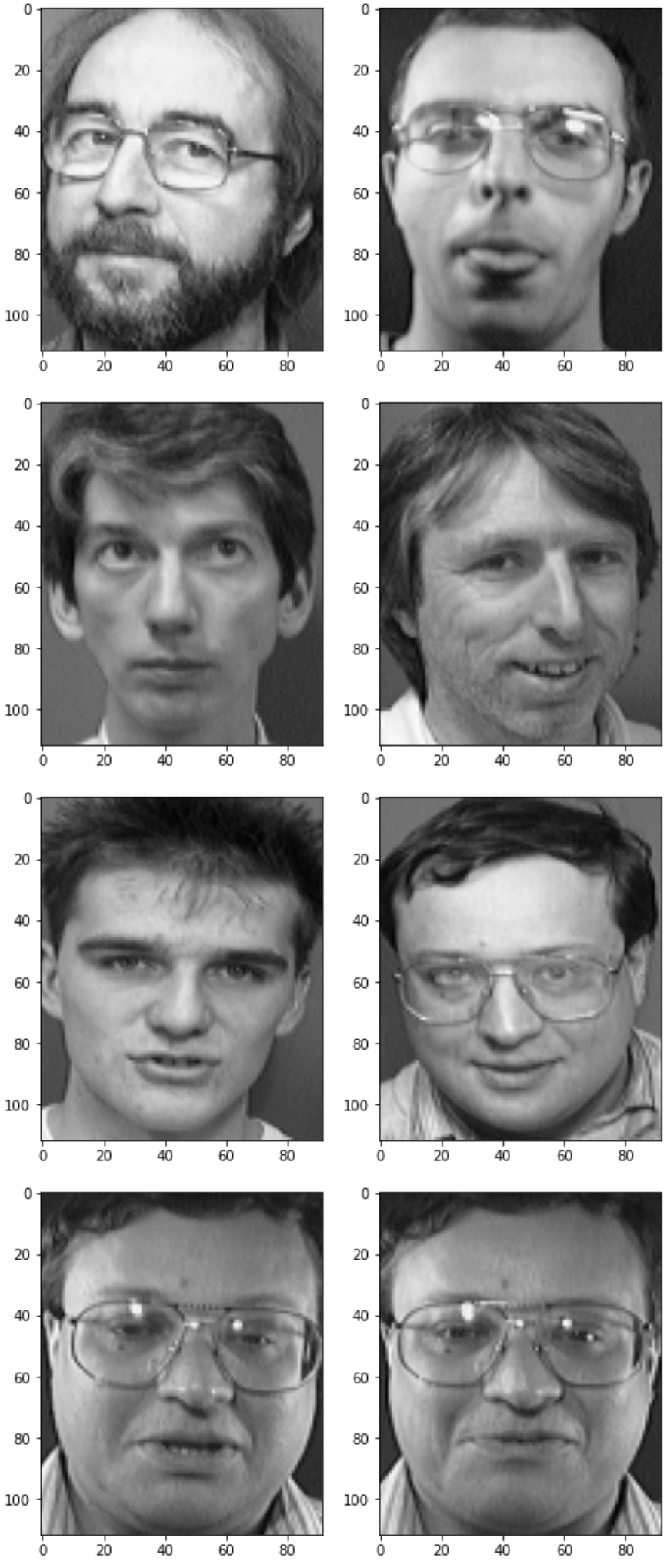

0
20
40
60
80
100
0
20
40
60
80

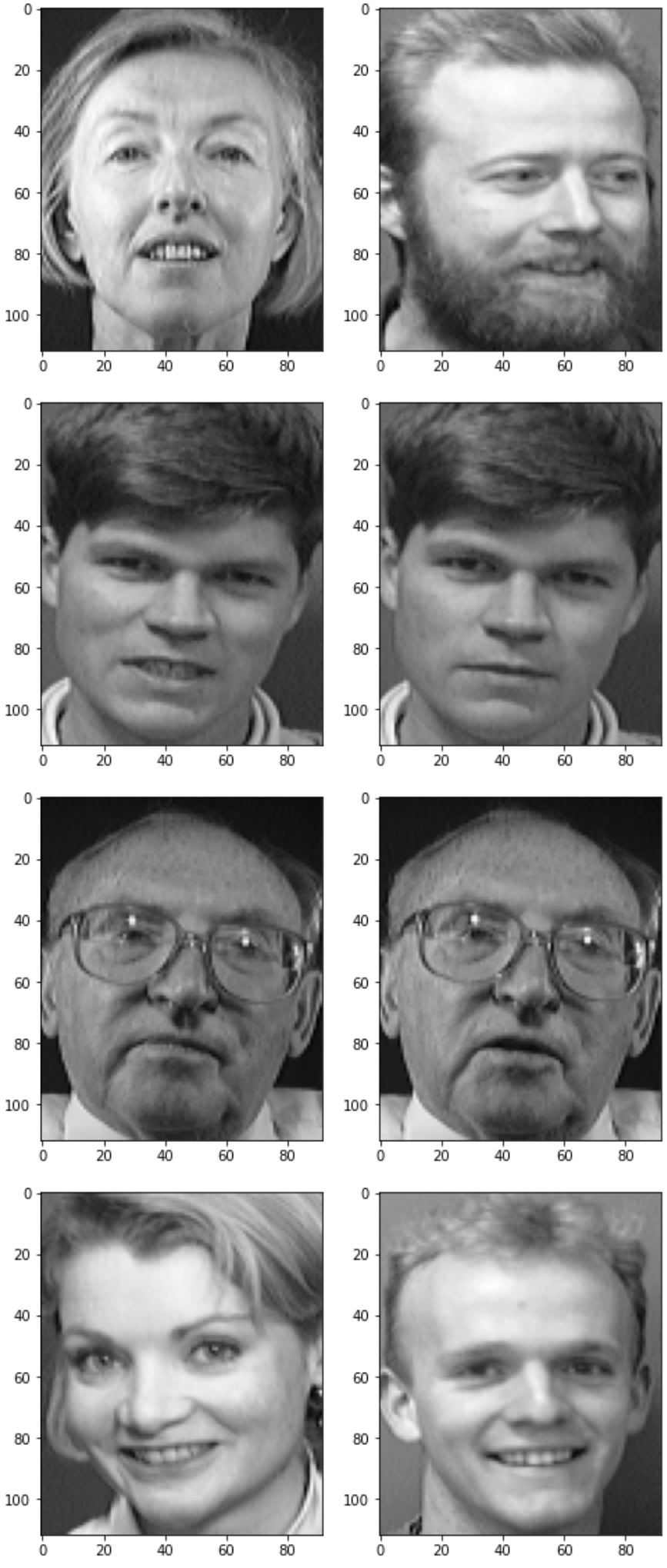

0
20
40
60
80
100
0
20
40
60
80

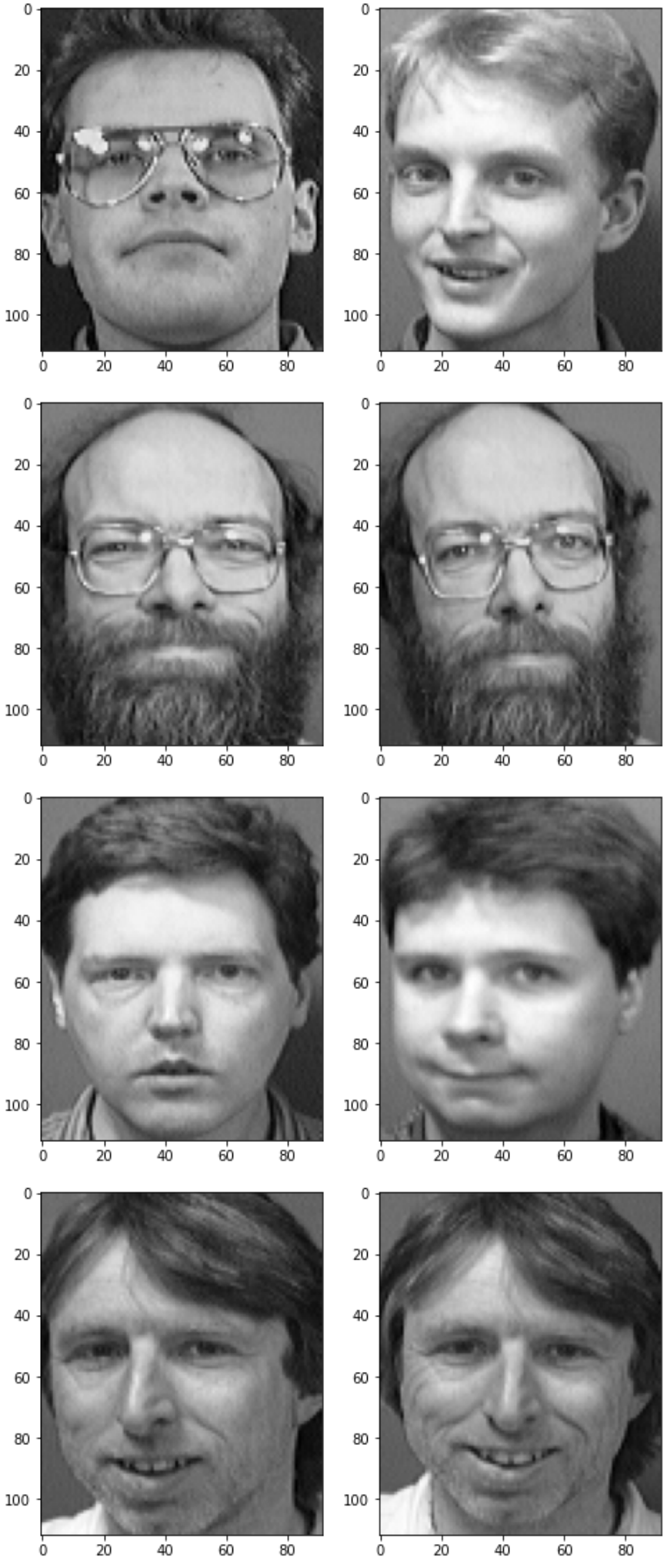

0
20
40
60
80
100
0
20
40
60
80

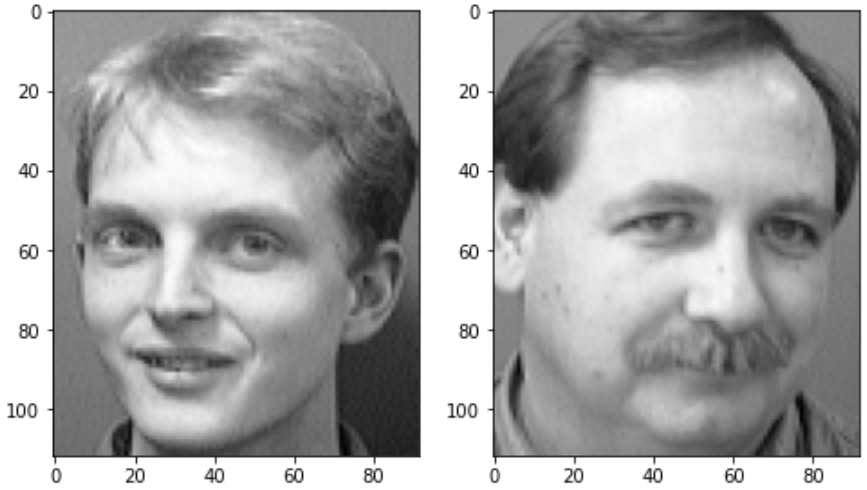

0
20
40
60
80
100
0
20
40
60
80


0
20
40
60
80
100
0
20
40
60
80


0
20
40
60
80
100
0
20
40
60
80


0
20
40
60
80
100
0
20
40
60
80


0
20
40
60
80
100
0
20
40
60
80



2.5. Report accuracy and total explained variance ratio by the selected components

In [ ]:
```
count_2 = 0
index_2 = 0
# Calculate the accuracy
for i in res_2:
    if i != index_2 + 1:
        count_2 = count_2 + 1
    index_2 = index_2 + 1
print("The accuracy of first 2 principal components:" + str(float(40 - count_2) / float(40)))
print("The total explained variance ratio of first 2 principal components:" + str(sum(pca_2.exp
```

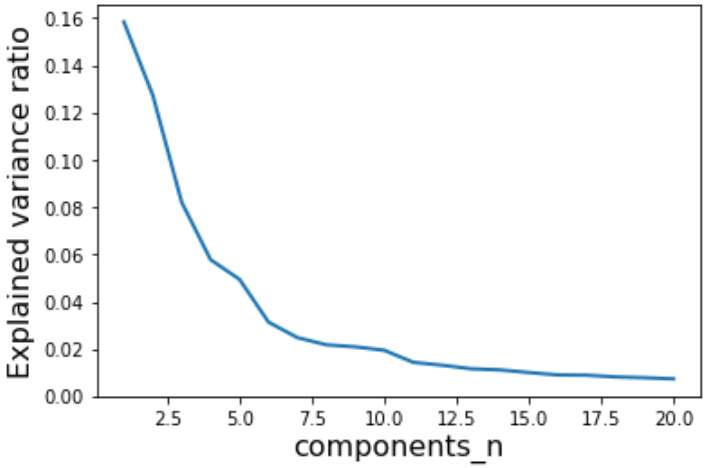
```
The accuracy of first 2 principal components:0.375
The total explained variance ratio of first 2 principal components:0.28585491291282616
```

3. Compare the results using the explained variance ratio of PCA

In [ ]:
```
print("The explained variance ratio of first 20 principal components:" + str(pca.explained_varia
print("The total explained variance ratio of first 20 principal components:" + str(sum(pca.expla
print("The explained variance ratio of first 2 principal components:" + str(pca_2.explained_var
print("The total explained variance ratio of first 2 principal components:" + str(sum(pca_2.exp
```

```
The explained variance ratio of first 20 principal components:[0.15842818 0.12742673 0.08206044
0.05779132 0.04947382 0.03138806
 0.02482998 0.0217958  0.02093417 0.01948993 0.01438802 0.0131587
 0.01164532 0.0111796  0.01006655 0.00904037 0.00892188 0.00821429
 0.00785629 0.00738417]
The total explained variance ratio of first 20 principal components:0.6954736141113826
The explained variance ratio of first 2 principal components:[0.15842818 0.12742673]
The total explained variance ratio of first 2 principal components:0.28585491291282616
```

In [ ]:
```
plt.figure()
plt.plot(np.arange(1,21),pca.explained_variance_ratio_,linewidth=2)
plt.xlabel('components_n', fontsize=16)
plt.ylabel('Explained variance ratio', fontsize=16)
plt.show()
```



We can see that the explained variance ratio of the first two principal conmponents are the same in part3 and part5. But the total explained variance ratio of first 20 principal components is almost 0.7 which is much more bigger than the total explained variance ratio of first 2 principal components which is 0.29. The accurcy of using 20 principal components is 0.95 while the accurcy of using 2 principal components is only 0.375. The efect is much better when using 20 principal components.

And the reason can be found from the figure, we can see that the explained variance ratio keeps being a large number when n is smaller than 7. In other words, the first 7 principal components can significantly improve the accuracy of PCA. Although I didn't do this, we can control the number of k by thresholding in the function. This balances the accuracy of result and the resources consumed.