Mini project 2

Face detection based on PCA

Name: Yujia Ding
PID: A59000953
Email: y5ding@ucsd.edu

Introduction:
Human face detection plays an important role in applications such as personal identity verification. However, realizing face detection is quite difficult because of the variation in appearance, different viewpoint and environmental distribution, such as varying lighting conditions and complex backgrounds. [1] The principle of face detection is treating each window in the image like a vector v and finding whether v match some vector u in the database. But directly using the space of all images is impossible because face images are extremely high-dimensional when viewing each pixel value as a vector. To solve this problem, we need construct a low-dimensional linear subspace, containing most of the face images with small errors. Some methods are used to extract data from images more accurately ,such as Local Binary Patterns, Histogram of Oriented Gradient(HOG), Principal Component Analysis (PCA).

Principal Component Analysis (PCA) is a method to compute the principal components which reflect a change of basis on the raw data. Through selecting and sorting few import principal components, it can realize dimensionality reduction and preserve

main features of raw data.[2] The following steps can be used to compute principal components:

1. Processing data: Suppose each face image is a n*m dimensional matrix, and we would like to obtain 1-D array of images which containing all features. Then we collect all images features as matrix A^{K*N} K is features of each matrix and N is the number of images.
2. Subtract the mean: calculate mean faces of all images. $\Psi = \frac{1}{M} \sum_{i=1}^{M} \Gamma_i$.
3. Calculate the covariance matrix of $A*A^T$ and perform eigenvalue decomposition on this matrix .
4. Calculate the eigenvectors and eigenvalues of the covariance matrix and the eigenvectors (eigenfaces) must be normalised so that they are unit vectors.

5. Select the principal components: Based on eigenvalues, we can select most import PCs.

Following above steps, we can do the experiments.

Implementation and Experiments

1. Process Faces: read image by using cv2 library and transform each image to 1-D array to collect all features of face and collect 190 faces from the train set. At this step we obtain a matrix, called img3 (size 31266*189).

2. Subtract the mean: We calculate mean of each row and obtain an array of mean with 31266 features. Then we subtract the mean from each image, obtaining a matrix img4(img3-imgmean)

3. **Construct covariance matrix and obtain AA^T eigenvectors and eigenvalues:** We calculate the eigenvectors and eigenvalues of the covariance matrix AA^T by using np.linalg.eig() function

4. Select the principal components: Plot the relationship between singular values of eigenvectors and the number of PCs.

5. Reconstruct faces:

(1) Calculate eigenfaces by using "np.dot(A, eigenVec)".(I called it as AV.)

(2) Normalize the eigenfaces: The normalization is very important because we can't get projection faces without the normalization.

However, I was stuck on the problem for several days, I try two ways to calculate the normalization of the eigenface:

First, I use the equation AV_norm=AV /np.linalg.norm(AV), but this function calculate the norm of all matrix. The result is wrong and I obtain extremely small eigenfaces norm from above equation.

Then I try another way. I use follow equations to calculate it:

V_i=eigenVec[i] .....(1)

AVi=np.dot(A,V_i).....(2)

AVi_unit=AVi/np.linalg.norm(AVi)..... (3)

But at step (3), it obtains the norm of each row. It still output a wrong result. The projection images seem good when I use less PCs, but MSE is strange, which is increasing when the number of PCs increases.

Finally, with the help of TAs, I find that I need calculate the L2 norm of each column instead of all matrix or the row.

I use the below equation to obtain good result.

AV_norm=AV/np.linalg.norm(AV, axis=0, keepdims=True)

(3) Choose different number of PCs.

(4) Load test image and process it as step 1 and 2.

(5) Calculate weigh and projection faces:
weigh= np.dot (uT ,w)
Proj_faces=np.dot(u,weigh)+mean.
w is the processed test image and u is the normalized eigenfaces.

6. Obtain reconstructed test images

7. Calculate MSE.

Results:

1. mean face:

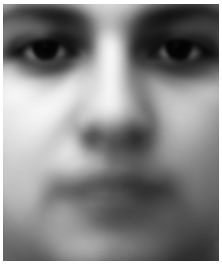


fig 1. The mean face of 190 individuals' neutral expression. The mean face contains the general information about all 190 faces,

2. singular values of the data matrix:

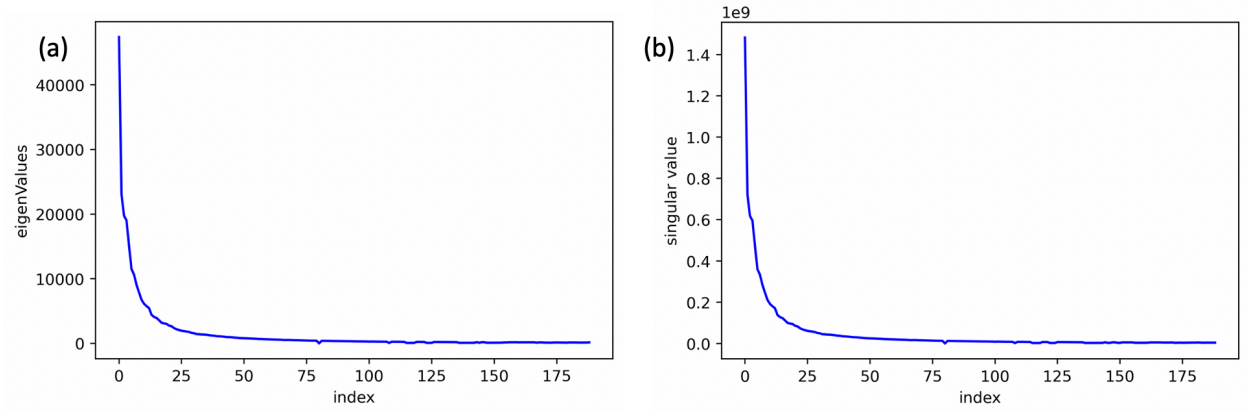


fig 2. (a) the eigenvalue decomposed from cov(A) versus index; (b) the singular values versus index.

From fig 2, we can see that the top 25 eigenvalues make sense to reconstruct faces, while eigenfaces with low eigenvalue contains trivial information.

3. Reconstruct one of 190 individuals' neutral expression image and plot MSE of reconstruction versus the number of principal components.

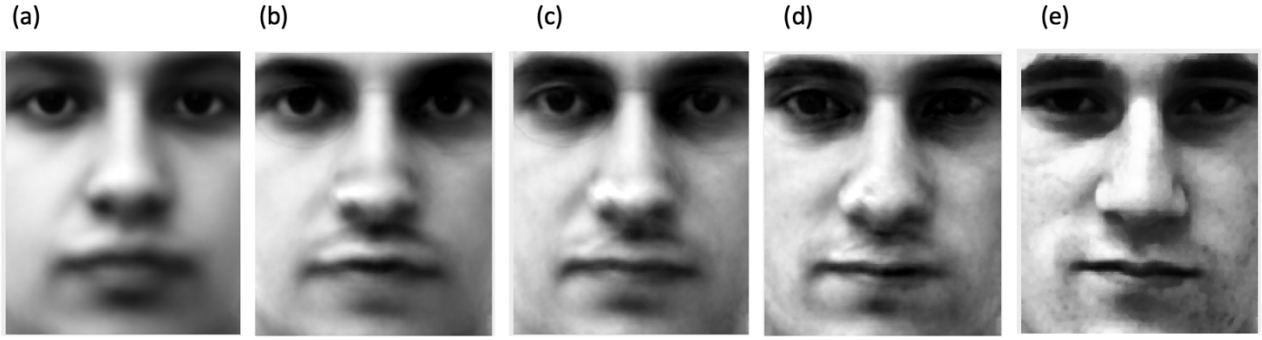


fig 3. (a) 1 PCs; (b) 10 PCs, (c) 25 PCs; (d) 60 PCs; (e) 189 PCs;

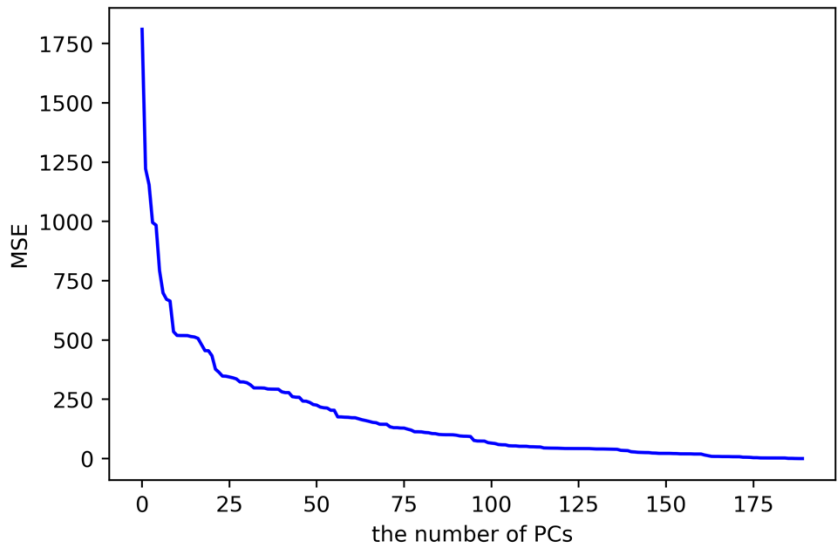


fig 4. the MSE of reconstruction faces versus the number of principal components.

When using only one PCs, it looks different from the mean face, adding some male characteristics. When using 10 PCs, we can distinguish the face as a male face. Then when adding up to 60 PCs, it looks similar to its original face. We can know who the man is from this reconstructed picture. When using all PCs, more details are added to the image.

Meanwhile the plot of MSE indicate the same conclusion: the MSE continuously decline when adding more PCs

4. Reconstruct one of 190 individuals' smiling expression image by using different number of principal components and plot the MSE of reconstruction versus the number of principal components.

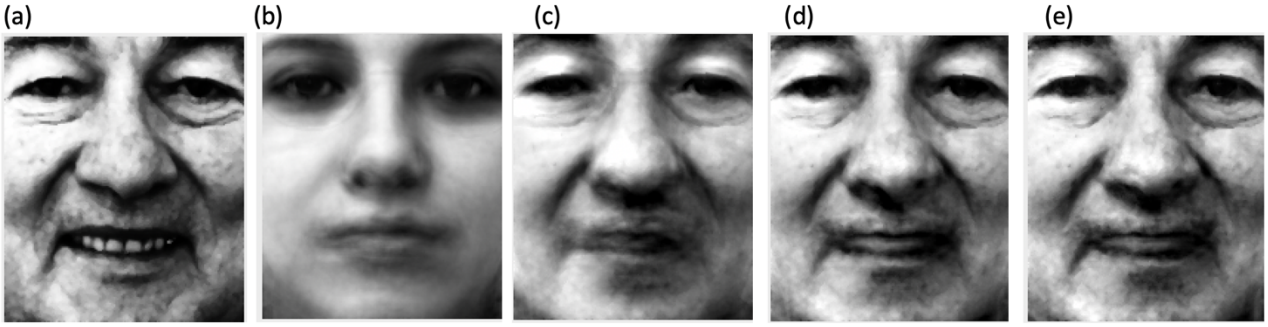


fig 5. (a) the original face; (b) 1 PCs, (c) 25 PCs; (d) 60 PCs; (e) 189 PCs;

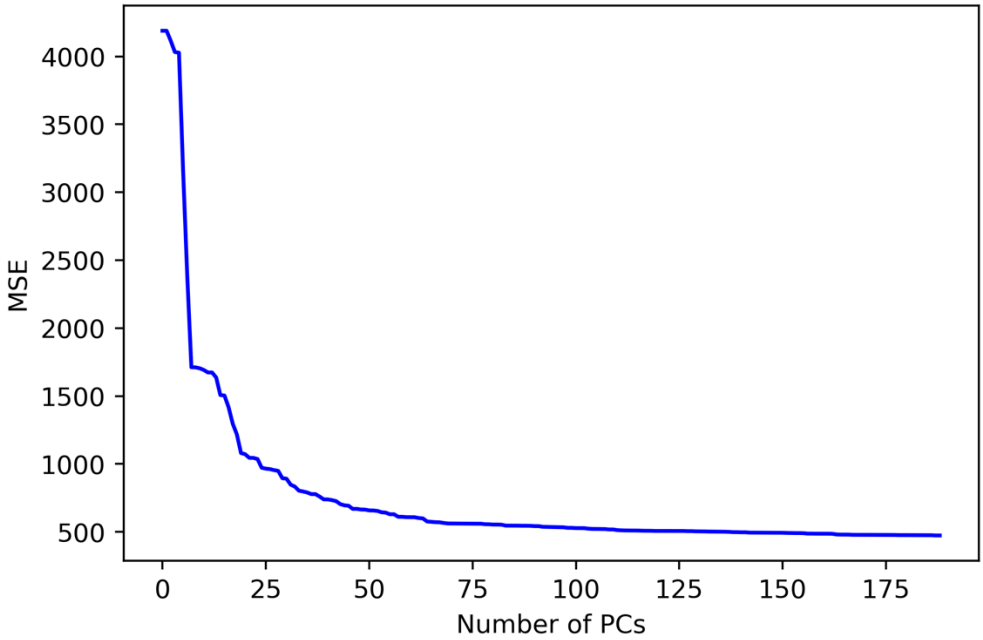


fig 6. the MSE of reconstruction faces versus the number of principal components.

When using 25 PCs, we can obtain an ambiguous smiling face, which looks like the original picture. Then when adding up to 60 PCs, it looks similar to its original face. Similarly, before 75 PCs, the MSE continuously decline as well as adding more PCs. However, after 75 PCs, the MSE doesn't keep going down, and the picture doesn't look any different or add more details comparing to the pervious one. Meanwhile, the error is larger than fig 4.

5. Reconstruct one of 10 individuals' neutral image by using different number of principal components and plot the MSE of reconstruction versus the number of principal components.

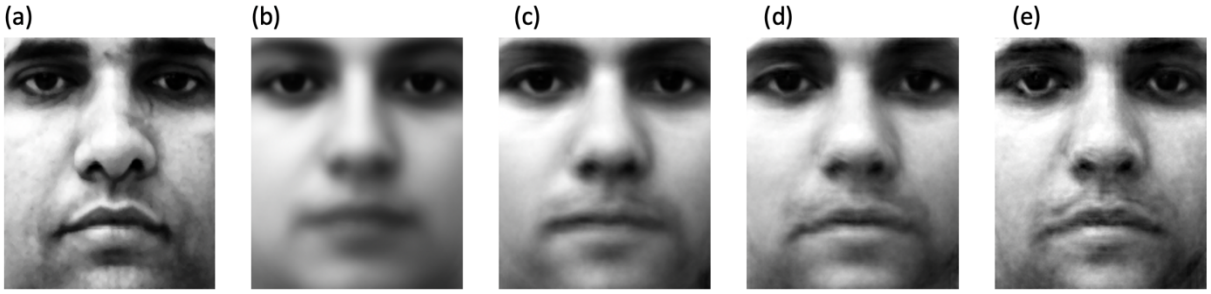


fig 7. (a) the original face; (b) 1 PCs, (c) 25 PCs; (d) 60 PCs; (e) 189 PCs;

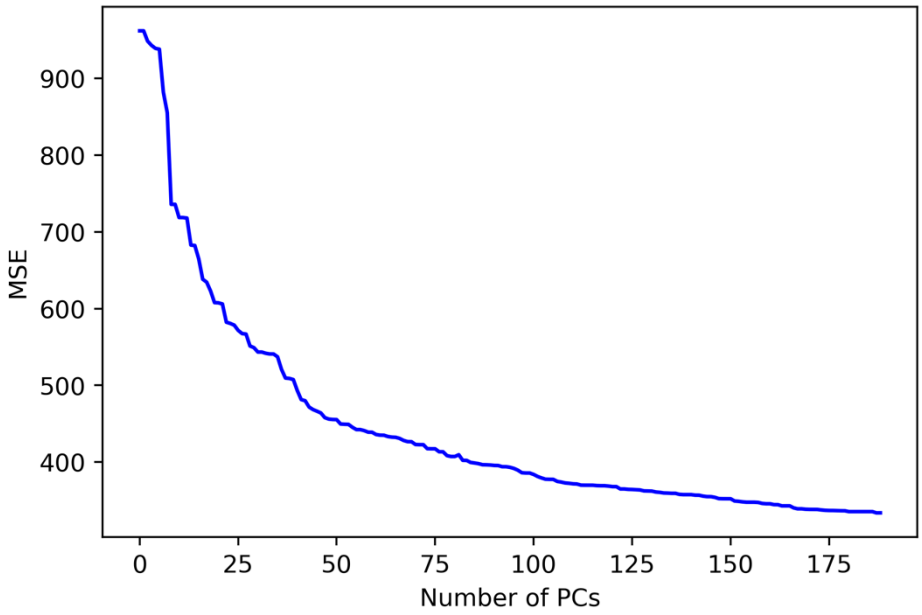


fig 8. the MSE of reconstruction faces versus the number of principal components.

This model performs well in the test individuals' neutral images and the error is smaller than that in fig6. Similarly, using 60 PCs is enough to reconstruct pictures with many details. However, using all PCS does not greatly improve results, and MSE cannot be reduced to near zero.

6. Reconstruct a cat image by using different number of principal components and plot the MSE of reconstruction versus the number of principal components.

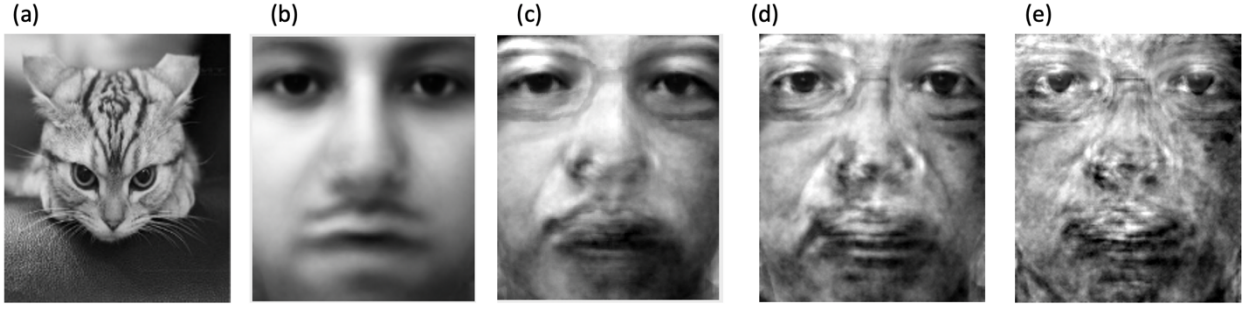


fig 9. (a) the original face; (b) 1 PCs, (c) 25 PCs; (d) 60 PCs; (e) 189 PCs;

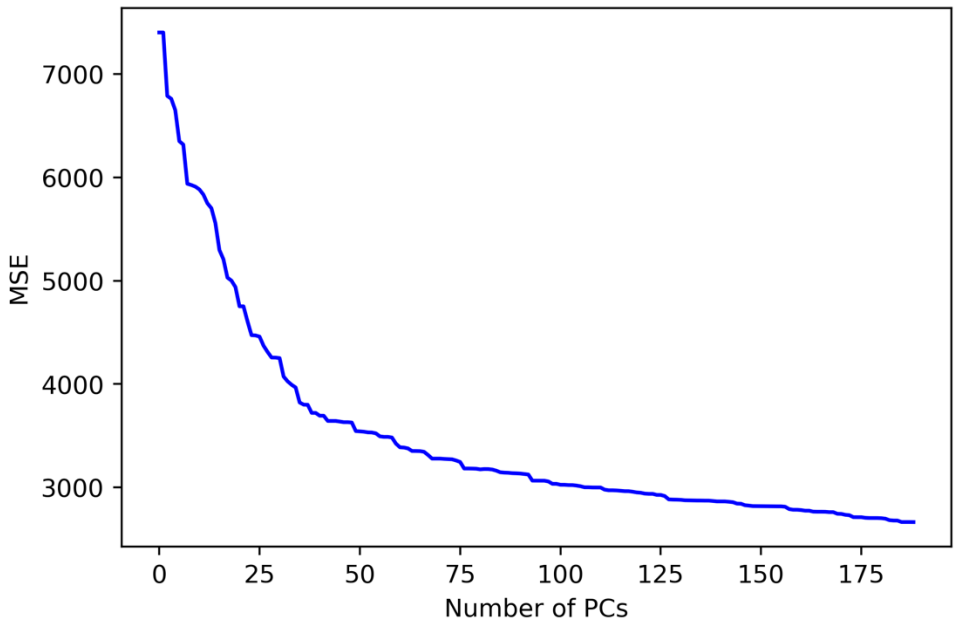


fig 10. the MSE of reconstruction faces versus the number of principal components.

This model performs bad in reconstructing a cat image. Meanwhile, the error is greatly higher than those in fig 4, fig6 and fig 8 even if using all PCS. The PCA method doesn't work when non-human images appear.

7. Reconstruct three rotated images of 81a (which test before) by using all PCs. The rotated degree is 90,180, and 270 degree.

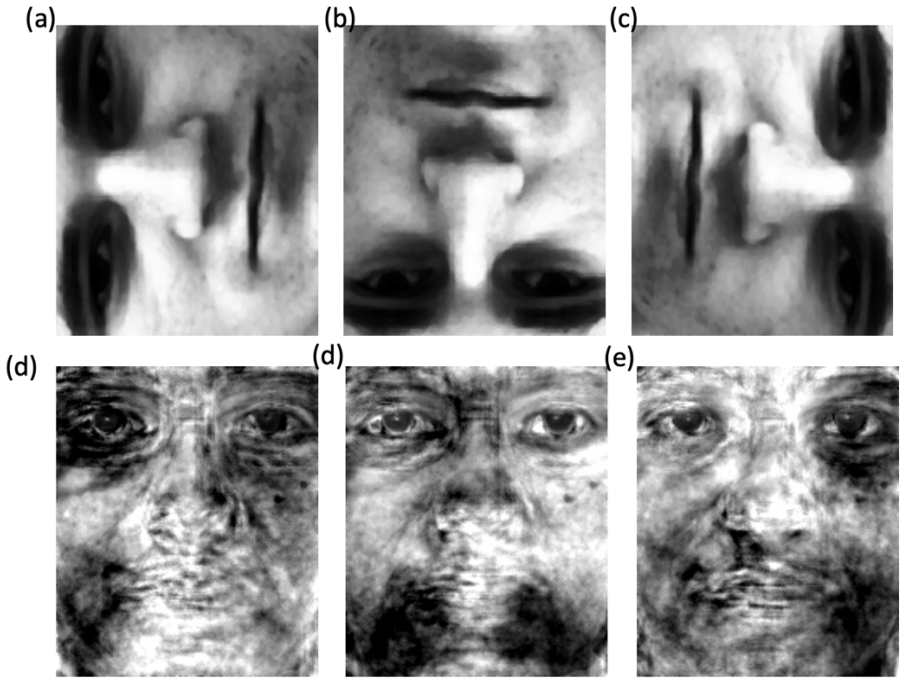


fig 11. (a) the original rotated image with 90 degree; (b) the original rotated image with 180 degree, (c) the original rotated image with 270 degree; (d) the reconstructed images by using all PCs(90 degree); (e) the reconstructed images by using all PCs(180 degree); (f) the reconstructed images by using all PCs(270 degree)

This model performs bad in reconstructing rotated images. Meanwhile, the error is greatly high even if using all PCS. The MSE of reconstructing rotated image with 90 degree is 2626.3126, and with 180 degree is 1841.4124, and with 180 degree is 2606.71578.

Conclusion

The PCA method performance well when the image has already appeared in the training set. It can also be used to reconstruct faces with different expression or the new

person's face. However, it doesn't work for non-human images or rotated images.

[1] Hsu, Rein-Lien, Mohamed Abdel-Mottaleb, and Anil K. Jain. "Face detection in color images." IEEE transactions on pattern analysis and machine intelligence 24.5 (2002): 696-706.
[2] Turk, Matthew A., and Alex P. Pentland. "Face recognition using eigenfaces." Proceedings. 1991 IEEE computer society conference on computer vision and pattern recognition. IEEE Computer Society, 1991.

In [2]:
```
pip install -i https://mirrors.aliyun.com/pypi/simple/ opencv-python
```
```
Looking in indexes: https://mirrors.aliyun.com/pypi/simple/
Collecting opencv-python
  Downloading https://mirrors.aliyun.com/pypi/packages/36/0f/b17c3c95aba11d7e7b79172a4a9f42c7
80723ce76bfbf6450b023bb277b2/opencv_python-4.4.0.46-cp38-cp38-macosx_10_13_x86_64.whl (52.4 M
B)
     |████████████████████████████████| 52.4 MB 2.5 MB/s eta 0:00:012     |██████████
| 16.4 MB 8.3 MB/s eta 0:00:05
Requirement already satisfied: numpy>=1.17.3 in /Applications/anaconda3/lib/python3.8/site-pa
ckages (from opencv-python) (1.18.5)
Installing collected packages: opencv-python
Successfully installed opencv-python-4.4.0.46
Note: you may need to restart the kernel to use updated packages.
```

In [1]:
```
import numpy as np
from PIL import Image
import cv2
import os
```

In [2]:
```
def readimage(imgfile):
    image = cv2.imread(imgfile, cv2.IMREAD_GRAYSCALE)
    return image
```

In [822]:
```
def readimage2(imgfile):
    image = cv2.imread(imgfile, cv2.IMREAD_GRAYSCALE)
    p0=cv2.resize(image,(162,193),interpolation=cv2.INTER_CUBIC)
    return p0
```

In [3]:
```
img1=readimage('frontalimages_spatiallynormalized_cropped_equalized_part1/1a.jpg')
```

In [4]:
```
type(img1) #first individual’s neutral expression image
```
Out[4]:
```
numpy.ndarray
```

In [5]:
```
print(img1)
print(img1.shape)  #The dimension of the image
```
```
[[254 252 252 ... 175 169 164]
 [255 253 253 ... 168 176 164]
 [254 254 254 ... 170 175 175]
 ...
 [104 100  99 ...  65  72  77]
 [ 96 100 100 ...  65  70  70]
 [ 96 100 100 ...  65  70  70]]
(193, 162)
```

In [713]:
```
def load_image_file(imagepath):
    Trainsets = os.listdir(imagepath)
    Num = len(Trainsets) #190 or 10
    img_arr=list()
    for i in range(1,Num):
        img = cv2.imread(imagepath+'/'+str(i)+'a'+'.jpg',cv2.IMREAD_GRAYSCALE) #read image
        img = img.reshape(img.size,1)
        img_arr.append(img)
    return img_arr
```

In [714]:
```
img2=load_image_file('netural_190')
img3=np.array(img2)
#img3.reshape(img3.shape[0],img3.shape[1])
img3=np.mat(img3).T
#img3=img3.reshape(img3.shape[1],img3.shape[0])
```

In [293]:
```
len(img2[80])
```
Out[293]:
```
31266
```

In [315]:
```
img2[80]
```
Out[315]:
```
array([[240],
       [240],
       [237],
       ...,
       [108],
       [108],
       [130]], dtype=uint8)
```

In [144]:
```
print(img3.shape)
```
```
(31266, 189)
```

In [145]:
```
imgmean=np.mean(img3, axis=1)
#imgmean=np.mat(imgmean).T
img4=img3-imgmean
#img4=np.mat(imgmean).T
```

In [396]:
```
imgmean
```
Out[396]:
```
matrix([[154.3968254 ],
        [153.61375661],
        [152.14814815],
        ...,
        [ 60.67195767],
        [ 61.56613757],
        [ 62.55026455]])
```

In [146]:
```
print(len(imgmean))
```
```
31266
```

In [90]:
```
covi=np.cov(img4,rowvar=False)
eigenValue, eigenVec=np.linalg.eig(covi) #or calculate matrix A.T*A
```

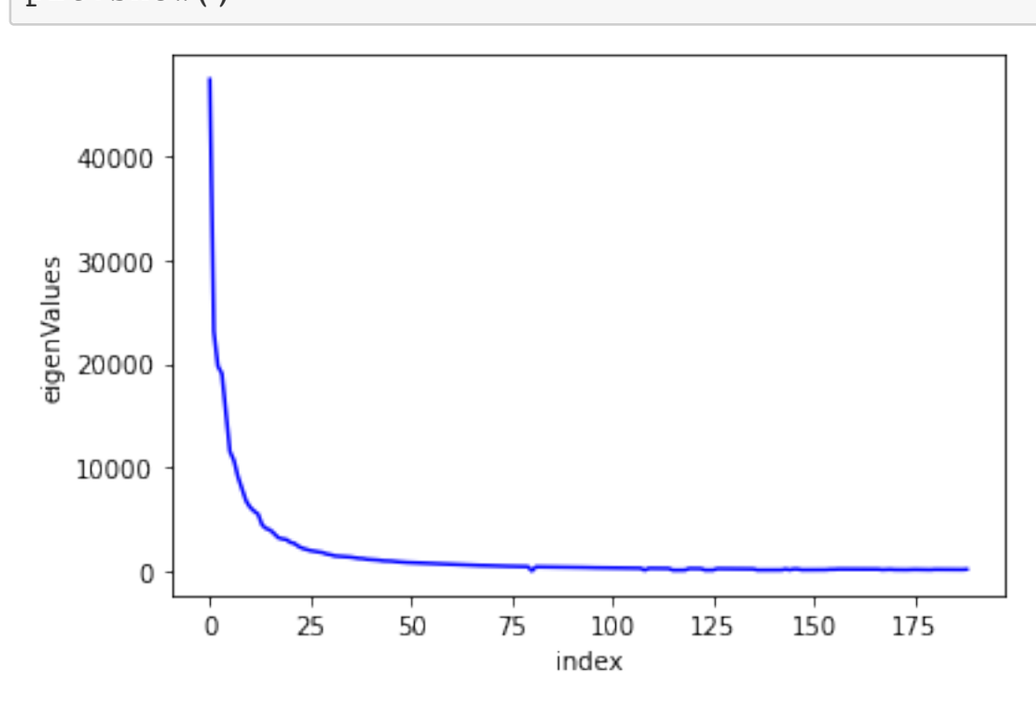
In [710]:
```
covi.shape
```
Out[710]:
```
(189, 189)
```

In [91]:
```
eigenValue
```
Out[91]:
```
array([4.74023159e+04, 2.30906876e+04, 1.97602697e+04, 1.90252851e+04,
       1.51568321e+04, 1.14970081e+04, 1.06458590e+04, 9.06079755e+03,
       7.88636609e+03, 6.75549878e+03, 6.13291663e+03, 5.76209369e+03,
       5.46585018e+03, 4.42177989e+03, 4.08577091e+03, 3.93475590e+03,
       3.58434751e+03, 3.19691299e+03, 3.08594173e+03, 3.02137984e+03,
       2.75407871e+03, 2.64769332e+03, 2.36779551e+03, 2.18030405e+03,
       2.06162324e+03, 1.95443108e+03, 1.89219405e+03, 1.82562232e+03,
       1.76519195e+03, 1.62706454e+03, 1.56259769e+03, 1.44039122e+03,
       1.41364326e+03, 1.37896792e+03, 1.36297377e+03, 1.32686733e+03,
       1.26441154e+03, 1.20713490e+03, 1.18561321e+03, 1.11695556e+03,
       1.09647400e+03, 1.06794788e+03, 1.03112455e+03, 9.74018220e+02,
       9.64940439e+02, 9.44434262e+02, 8.95527650e+02, 8.67125623e+02,
       8.27583655e+02, 8.09179909e+02, 7.92402723e+02, 7.82958939e+02,
       7.65256839e+02, 7.48058145e+02, 7.25278308e+02, 7.03483866e+02,
       6.75778631e+02, 6.70097318e+02, 6.51637001e+02, 6.32161176e+02,
       6.23094678e+02, 6.00983302e+02, 5.92574883e+02, 5.75882613e+02,
       5.66326477e+02, 5.50562651e+02, 5.15886556e+02, 5.26620692e+02,
       5.25251179e+02, 5.08751201e+02, 4.92031815e+02, 4.87224901e+02,
       4.74790287e+02, 4.64166464e+02, 4.46465289e+02, 4.38828477e+02,
       4.29792573e+02, 4.26338446e+02, 4.18456369e+02, 4.06960782e+02,
       3.44368626e-13, 3.93810919e+02, 3.90130086e+02, 3.84607674e+02,
       3.76525484e+02, 3.68022140e+02, 3.62742430e+02, 3.65554277e+02,
       3.48948145e+02, 3.52350876e+02, 3.40256646e+02, 3.37148794e+02,
       3.32585210e+02, 3.27995506e+02, 3.12101159e+02, 3.07375664e+02,
       3.01615858e+02, 2.95024920e+02, 2.93873963e+02, 2.88685530e+02,
       2.69399454e+02, 2.81073051e+02, 2.77933966e+02, 2.67775332e+02,
       2.64212961e+02, 2.59857421e+02, 2.55177445e+02, 2.50836395e+02,
       6.16558729e+01, 2.47478296e+02, 2.43811384e+02, 2.45369284e+02,
       2.37484163e+02, 2.31777954e+02, 2.29875735e+02, 6.75538700e+01,
       7.00357000e+01, 7.17248876e+01, 7.37896620e+01, 2.26816438e+02,
       2.23630662e+02, 2.16975216e+02, 2.15254938e+02, 7.66173286e+01,
       7.83105610e+01, 7.94254185e+01, 2.14084848e+02, 2.08841755e+02,
       2.08042508e+02, 2.00987717e+02, 2.01971471e+02, 1.98154603e+02,
       1.91864591e+02, 1.88037247e+02, 1.85283515e+02, 1.83400391e+02,
       8.16526919e+01, 8.36064064e+01, 8.56223975e+01, 8.67156596e+01,
       9.04411378e+01, 8.97361123e+01, 8.88309195e+01, 1.81085781e+02,
       9.24397647e+01, 1.78409915e+02, 1.75396181e+02, 9.64864989e+01,
       9.52929175e+01, 9.91364197e+01, 9.99852648e+01, 1.02942395e+02,
       1.05773346e+02, 1.07467012e+02, 1.43731081e+02, 1.42030556e+02,
       1.69812865e+02, 1.68703188e+02, 1.67603965e+02, 1.73272822e+02,
       1.49531411e+02, 1.64252778e+02, 1.55439317e+02, 1.61554284e+02,
       1.60055254e+02, 1.57122344e+02, 1.58769677e+02, 1.09514823e+02,
       1.39374056e+02, 1.48249365e+02, 1.13952555e+02, 1.10585616e+02,
       1.11905239e+02, 1.03564628e+02, 1.30828441e+02, 1.33584258e+02,
       1.24933905e+02, 1.19705687e+02, 1.17007969e+02, 1.06271505e+02,
       1.37881471e+02, 1.34048703e+02, 1.28773602e+02, 1.28176964e+02,
       1.17373135e+02, 1.18766913e+02, 1.23194903e+02, 1.23113473e+02,
       1.52362553e+02])
```

In [92]:
```
eigenValue[:5]
```
Out[92]:
```
array([47402.31589263, 23090.6875652 , 19760.26971613, 19025.2850605 ,
       15156.83209669])
```

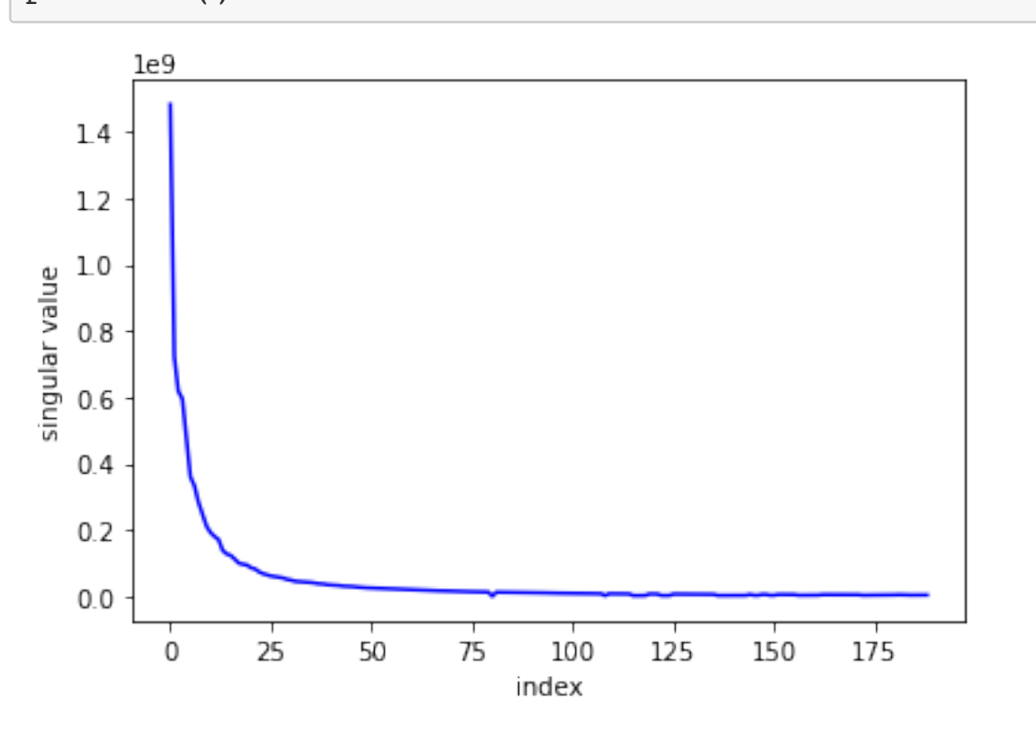
In [93]:
```
import matplotlib.pyplot as plt
```

In [94]:
```
plt.plot((eigenValue), '-', label='eignvalue', c='b')
plt.xlabel('index')
plt.ylabel('eigenValues')
#plt.title('threshold of SVD++')
plt.savefig('pca1.png',dpi=400,bbox_inches='tight')
plt.show()
```



In [586]:
```
eigenValue2, eigenVec2=np.linalg.eig((img4.T)*img4)
```

In [395]:
```
plt.plot((eigenValue2), '-', label='eignvalue', c='b')
plt.xlabel('index')
plt.ylabel('singular value')
#plt.title('threshold of SVD++')
plt.savefig('pca2.png',dpi=400,bbox_inches='tight')
plt.show()
```



In [147]:
```
import matplotlib.image as mp
meanface=np.array(imgmean).reshape(193, 162)
meanface2 = Image.fromarray(meanface)
mp.imsave('meanface.png',meanface2)
meanface2.show()
```

In [258]:
```
egin1=np.dot(img4,eigenVec) #A*eigenVec
egin2=np.dot(egin1,eigenVec.T)+imgmean
egin3=np.mat(egin2).T
face1=np.array(egin3[80]).reshape(193, 162)
face1_img = Image.fromarray(face1) #all PCs
face1_img.show()
```

In [619]:
```
index=np.argsort(-eigenValue).tolist()
```

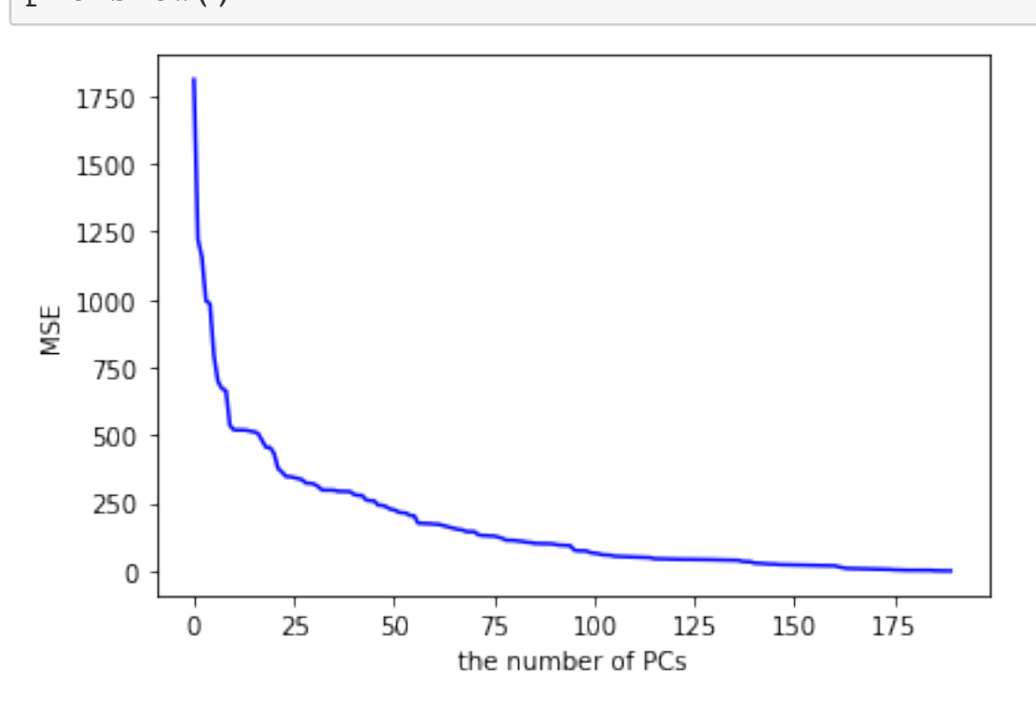
In [291]:
```
def MSE(predictions, labels):
    differences = [(x-y)**2 for x,y in zip(predictions,labels)]
    return sum(differences) / len(differences)
```

In [345]:
```
img81a=readimage('frontalimages_spatiallynormalized_cropped_equalized_part1/81a.jpg')
img81a= img81a.reshape(img81a.size,1)
labels=[]
for i in img81a:
    a=i.tolist()
    labels.append(a[0])
```

In [608]:
```
mse=list()
PCs=list()
for i in range(190):
    n_index=index[:i]
    PCs.append(i)
    n_eigenVec=eigenVec[:, n_index] ###
    lowd=np.dot(img4,n_eigenVec)
    highd=np.dot(lowd,n_eigenVec.T)+imgmean
    egin4=np.mat(highd).T
    pred=np.array(egin4[80])
    mse.append(MSE(pred[0],labels))

#face2=np.array(egin4[80]).reshape(193, 162)
#face2_img = Image.fromarray(face2) #all PCs
#face2_img.show()
```

In [610]:
```
plt.plot((mse), '-', label='MSE', c='b')
plt.xlabel('the number of PCs')
plt.ylabel('MSE')
#plt.title('threshold of SVD++')
plt.savefig('pca3.png',dpi=400,bbox_inches='tight')
plt.show()
```



In [538]:
```
img30b=readimage('frontalimages_spatiallynormalized_cropped_equalized_part1/30b.jpg')
img30b= img30b.reshape(img30b.size,1)
labels2=[]
for i in img30b:
    a=i.tolist()
    labels2.append(a[0])
```

In [702]:
```
img30b.shape
```
Out[702]:
```
(31266, 1)
```

In [799]:
```
AV=np.dot(img4,eigenVec)# img4=trainset(31266,189)-mean(31266,1)
AV_norm=AV/np.linalg.norm(AV, axis=0, keepdims=True)
eigenfaces=AV_norm[:,1:189] # 1,25,60,120,189 pcs
#Proj_space=eigenfaces*np.mat((eigenfaces.T)*eigenfaces)*(eigenfaces.T)
diff=img30b-imgmean #test image--img30b(31266, 1)
#Proj_faces=(Proj_space.T)*diff
```

In [800]:
```
weigh=np.dot(eigenfaces.T,diff)
Proj_faces=np.dot(eigenfaces,weigh)+imgmean
```

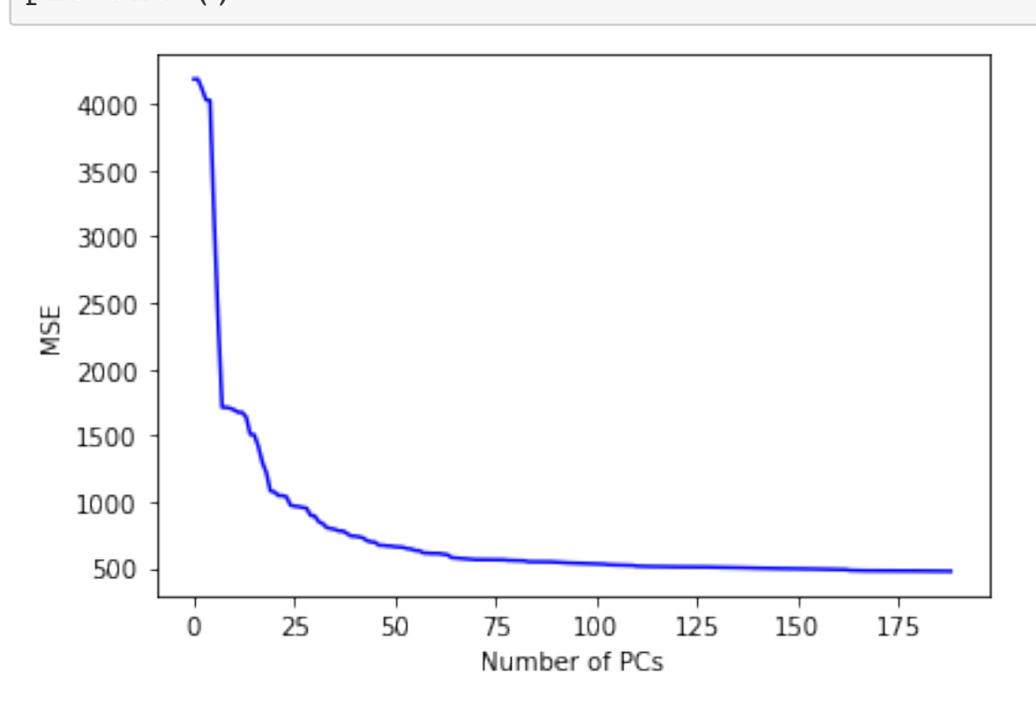
In [801]:
```
Proj_faces
```
Out[801]:
```
matrix([[199.65507694],
        [200.19437954],
        [196.82397102],
        ...,
        [ 24.10907602],
        [ 19.5020284 ],
        [ 37.05702192]])
```

In [802]:
```
face4=np.array(Proj_faces).reshape(193, 162)
face4_img = Image.fromarray(face4) #all PCs
face4_img.show()
```

In [ ]:
```
#mse
```

In [803]:
```
mse3=list()
for i in range(189):
    eigenfaces=AV_norm[:,1:i] # 1,25,60,120,189 pcs
    weigh=np.dot(eigenfaces.T,diff)
    Proj_faces=np.dot(eigenfaces,weigh)+imgmean
    pred=np.array(Proj_faces.T)
    mse3.append(MSE(pred[0],labels2))
```

In [804]:
```
plt.plot((mse3), '-', label='MSE', c='b')
plt.xlabel('Number of PCs')
plt.ylabel('MSE')
#plt.title('threshold of SVD++')
plt.savefig('pca4.png',dpi=400,bbox_inches='tight')
plt.show()
```



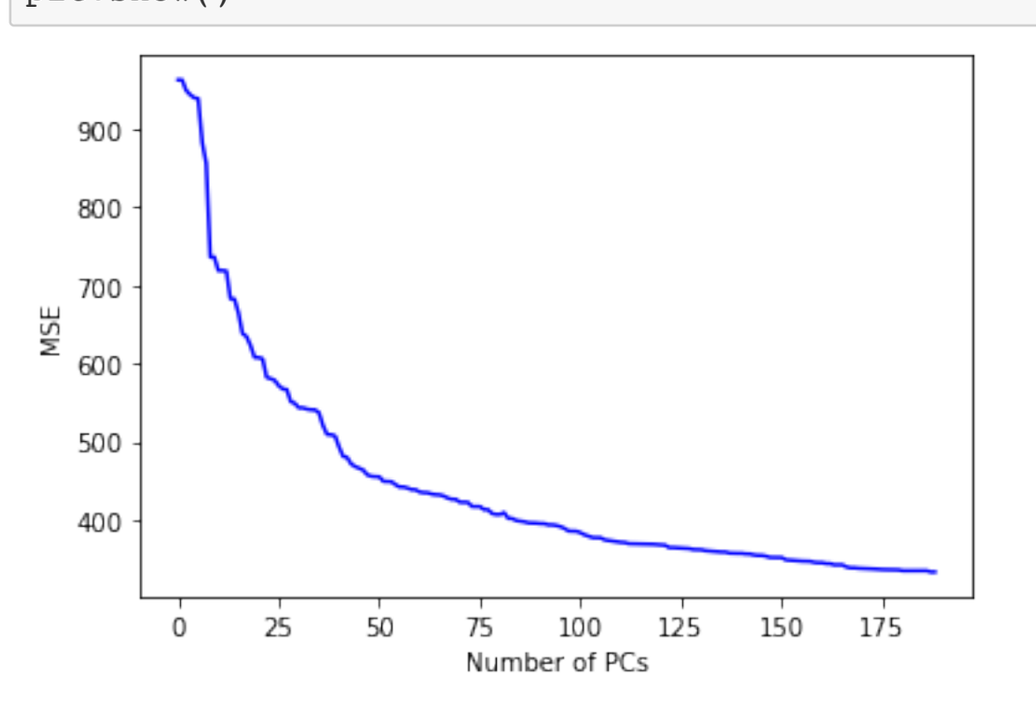
In [805]:
```
#one of the other 10 individuals’ neutral expression
```

In [808]:
```
img191a=readimage('test_netural_10/191a.jpg')
img191a= img191a.reshape(img191a.size,1)
labels3=[]
for i in img191a:
    a=i.tolist()
    labels3.append(a[0])
```

In [815]:
```
AV=np.dot(img4,eigenVec)# img4=trainset(31266,189)-mean(31266,1)
AV_norm=AV/np.linalg.norm(AV, axis=0, keepdims=True)
eigenfaces=AV_norm[:,1:2] # 1,25,60,120,189 pcs
diff=img191a-imgmean #test image--img30b(31266, 1)
weigh=np.dot(eigenfaces.T,diff)
Proj_faces=np.dot(eigenfaces,weigh)+imgmean
face5=np.array(Proj_faces).reshape(193, 162)
face5_img = Image.fromarray(face5) #all PCs
face5_img.show()
```

In [816]:
```
#mse
mse4=list()
for i in range(189):
    eigenfaces=AV_norm[:,1:i] # 1,25,60,120,189 pcs
    weigh=np.dot(eigenfaces.T,diff)
    Proj_faces=np.dot(eigenfaces,weigh)+imgmean
    pred=np.array(Proj_faces.T)
    mse4.append(MSE(pred[0],labels3))
```

In [817]:
```
plt.plot((mse4), '-', label='MSE', c='b')
plt.xlabel('Number of PCs')
plt.ylabel('MSE')
#plt.title('threshold of SVD++')
plt.savefig('pca5.png',dpi=400,bbox_inches='tight')
plt.show()
```



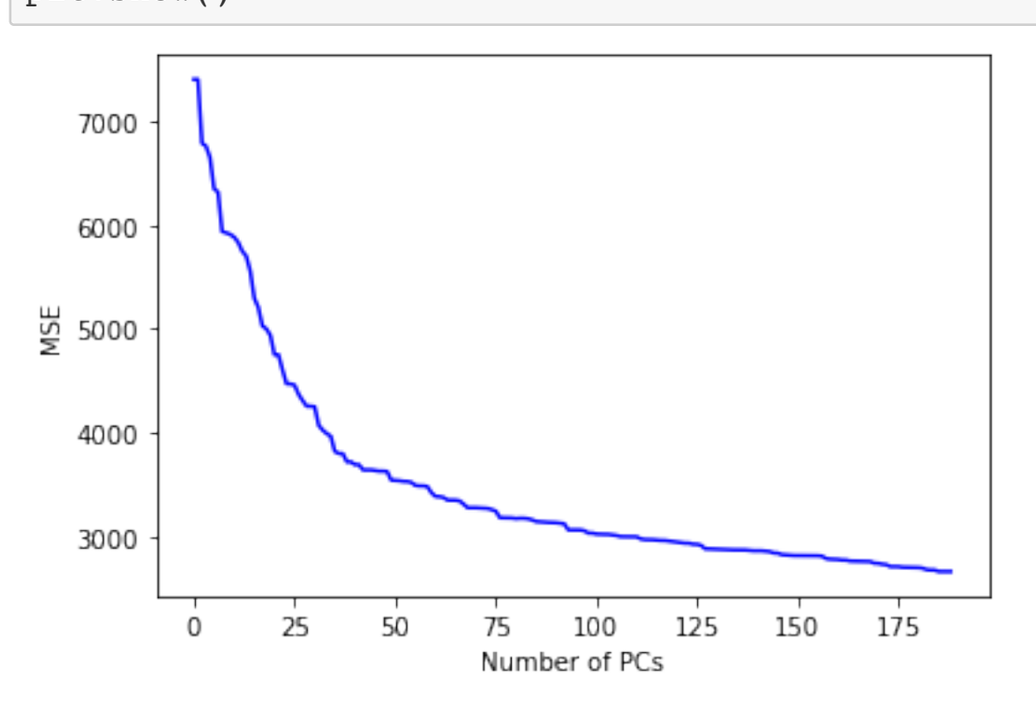
In [835]:
```
#non-human image
```

In [823]:
```
imgcat=readimage2('face/1a.png')
cat_img = Image.fromarray(imgcat) #all PCs
cat_img.show()
imgcat= imgcat.reshape(imgcat.size,1)
labels4=[]
for i in imgcat:
    a=i.tolist()
    labels4.append(a[0])
```

In [828]:
```
AV=np.dot(img4,eigenVec)# img4=trainset(31266,189)-mean(31266,1)
AV_norm=AV/np.linalg.norm(AV, axis=0, keepdims=True)
eigenfaces=AV_norm[:,1:25] # 1,25,60,120,189 pcs
diff=imgcat-imgmean #test image--img30b(31266, 1)
weigh=np.dot(eigenfaces.T,diff)
Proj_faces=np.dot(eigenfaces,weigh)+imgmean
face5=np.array(Proj_faces).reshape(193, 162)
face5_img = Image.fromarray(face5) #all PCs
face5_img.show()
```

In [831]:
```
#mse
mse5=list()
for i in range(189):
    eigenfaces=AV_norm[:,1:i] # 1,25,60,120,189 pcs
    weigh=np.dot(eigenfaces.T,diff)
    Proj_faces=np.dot(eigenfaces,weigh)+imgmean
    pred=np.array(Proj_faces.T)
    mse5.append(MSE(pred[0],labels4))
```

In [833]:
```
plt.plot((mse5), '-', label='MSE', c='b')
plt.xlabel('Number of PCs')
plt.ylabel('MSE')
#plt.title('threshold of SVD++')
plt.savefig('pca6.png',dpi=400,bbox_inches='tight')
plt.show()
```



roate image

In [842]:
```
from PIL import Image
```

In [861]:
```
im = Image.open('frontalimages_spatiallynormalized_cropped_equalized_part1/81a.jpg')
rotated1 = im.transpose(Image.ROTATE_270)
out=rotated1.resize((162,193))
out.show()
```

In [855]:
```
r_81a=readimage('face/81a_r1.png')
#imgcat= imgcat.reshape(imgcat.size,1)
r_81a= r_81a.reshape(r_81a.size,1)
labels6=[]
for i in r_81a:
    a=i.tolist()
    labels6.append(a[0])
```

In [856]:
```
AV=np.dot(img4,eigenVec)# img4=trainset(31266,189)-mean(31266,1)
AV_norm=AV/np.linalg.norm(AV, axis=0, keepdims=True)
eigenfaces=AV_norm[:,1:189] # 1,25,60,120,189 pcs
diff=r_81a-imgmean #test image--img30b(31266, 1)
weigh=np.dot(eigenfaces.T,diff)
Proj_faces=np.dot(eigenfaces,weigh)+imgmean
face5=np.array(Proj_faces).reshape(193, 162)
face5_img = Image.fromarray(face5) #all PCs
face5_img.show()
```

In [857]:
```
pred=np.array(Proj_faces.T)
MSE(pred[0],labels6)
```
Out[857]:
```
2626.31263392082
```

In [858]:
```
r2_81a=readimage('face/81a_r2.png')
#imgcat= imgcat.reshape(imgcat.size,1)
r2_81a= r2_81a.reshape(r2_81a.size,1)
labels7=[]
for i in r2_81a:
    a=i.tolist()
    labels7.append(a[0])
```

In [860]:
```
AV=np.dot(img4,eigenVec)# img4=trainset(31266,189)-mean(31266,1)
AV_norm=AV/np.linalg.norm(AV, axis=0, keepdims=True)
eigenfaces=AV_norm[:,1:189] # 1,25,60,120,189 pcs
diff=r2_81a-imgmean #test image--img30b(31266, 1)
weigh=np.dot(eigenfaces.T,diff)
Proj_faces=np.dot(eigenfaces,weigh)+imgmean
face5=np.array(Proj_faces).reshape(193, 162)
face5_img = Image.fromarray(face5) #all PCs
face5_img.show()
pred=np.array(Proj_faces.T)
print(MSE(pred[0],labels7))
```
```
1841.4124624760848
```

In [864]:
```
r3_81a=readimage('face/81a_r3.png')
#imgcat= imgcat.reshape(imgcat.size,1)
r3_81a= r3_81a.reshape(r3_81a.size,1)
labels8=[]
for i in r3_81a:
    a=i.tolist()
    labels8.append(a[0])
```

In [865]:
```
AV=np.dot(img4,eigenVec)# img4=trainset(31266,189)-mean(31266,1)
AV_norm=AV/np.linalg.norm(AV, axis=0, keepdims=True)
eigenfaces=AV_norm[:,1:189] # 1,25,60,120,189 pcs
diff=r3_81a-imgmean #test image--img30b(31266, 1)
weigh=np.dot(eigenfaces.T,diff)
Proj_faces=np.dot(eigenfaces,weigh)+imgmean
face5=np.array(Proj_faces).reshape(193, 162)
face5_img = Image.fromarray(face5) #all PCs
face5_img.show()
pred=np.array(Proj_faces.T)
print(MSE(pred[0],labels8))
```
```
2606.715796127746
```

In [834]:
```
n_index=index[:60]
n_eigenVec=eigenVec[:, n_index]
eignface=np.dot(img4,n_eigenVec)
diff=img81a-imgmean
w=list()
for i in range(len(n_index)):
    eigenVec_i=eigenVec[i]
    eign_face1=np.dot(img4,eigenVec_i)
    norm=1/(np.linalg.norm(eign_face1))
    eign_face2=np.dot(eign_face1,norm) #unity
    weigh=eign_face2*diff
    w.append(weigh*eign_face2)
```

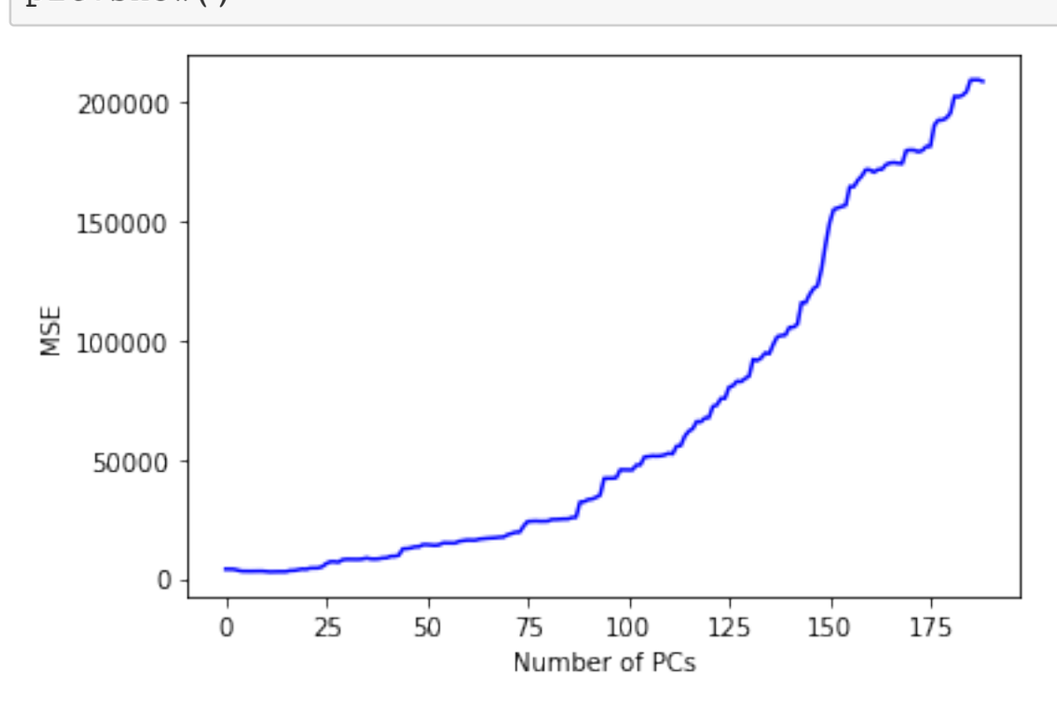
In [630]:
```
face4=np.array(Proj_faces).reshape(193, 162)
face4_img = Image.fromarray(face4) #all PCs
face4_img.show()
```

In [738]:
```
def reg_face(diff,n_index):
    w=list()
    for i in range(len(n_index)):
        V_i=eigenVec[i]
        AVi=np.dot(img4,V_i)
        #print(AVi)
        norm=np.sum(AVi**2, axis=0, keepdims=True)**0.5
        #norm=1/(np.linalg.norm(AVi, ))
        #norm=1/(int(AVi*AVi.T))**(1/2)
        AVi_unit=AVi/norm #unity
        print(AVi_unit)
        weigh=np.dot(AVi_unit,diff)
        wiui=np.dot(weigh,AVi_unit)
        w.append(wiui)
    return sum(w)
```

In [669]:
```
def reg_face2(diff):
    w=list()
    for i in range(189):
        V_i=eigenVec[i]
        AVi=np.dot(img4,V_i)
        AVi_unit=AVi/np.linalg.norm(AVi)
        #AVi_unit=np.dot(AVi,norm) unity
        weigh=(AVi_unit)*diff
        w.append(weigh*AVi_unit)
    return w
```

In [670]:
```
#calculate MSE
mse2=list()
diff2=img30b-imgmean
for i in range(189):
    a=sum(reg_face2(diff2)[:i+1])
    proj=a.T+imgmean
    pred=np.array(proj.T)
    mse2.append(MSE(pred[0],labels2))
```

In [671]:
```
plt.plot((mse2), '-', label='MSE', c='b')
plt.xlabel('Number of PCs')
plt.ylabel('MSE')
#plt.title('threshold of SVD++')
plt.savefig('pca4.png',dpi=400,bbox_inches='tight')
plt.show()
```



In [ ]: