COMS 3251 Spring 2021: Lab 6

YOUR NAME(s): Vicente Farias

YOUR UNI(s): vf2272

Principal Component Analysis

Principal component analysis (PCA) is the process of computing orthogonal directions (uncorrelated components) of a data set that best explain the data. This process can help us better understand the descriptive features of a data set, as well as perform dimensionality reduction by removing features that provide little explanatory power.

PCA can be achieved using the singular value decomposition (SVD). Suppose that we have n data vector with p features. We first demean the data by subtracting from the mean of all the data from each data vector. We can then stack the demeaned data vectors as rows of a $n \times p$ matrix.

$$X = \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{bmatrix} - \frac{1}{n} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} (\mathbf{x}_1 + \cdots + \mathbf{x}_n)$$

Singular Value Decomposition

We now compute a singular value decomposition of the matrix:

$$X = U\Sigma V^T$$

The right singular vectors $\mathbf{v}_i$ in V (not V^T!) are the principal components of the data set. The scaled singular values $\frac{1}{n-1}\sigma_i$ are the corresponding variances along each component.

Finally, the coordinates of the original data expressed relative to the principal component basis are given by $U\Sigma$. If you are interested in the coordinate transformation of a particular data vector $\mathbf{x}_i$, you can extract the ith row of $U\Sigma$. You can also compute the coordinates of a new data vector $\mathbf{y}$ in the PC basis as $\mathbf{y}V$.

Dimensionality Reduction

We may see that the first few singular values of X are much larger relative to the others. This tells us that most of the variance in the data set occurs along the corresponding components.

We can find a low-rank approximation of the data by dropping those components that contribute little explanatory power.

A rank-r approximation of the original data set is thus given by

$$\tilde{X} = [\,\mathbf{u}_1 \quad \cdots \quad \mathbf{u}_r\,] \begin{bmatrix} \sigma_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_r \end{bmatrix} \begin{bmatrix} \mathbf{v}_1^T \\ \vdots \\ \mathbf{v}_r^T \end{bmatrix}$$

The "difference" or error between a data vector and its low-rank approximation can be found as the norm between the corresponding rows of X and $\tilde{X}$. The total error, as the sum of squared errors over all data, can be found as the Frobenius norm of the matrix $X - \tilde{X}$.

Eigenfaces

PCA can be performed on image data, and in particular on images of faces. *Eigenfaces* refer to the set of principal components derived from a data set of facial images, with the idea that human faces can be approximated as a linear combination of the eigenfaces. As with PCA in general, the set of eigenfaces generally has a lower dimensionality than the original images, giving us a method to perform classification or facial recognition.

Since we are dealing with image data, we do have to make sure that the images are standardized. This allows PCA to capture principal components due to the faces themselves, instead of irrelevant differences in lighting conditions or face alignment within the frame. Each image must also have the same pixel resolution $(r \times c)$, and we can treat a single image as one vector with $r \times c$ elements.

You will be working with a set of images provided by a lab assignment in the textbook. First upload `faces.zip` and then import the images into Python by running the code block below. The code will also display the first couple images as output.

In [1]:
```
import numpy as np
from scipy.linalg import svd
import matplotlib.pyplot as plt
import zipfile
from PIL import Image
from google.colab.patches import cv2_imshow

def display_imgvector(v):
  imgarray = v.reshape((189,166))
  img = Image.fromarray(imgarray,mode='L')
  cv2_imshow(imgarray)

imgzip = zipfile.ZipFile("faces.zip")
inflist = imgzip.infolist()
faces_array = np.empty((len(inflist),31374))
for i in range(len(inflist)):
  img = Image.open(imgzip.open(inflist[i]))
  faces_array[i,:] = np.array(img).reshape(31374)
```

```
display_imgvector(faces_array[0,:])
display_imgvector(faces_array[1,:])
```

Exercises

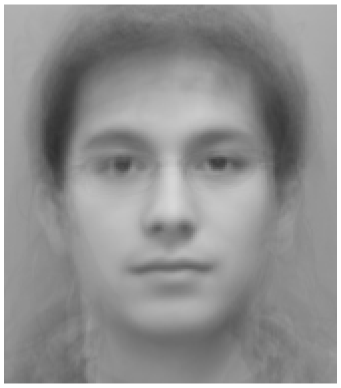
Your tasks will be to compute the principal components of this image data set, find the low-rank approximations of the images, and then use the decomposition to determine whether a new set of images contains faces.

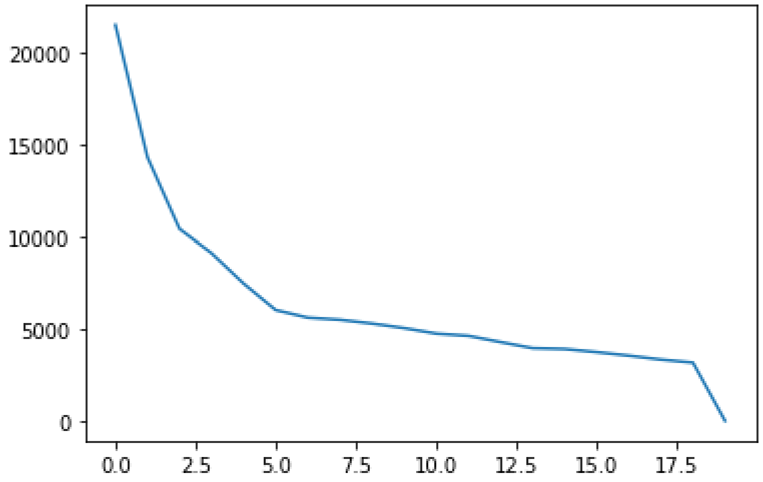
TASK 1 (15 points)

- First compute the mean image of `faces_array` defined above. Keep in mind that each image is a row of this matrix.
- Display this mean image in the code output below.
- Compute the SVD of the demeaned image data and save the matrices U, Σ, and V^T. Note that we are using SciPy's version of SVD instead of NumPy's, as the former is much more memory-efficient.
- Generate a plot of the singular values (simply use the `plot` function).

In [2]:
```
# YOUR CODE HERE
mean = np.mean(faces_array, axis=0)

display_imgvector(mean)

demeaned_array = faces_array - mean
U, s, V_T = svd(demeaned_array)
```

```
plt.plot(s)
plt.show()
```





1. Does this mean image resemble any individual image in the data set? How would you describe the facial features of the mean image?
2. From inspection of the singular values, how many principal components does this image data set have?

ENTER YOUR RESPONSES HERE

1. No, though it has the same general facial features (eyes, nose, mouth) the face does not actually resemble any specific image in the data set. The facial features appear perfectly average, though a little blurred.
2. From inspection of the singular values it looks as though this image data set has 18 principal components.

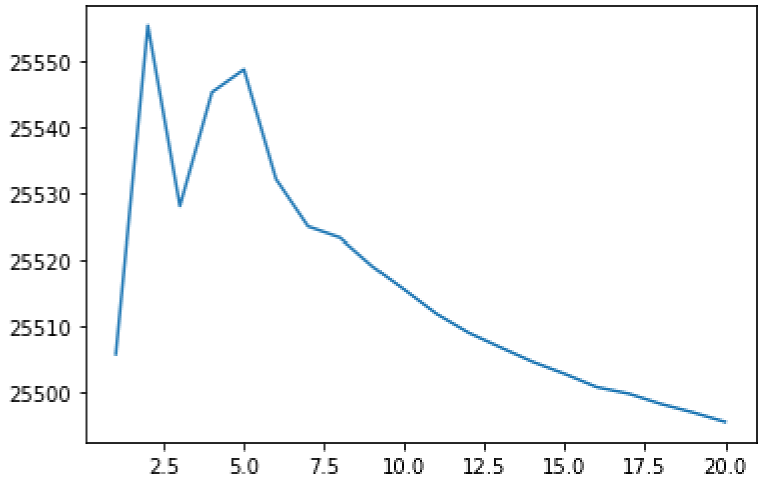
TASK 2 (15 points)

Let's now investigate low-rank approximations of our images.

- Compute and display the rank-5, rank-10 and rank-15 approximations of the first image in the data set. Note that when you display the approximated image, you will have to add your mean image to the approximation you obtain from the SVD.

- Compute the Frobenius norm of the difference between `faces_array` and its rank-k approximation for k ranging from 1 to 20. Remember to account for the mean image when taking the difference. Generate a plot of the norm values.

In [28]:
```
# YOUR CODE HERE
img1 = faces_array[0, :]
U1, s1, V_T1 = svd(img1.reshape(189, 166))

rank5 = np.matmul(U1[:, :5], np.matmul(np.diag(s1[:5]), V_T1[:5, :]))
display_imgvector(rank5 + mean.reshape(189, 166))

rank10 = np.matmul(U1[:, :10], np.matmul(np.diag(s1[:10]), V_T1[:10, :]))
display_imgvector(rank10 + mean.reshape(189, 166))

rank15 = np.matmul(U1[:, :15], np.matmul(np.diag(s1[:15]), V_T1[:15, :]))
display_imgvector(rank15 + mean.reshape(189, 166))

norms = []
for k in range(1, 21):
  approx = np.matmul(U1[:, :k], np.matmul(np.diag(s1[:k]), V_T1[:k, :]))
  norms.append(np.linalg.norm(img1.reshape(189, 166) - (approx + mean.reshape(189, 16

plt.plot([k for k in range(1, 21)], norms)
plt.show()
```



Describe your observations of how the images above vary with the different approximations.

ENTER YOUR RESPONSE HERE

The rank 5 approximation shows the very basic features of the face, and increasing the rank increases the quality and definition of each component of the face.

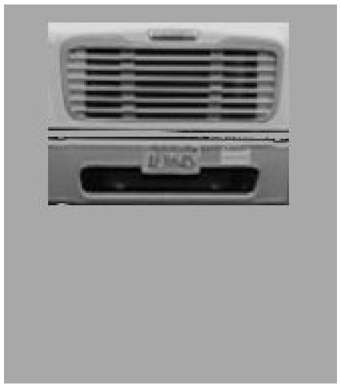
TASK 3 (10 points)

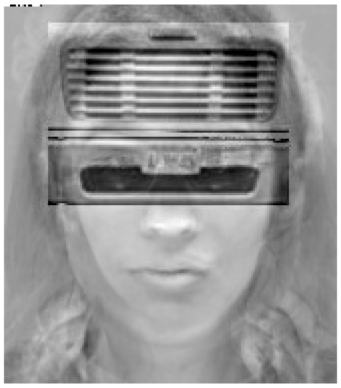
We will now examine how we can use the eigenfaces that we computed to analyze new images. First, upload `unclassified.zip` by running the code block below. As with the face images, this will create a 2D array of data for 11 images, all of the same size.

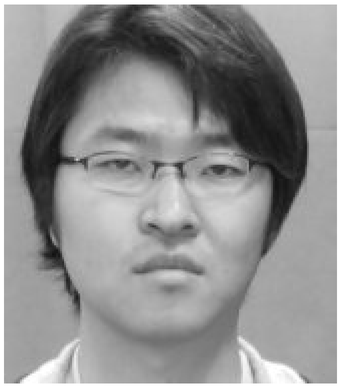
In [21]:
```
imgzip = zipfile.ZipFile("unclassified.zip")
inflist = imgzip.infolist()
img_array = np.empty((len(inflist),31374))
for i in range(len(inflist)):
  img = Image.open(imgzip.open(inflist[i])).convert('L')
  img_array[i,:] = np.array(img).reshape(31374)
```

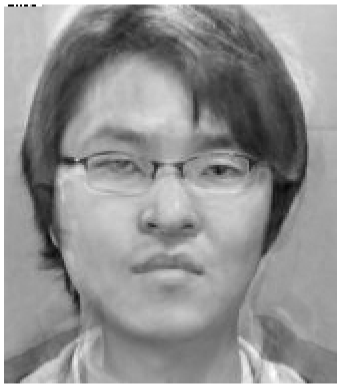
Compute the projection of each of the new images onto the eigenfaces. Then display each of the original images along with the projections (you can just call display twice, once for each version). Be careful with this computation. You will have to adjust, again, for the mean image that you computed above both before and after projecting to display the new image.

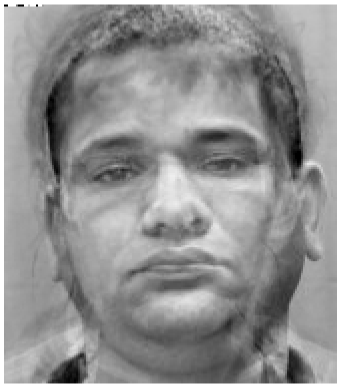
In [32]:
```
# YOUR CODE HERE
for i in img_array:
  demeaned = i - mean
  proj = np.matmul(demeaned, V_T.T)
  display_imgvector(i)
  display_imgvector(proj + mean)
  print()
```

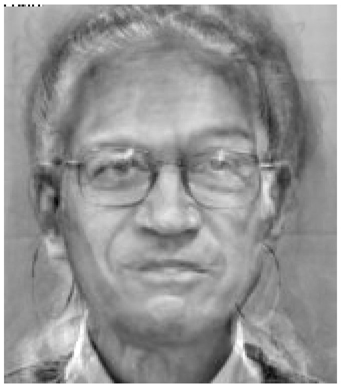


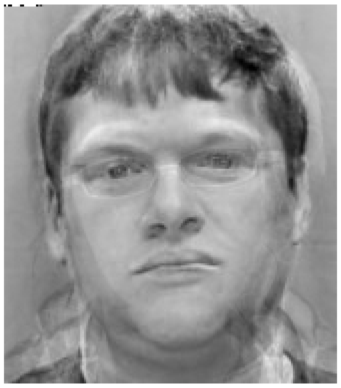


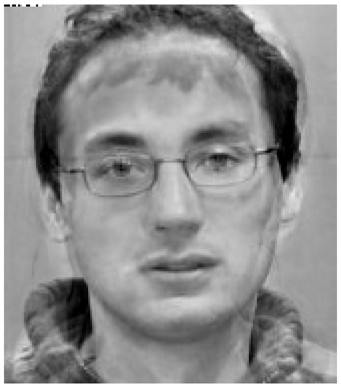


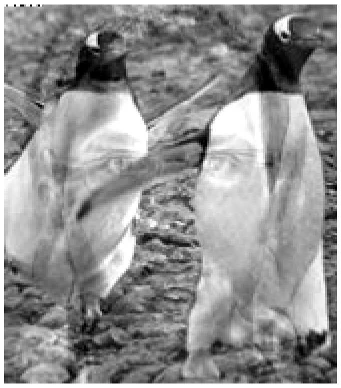


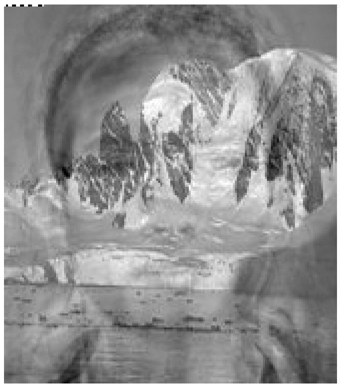


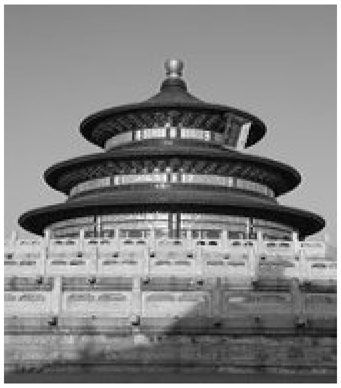


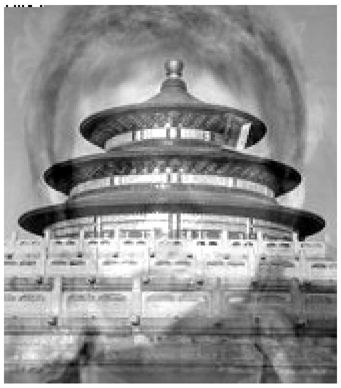


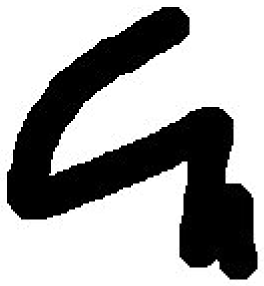


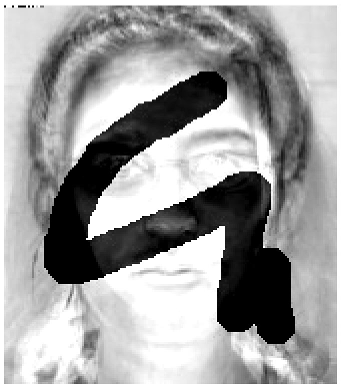












Describe what you see from the projections above. Compare and contrast the projections of the images that contain faces with those that are clearly not faces.

ENTER YOUR RESPONSES HERE

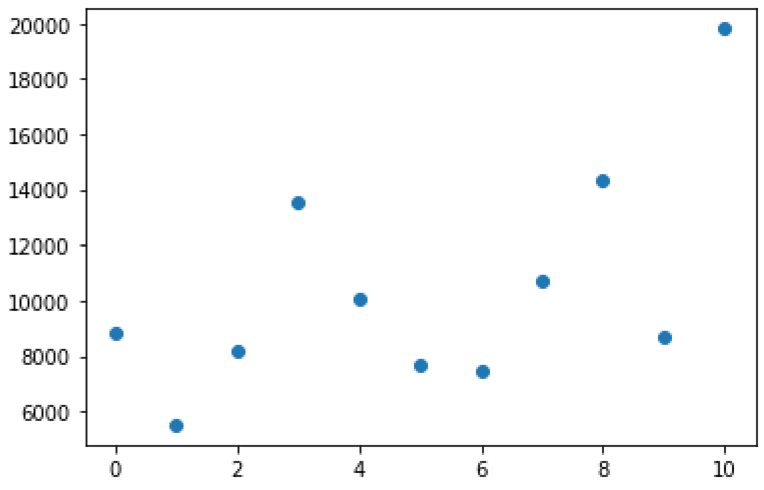
In the projections above, we see each face projected onto the basis for "face space". This results in the addition of a face-like feature to each of the images - regardless of whether or not the image is a face. For faces, the image looks relatively similar to the original face. For non-faces, the image looks like the original image but with a face superimposed on it. The face images therefore retain most of their original form, while the non-faces get converted into a completely different image.

TASK 4 (20 points)

The projections can be used to help automate the task of face recognition. Instead of visually inspecting each projection individually, we can write a simple program to make this determination. For each of the images, compute the norm of the difference between the image and the image's projection onto the eigenfaces (once again, remember to adjust for the mean image). Generate a plot of the error for each of the 11 images, starting with index 0.

In [35]:
```
# YOUR CODE HERE

norms = []
for i in img_array:
  demeaned = i - mean
  proj = np.matmul(demeaned, V_T.T)
  norms.append(np.linalg.norm(i - (proj + mean)))
```

```
plt.scatter([i for i in range(0, 11)], norms)
plt.show()
```



1. How does the error plot tell you whether an image is likely to be a face?
2. From the projection errors, which face images appear to be least "face-like"? Which non-face images appear to be the most "face-like"?

ENTER YOUR RESPONSES HERE

1. If the error is low, then the image is likely to be a face.
2. The old man with the glasses (index 3) is the face image that appears to be the least face-like. The image of the penguins (index 6) appears to be the most face-like of the non-face images.