Project 3: Real-time Face Recognition using the SVD.
Due date: 12/23/2022

Project Description

The goal of this project is to develop a **real-time** system for face recognition using dimensionality reduction techniques. The cost (in terms of time and memory) to identify a face is greatly reduced by representing face images in a reduced subspace (called the "face space"), instead of using the original "image space" of $m \times n$ pixels.

Face Recognition in the Image Space

First let us describe a simple (but slow) algorithm to recognize faces in the original image space.

A face image of $w \times h$ pixels can be regarded simply as a vector of $m = w \times h$ (real) grayscale values in the range 0:255. For simplicity, from now on a face image will be simply called a 'face' f. Then we have that $f \in \mathbb{R}^m$.

Suppose that we have a database $F = [f_1, f_2, \ldots, f_n]$ of n *known* faces, i.e. each face has an associated identity (e.g. "subject01", "subject02", etc.). There may be more than one face per subject in the database.

Given an *unknown* face f that is not in the database, an easy way to recognize f is simply to compare it to each face f_i in the database:

$$\varepsilon_i = \|f - f_i\|_2 \qquad \text{for } i = 1, \ldots, n. \tag{1}$$

We assign to the unknown face, the identity of its **closest face in the database**. That is, for the **minimum norm** ε_i, we assign identity(f) = identity(f_i).

Notice that we are implicitly assuming that the unknown face always belongs to a subject that is already in the database.

Clearly, this simple face recognition system will be slow if the number of pixels m is large: the cost (both in time and memory) of each face comparison (1) is linear in m. If the database of known faces is very large, this cost becomes prohibitive.

Face Recognition in the Face Space

To speed up our system, we reduce the dimensionality of our data (face images) by working in the reduced "face space".

The idea here is to:

1. Find the best approximating subspace of dimension $p \leq m$ to our set of faces using Singular Value Decomposition

$$F = U\Sigma V^T.$$

   The reduced face space is spanned by $< u_1, u_2, \ldots, u_p >$.

2. Project all (known and unknown) faces to this face space. This way, each face $f \in \mathbb{R}^m$ is represented as a much smaller vector $x \in \mathbb{R}^p$.

3. Perform face recognition just as we did before, but working with the projections. That is, compare an unknown face x to each face x_i in the database using euclidean distance:

$$\varepsilon_i = \|x - x_i\|_2 \qquad \text{for } i = 1, \ldots, n. \tag{2}$$

This greatly speeds up each comparison (2), since vectors x are much smaller than f. Even if the face database contained a huge number of faces, recognition can be performed extremely fast.

For more details on how to perform the dimensionality reduction, see my lecture notes `12-Eigenvalues and SVD.pdf` and the paper *Facial Recognition with Singular Value Decomposition* that is included in this project's folder.

Suggested Activities

You are asked to evaluate and compare the *accuracy* and the *efficiency* of the face recognition algorithm in the **image space versus the reduced space**.

- **Accuracy:** How well does the system perform? That is, what is the percentage of unknown images that the system is able to recognize correctly?
- **Efficiency:** How fast and memory-intensive is the algorithm?

More specifically, do the following activities.

1. Load all $n = 165$ images from the Yale database (in the project folder). Plot a few faces, to make sure they have been loaded correctly, as in Figure 1 of the paper "Facial Recognition With SVD".

2. Write a function `img2vec` that given an image of 243×320 pixels, converts it to a 1-dimensional array of size $m = 77760 = 243 \times 320$ of **real** values. Write a function `vec2img` that performs the reverse operation: converts an array to an image which can then be displayed.

   Convert all faces to vectors and store them as columns of an $m \times n$ matrix, which we will call the *dataset*.

   Compute the *mean face*

   $$\bar{f} = \frac{1}{n} \sum_{i=1}^{n} f_i$$

   and plot it, as in Figure 2 of the paper. Normalize the dataset by substracting the mean face to each face in the dataset.

3. Divide the dataset into two disjoint sets: a set of known faces (which we call the *database* F), and a set of unknown faces (which we call the *query set* Q). You can decide which faces go into each set.

   For example, since there are 11 faces per individual, you may decide to treat 6 of them as 'known' and 5 of them as 'unknown'. Explain how you have decided to divide your dataset.

4. **Let's perform face recognition in the original image space** (i.e. we don't use SVD just yet). Write a function `id = query(face, database)` that receives an unknown face, compares it to the database, and returns the identity of the closest match in the database (e.g. `id = "subject01"`). Test it with a several unknown faces. Check that some faces are recognized correctly, whereas others are not. Most notably, faces with extreme lighting conditions (such as `subjectXX.leftlight` or `subjectXX.rightlight`) are sometimes not recognized correctly.

5. Query every image in the query set, and check if it has been correctly identified or not. Report the accuracy of your algorithm as the percentage of query images that it is able to recognize. Of course, accuracy will depend on how many faces you put in the database vs the query set. Try dividing the dataset into F/Q in different ways, and see how this affects the accuracy.

6. **Let's now perform recognition in the reduced face space**. Compute the SVD of the matrix $F = U\Sigma V^T$ using the standard (python or Matlab) library function `svd`.

The face space is spanned by the first columns $u_1, u_2, \ldots, u_p$ of U. These form a basis of the face space, so they are called the "base faces". Plot the base faces, as in Figure 3 of the paper.

Remark: The base faces are eigenvectors, i.e. they are defined up to a normalizing constant. The SVD function normalizes them to norm 1. However, to visualize them as images we want their values to be in the range 0:255. Thus before plotting them, you will have to scale them in such a way that the maximum value in the array is 255. For instance:

```
# Plot first base face just for fun
u1 = np.copy(U[:,0])    # 1st column u_1 of U
# Scale values from [0:1] to [0:255]
u1 = u1*255/np.amax(u1)
img = vec2img(u1)
img.show()
```

7. How to the select the dimension p of the face space? You want to select a value of p that is not too large (to keep the dimension of the face space small) but it's also not too small (to avoid discarding too much information). Plot the singular values $s_1, s_2, \ldots$ as a function of p, and look at their decay. A good compromise is usually to select a value of p that corresponds to the "elbow" of this graph. Which p do you find?

8. Project both the database F and the query set Q into the reduced face space. Each face is thus represented by a much smaller vector $x \in \mathbb{R}^p$. Notice that the entries of x do not correspond to pixels anymore, so x cannot be displayed directly as images. After obtaining the projected database/query set, you can perform face recognition using the same function `query` as before.

   Query every image in the query set, and and report the new accuracy. Does accuracy improve or degrade when you work in the reduced space? How does accuracy depend on p?

   Compare the efficiency (time/memory cost) of face recognition in the full space versus the reduced space. In particular, compare the execution time per query.

   Write your conclusions: Does it make sense to use SVD for this problem? How does it affect accuracy and efficiency?

Extra Credit

1. Use your own code to compute the SVD. For instance, compute U, V, Σ from the eigenvalues/eigenvectors of the matrix $F^T F$. Compare your result with that of the standard (python or Matlab) library function *svd*.

The paper *Facial Recognition with Singular Value Decomposition* mentions several extensions/improvements to this simple face recognition system. Implement either/both of the improvements below.

2 In the database, each individual may have more than one face images with different angles, expressions, and so on. In this case, we can use the **average** of them for the identification.

3 A face f is classified as face f_i when the minimum ε_i is less than some predefined threshold ε_0. Otherwise the face f is classified as "unknown face", and optionally be used to initialize a new identity.