Face Recognition using SVD

Group members:
Ovadia Sutton
Haider Ali

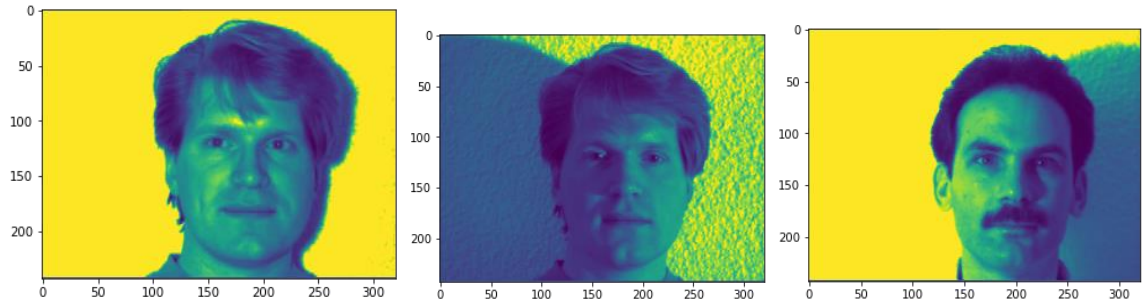
Introduction

In this project we use SVD to identify if a person's face is present in an image. If the model gives more than 95% accuracy if can be used as a security door lock, Facial attendance, etc. There are many more applications in the field of Neuroscience, Psychology, Security, and Computer Vision.

We explore the ability recognize faces based on their matrix representation. A black and white image can be characterized by a square (m by n) matrix representing the pixels and pixel gradation values ranging from 0-255. We can always flatten this matrix. Here we work with images 243 x 320 pixels that can be flattened to a vector of size 77760. We store a database of faces in these vectors. We then consider a new face and compare it to our faces. Can we find a way to identify the subject of the image based on its "closeness" to the images in our database. We go about this in two possible ways. First, we consider directly the difference (and its norm) of the image vector to each image vector in the database and assume it matches the identity of the one with the least difference. Secondly, we reduce our database to base faces using Singular Value Decomposition. We can then project our faces into this base face space and do a similar comparison of vectors to try to identify the subject. We compare the two methods accuracy and efficiency. (We note that we can be more elaborate by condensing images of the same subject, but reserve that for another time).

1. Loading Images

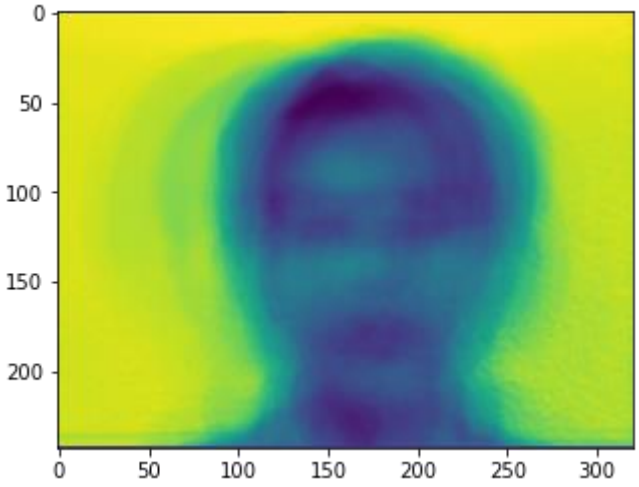
We download a folder of images from the Yale database. We loop through the folder and extract the image in our Python environment. We plot a few of these faces to ensure they loaded correctly. While the recommended library/function for viewing these images is PIL.Image and more accurate portrayal of the black and white image, here we choose Matplotlib.imshow to view our images. We do this because it does not require us to scale our vectors. Below are some plotted images.



2. Images to Vectors (and back) and Normalization

Since we will be working with the images in vector form and would like to view them, we write two functions: one that converts the image to our desired vector form and the other that reverses this process, mapping the vector back to a plottable image. The primary idea here is to convert the image into a NumPy array and reshape it from the 243 x 320 to 77760 x 1 and vice versa. We then use this function to store our images as vectors and then store all the vectors in a matrix called our dataset (of size 165 (number of images) x 77760 (pixels per image)). We then compute the mean face of the dataset which sums the 165 values of a specific pixel spot and then divides by 165 giving the average of that pixel value. We plot this mean face and show it below. We then normalize the dataset by subtracting off the mean face from each vector.

Face Recognition using SVD



3. Dividing the dataset

We divide our entire dataset in two steps. Firstly, we divide the data based on the number of subjects. That is each bucket will contain all (11) images of one specific subject, resulting in 15 buckets of 11 images each. We then, shuffle the images in each bucket so they are in no particular order (as opposed to the initial order that they were in which was consistently ordered by image type). This shuffling will cause the program to return different results every time based on how diversely images are shuffled. Then for each subject we choose 6 random images to be part of our database and 5 other images to be part of the query set that we will compare to the database and try to see if we can recognize the identity of the subject.

4. Base Face Recognition Method

We use our first approach to facial recognition. That is for a given image vector given to us, we compare it to each image vector in our database. We subtract one vector from the other and compute the norm. We consider the minimum norm returned across all vectors and match the identity of the image to the subject of the image with the minimum norm of difference. We also retain the true identity of the image's subject based on the bucket it was taken from.

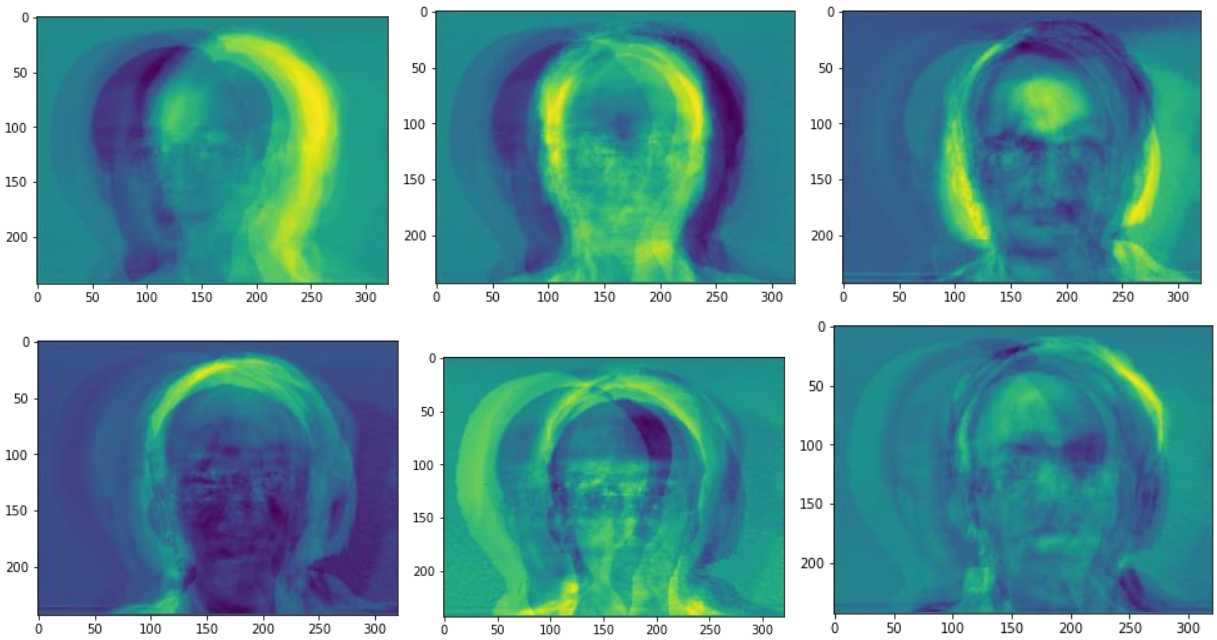
5. Querying all Images in query set

We do this with each vector and consider when we get the true identity and when we get a false identity. We count truths/total as our accuracy percentage. We return `Accuracy: 0.86. That is 65 out of 75 faces queried were matched correctly with their true identity. The code took 0.7823131084442139 seconds to run` We note that this will vary each time we run this code per the shuffle function mentioned above. We can change the way we divide our dataset. The more images in dataset the more likely a more accurate match will be the minimum.

6. Transforming our Datset Using Singular Value Decomposition.

We use the numpy SVD function to divide our database into USigmaVt. We do this specifically, on the 90 images that we have stored in our database (not all 165). We think that this is often the more practical way to perform SVD as we will not always know about the data we are going to query when designing it. (Although we do think the more faces used, the more helpful SVD will be and therefore maybe more accurate (especially since the actual query vectors are being included in the decomposition which is creating the base space. We consider the first p columns of U (or first p rows of U transposed). These will be the core vectors that we use as our base space. Below we display some base faces.

Face Recognition using SVD



7. **On Choosing P (Number of vectors of U that we use to construct a base space)**

In choosing p (how many of the first k/n vectors we will use to create our base space is difficult), At the very least, we think we need at least one vector per person (15) to maintain unique identities. At most, once the plotted images of each U vector go blank (vector values are close to 0 and we have surpassed the rank r of the matrix) the extra vectors do not add much information. We have distilled our range between the first 15-50 vectors that is 15 < p < r where r is about 50 in our case. We plot p against our σii entries which gives weights to each vector in U. As these weights decrease, their corresponding column becomes less significant, so we can consider setting p=k where σkk<error. We plot the values of σ against k and watch the decline. We observe the elbow at r about 50 the same value in which the images started looking blank.

8. **SVD Facial Recognition**

We project each vector in our database and query set onto the space spanned by the first p vectors of U. We now run the same query function comparing a projected query vector to the database of projected vectors and try to see if we can identify the subject in the image. For p = 20 we obtain that the accuracy is the exact same as the above, but the time run is faster at `0.06576848030090332 seconds to run`. This increase in runtime is due to the fact that our new vectors are of size p as opposed to 77760. Running this multiple times we have been getting the exact same or very similar accuracy unless we decrease p a lot. We can increase p for accuracy but will lose efficiency or decrease p for efficiency but lose accuracy. We can now play with this code to in three ways 1) change the way we divide our data 2) use SVD on database or entire set 3) alter values of p between 15 and the rank (lower p is faster but less accurate and higher p is slower but more accurate). We compared the SVD and base model results for different P values.

P	Base Accuracy	SVD Accuracy	Base time	SVD time
10	0.866667	0.773333	0.706521	0.046470
20	0.813333	0.773333	0.633335	0.047130
30	0.840000	0.746667	0.615559	0.044129
50	0.853333	0.853333	0.602645	0.049954

Face Recognition using SVD

Conclusion

We have found a way to consider a vector representing an image against vectors of images that our in our database to attempt to identify the new image subject based on the identities of subjects in the database. We do this by just comparing the vectors and using the norm of the difference which provides a decent but definitely fallible accuracy. We try to the use singular value decomposition to create a base face space and projecting our vectors into this space and then trying the same method of identification. We obtain a faster runtime for the recognition without a big difference in accuracy. Accuracy increases at the expense of efficiency as we increase our subspace of U by first p columns.

A note on the inaccuracies. We attribute the mismatched faces to two causes. The images are taken from different angles and with different features (right light, left light, center light, glasses, no glasses, happy sad). 1) The images of the same subject may differ greatly based on the different features (right light may be very different from left light) and therefore the norm is high and it doesn't recognize the matching subject. 2) All subjects have 11 types of pictures with the same features (Subject 1 right light Subject 2 right light...) As such, the resemblance of one image to another may not be based on the subject but based on the shared feature (In fact the same way we ran subject recognition we can consider running a program to match feature recognition). Combining these two points, the program may find the minimum norm, that is the best match to a different subject with a shared feature over the same subject with a different feature especially if it is a unique and glaring feature that greatly changes the matrix of pixels. We hoped that SVD would mitigate this issue, but it doesn't seem to have changed much. We think that if we compose all images of an individual subject into a single "mean" vector for each individual, our accuracy will improve greatly as it neutralizes external factors (such as lighting...). This will also improve runtime as we do not need to compare to every image, but only one mean image per subject in the database. As well, if our database contains more images per subject the recognition will improve.