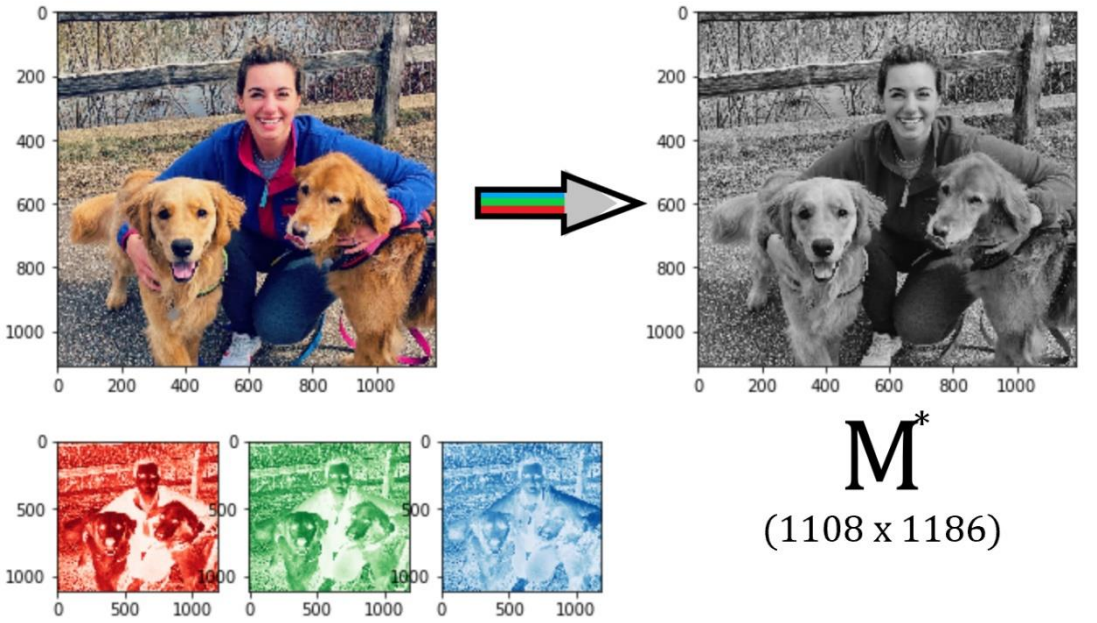
Introduction

Over the last decade, facial recognition technology has improved as the availability of big data and cloud computing has grown. Overall facial recognition has three steps, first isolate the faces in an image. Next identify the key features of the face and express them mathematically. Finally use a classifier model to label the mathematical representation of the face. These techniques are being applied in a growing number of fields in industry and academia. To facilitate this growth a host of open-source software libraries are available.

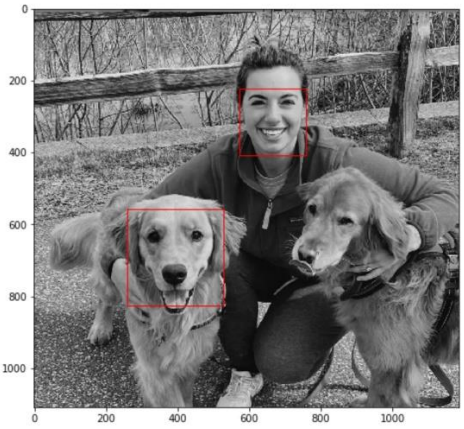
For this project we are applying Single Value Decomposition (SVD) to reduce the dimensionality of the image, before creating a mathematical representation of the face. Instead of implementing a full classifier model, the central question will evaluate the effect SVD has on the mathematical representation of the facial features. If SVD does not change the mathematical representation of the facial features, we can safely use SVD as an image compression technique. However, if there are changes, we will need to consider the effects image compression can have on our classifier model.

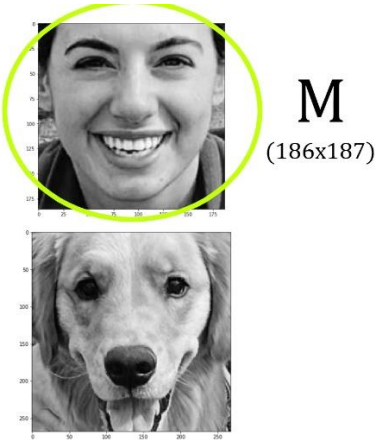
Data Processing

The original image is 3 arrays of 8-bit pixels with each array representing a color (Red, Green, Blue). In order to Use SVD and Principle Component Analysis (PCA), the 24-bit color image must be converted to an 8-bit grayscale image. Each pixel is now an integer from 0 to 255. This grayscale representation of intensity is the result of our first transformation.



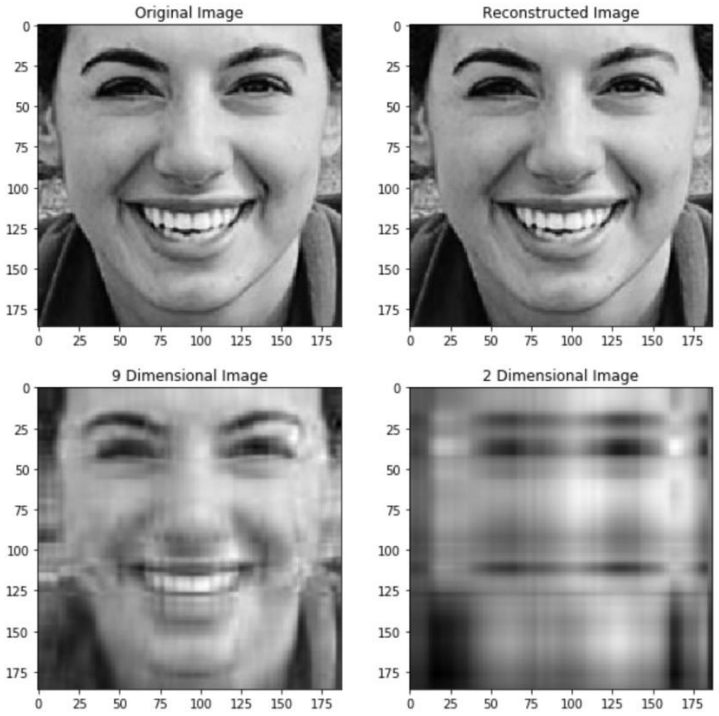
The M^* matrix is not the conjugate of matrix M but the base image we will use to detect faces from. Dlib-ml library has a frontal face detector we use to identify all the faces in the photo. We will then crop the image around the faces to create our M matrix. For the photo selected, two faces were identified. Although the dog face is adorable, this project is sticking to human faces. Our M matrix is now an 8 bit 186 x 187 image. We can normalize the data by dividing each element by 255. We are now ready to perform our SVD analysis.



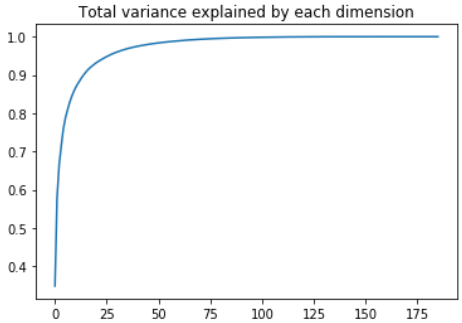


SVD and PCA analysis

Python has many numerical packages for matrix operations. This analysis uses the numpy svd function and the sklearn PCA function. Both are opensource and can be implemented with the attached jupyter notebook. Applying SVD we find U, s, and V. U is a 186 x 186 matrix and V is a 187 x 187 matrix. U and V are rotation and reflection matrices. s is a list of eigenvalues that form a diagonal scaling matrix.



The original data can be returned by multiplying U, S, and Vt, where Vt is the transpose of the V matrix and S is the diagonalized matrix of eigenvalues s. By reducing the number of non-zero eigenvalues in s we can eliminate dimensions of variance. This is best illustrated by the bottom two images to the left. The lower left figure is the representation of the image with 9 dimensions. According to PCA 9 dimensions are required to explain 85% of the data (see weaknesses section). The lower right image is the projection of data into 2 dimensions. The results of PCA analysis are plotted as variance explained vs dimensions below.

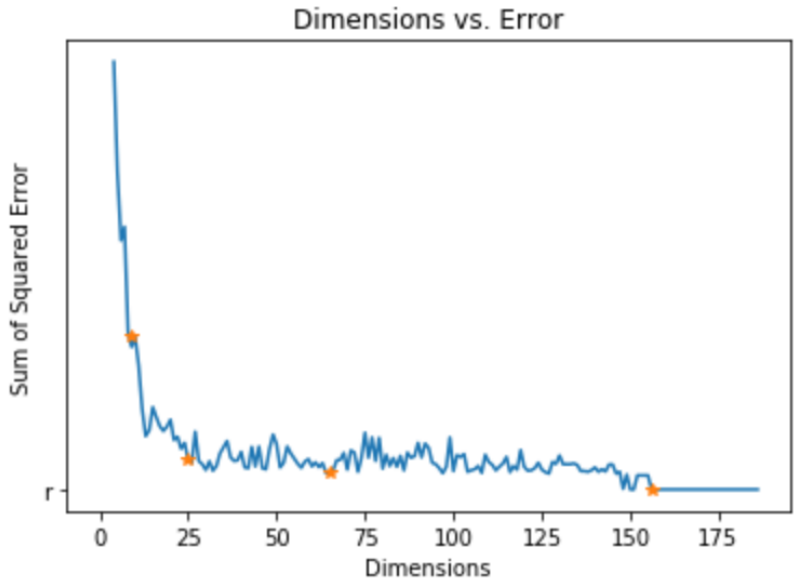


Weaknesses

Although PCA analysis tells us 9 dimensions are required to capture 85% of the image, it does not tell us which 9 eigenvalues to keep in our SVD. This is evident in the graph on the next page where the 9th dimension has a much larger SSE than 25 dimensions.

SVD and mathematical representation of faces

dlib-ml library has trained a face feature model that identifies the location of 68 key facial features. These 68 feature locations can be plotted on the image. To evaluate if the SVD image compression was acceptable, the location of each feature on the compressed image is compared to the location of each feature on the uncompressed (original) image. The sum of squared error is plotted against the number of dimensions in the image. This graph shows how increasing the first few dimensions increases the accuracy of the facial feature finder until the first 25 eigenvalues are used. After this point the error stays constant around 100 until dropping to zero just past 160 dimensions. The red points are plotted as images in the appendix.



Conclusion

SVD is a valid method for image compression when performing facial recognition. Given the results of this project it is safe to say that only 25 dimensions are required to transform this image into a mathematical representation of 68 facial features. This reduces data sizes and ultimately the amount of resources facial recognition technologies consume.

References:

Davis E. King. Dlib-ml: A Machine Learning Toolkit. Journal of Machine Learning Research 10, pp. 1755-1758, 2009

Rosebrock, Adrian. “Facial Landmarks with Dlib, OpenCV, and Python.” Pyimagesearch, 3 Apr. 2017, www.pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/.

Bradski, G. “The OpenCV Library.” Dr. Dobb's Journal of Software Tools, 15 Jan. 2008.

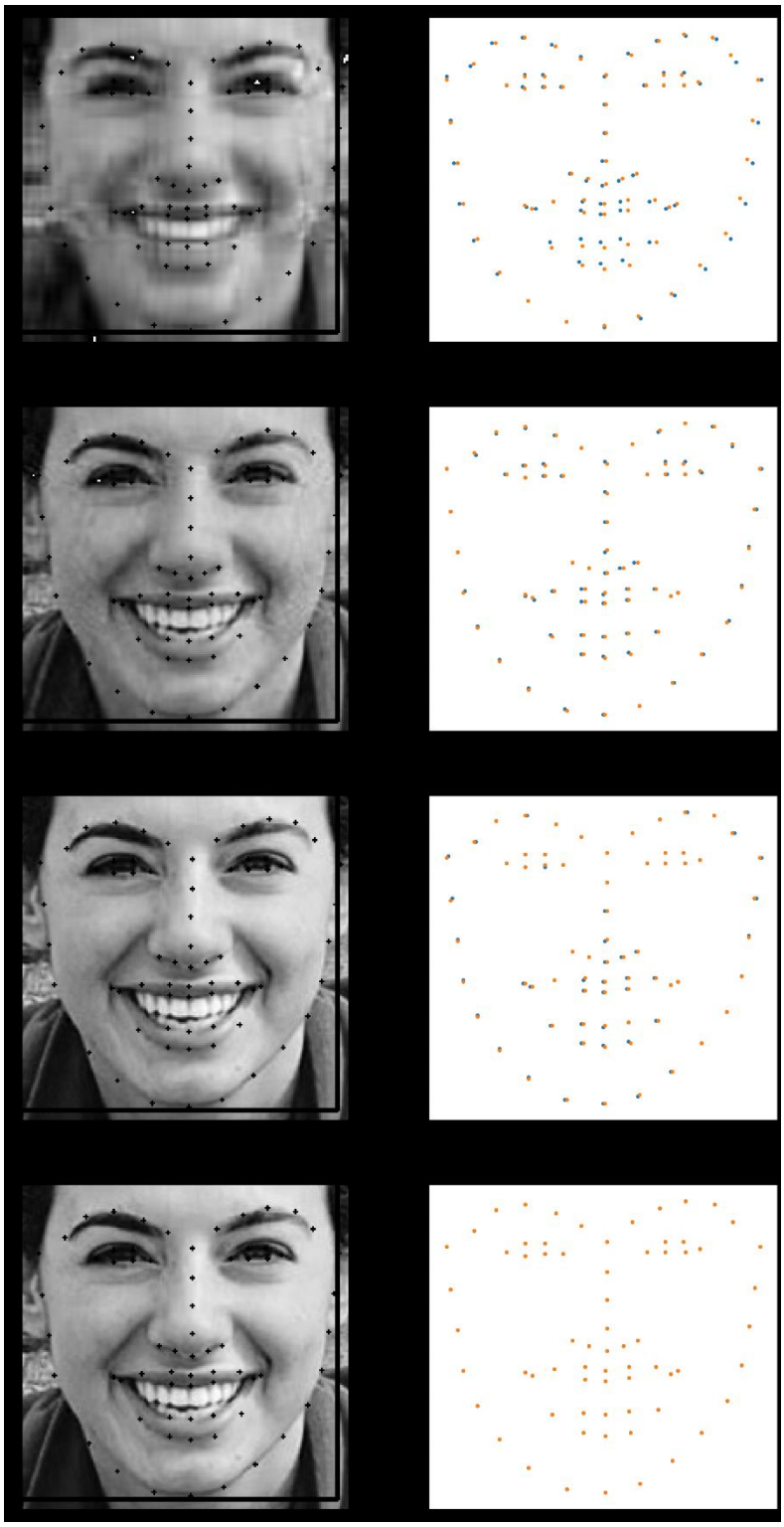
John D. Hunter. Matplotlib: A 2D Graphics Environment, Computing in Science & Engineering, 9, 90-95 (2007)

B. Amos, B. Ludwiczuk, M. Satyanarayanan,

"Openface: A general-purpose face recognition library with mobile applications,"

CMU-CS-16-118, CMU School of Computer Science, Tech. Rep., 2016.

Appendix:



Dimensions: 9

Sum of Squared Error: 395

Dimensions: 25

Sum of Squared Error: 82

Dimensions: 65

Sum of Squared Error: 48

Dimensions: 156

Sum of Squared Error: 0

Github Repo: https://github.com/bstark22/SVD_FM5002