New York University

Mathematical Techniques for Computer Science Applications

CSCI-GA.1180-001

Optimal Facial Image Compression by SVD

Dewei Chen

December 16, 2015

Abstract

Images are prevalent in the Internet and throughout the digital space today. While they are an excellent way to communicate information (an image is worth a thousand words), it is also an expensive form of data to be transported across the network. To solve this problem, compression software is often used to encode an image to a smaller physical size so it can be transported more efficiently while minimizing the perceptible quality change from its original copy. In light of this, a question that comes up is: how much could an image be compressed before it is no longer valuable in providing information to the user? In this paper we will discuss how to optimize an image compression algorithm that is based on finding the singular value decomposition of a matrix. Due to the subjective nature of image quality examination, we also introduce an image structural similarity index to the algorithm so it could adjust the compression ratio automatically.

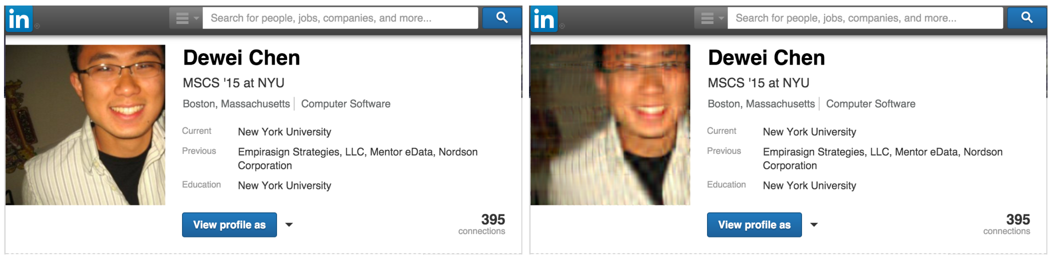
Keywords: Singular Value Decomposition, Image Compression

1 Introduction

Social network websites such as Facebook, Twitter and LinkedIn all provide the user with a "headshot" or an image that describes who they are on their profile page. Users have the option to upload a picture of any size from their computer and the website backend will handle resizing the image such that it fits in a designated window. Since these websites are often visited by millions of users at any given moment, it is important that profile images are loaded quickly. To do this, they must compress the image so that it could be stored with minimal space and efficiently loaded.

While it is important to minimize the physical size of a compressed image, validity of the image after compression must also be considered. For example, the LinkedIn website provides a professional service where a user's profile page may be viewed by many future employers. Recruiters passing by a potential candidate's LinkedIn profile would most likely skip to the next page if they saw a blurry and unrecognizable (but compressed) image of a candidate (see fig.1)! Therefore, it is practical for the compression software to find an optimal balance between image quality and compression ratio.

How much should we compress a facial image such that we could still identify the individual while image compression ratio is minimized? Common



Dewei Chen
MSCS '15 at NYU
Boston, Massachusetts | Computer Software
Current New York University
Previous Empirasign Strategies, LLC, Mentor eData, Nordson Corporation
Education New York University
395 connections

Dewei Chen
MSCS '15 at NYU
Boston, Massachusetts | Computer Software
Current New York University
Previous Empirasign Strategies, LLC, Mentor eData, Nordson Corporation
Education New York University
395 connections

Figure 1: A good versus blurry LinkedIn profile page

image formats such as JPEG have a compression ratio option that must be manually specified by a user when generating the image. A user must make subjective judgment as to whether the encoded image is "acceptable" or not. This paper finds an algorithm that can automatically adjust the compression ratio of a facial image by using the structural similarity index [2] as a determinant of image quality and automatically fine tunes rank of the image matrix to obtain an optimal compression of the image.

2 Background

While there are many state of the art image compression/encoding techniques used in practice today, this paper uses the singular value decomposition (SVD) technique that will be described below. The SVD is used here because it is easier to examine the changes in an explicit image representation (a matrix) as opposed to other more complicated transformation or encoding functions. However, the technique used could be applied with other image compression techniques.

2.1 Singular Value Decomposition

An image is represented as a $m \times n$ matrix where m and n represent height and width of the image in number of pixels, respectively. For a grayscale image, each matrix element is an 8-bit number ranging from 0-255 indicating the level of brightness at that pixel. For a colored image (or RGB image) each matrix element contains an array of 3 elements, each 8-bits with value representing the intensity of 3 color channels: red, green and blue.

Using this matrix representation of an image, we could perform various operations on the image as we would on a regular matrix. The Singular Value Decomposition is a way of factorizing a matrix A so that it is decomposed into U, S and V^T matrices with both U and V being orthogonal matrices and S is a diagonal matrix composed of singular values of A in decreasing order:

$$A = \begin{bmatrix} u_{11} & \cdots & u_{1m} \\ u_{21} & \cdots & \cdots \\ \cdots & \cdots & \cdots \\ u_{m1} & \cdots & u_{mm} \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 & 0 & \cdots \\ 0 & \sigma_2 & 0 & \cdots \\ 0 & 0 & \sigma_3 & \cdots \\ \cdots & \cdots & \cdots & \cdots \end{bmatrix} \begin{bmatrix} v_{11} & \cdots & v_{1n} \\ v_{21} & \cdots & \cdots \\ \cdots & \cdots & \cdots \\ v_{n1} & \cdots & v_{nn} \end{bmatrix} \tag{1}$$

The non-zero singular values σ_i stored in S essentially captures the important information contained by the original matrix A. Because of this, by finding the singular values of an image expressed in matrix form, we are capturing the essential information contained in the image.

Image compression is achieved by removing some of the smaller (i.e. less crucial information of the image) singular values from S. Since S contains the singular values sorted in decreasing order, we simply take the largest k elements along the diagonal and set others to 0. In a similar fashion, we remove (set to 0) columns in U and rows in V^T with index larger than k. Our new compressed matrix A' calculated by re-multiplying the new U, S and V^T. For some given k, we have the following result for the compressed A':

$$A' = \begin{bmatrix} u_{11} & \cdots & u_{1k} & 0 & 0 \\ u_{21} & \cdots & u_{2k} & 0 & 0 \\ \cdots & \cdots & \cdots & 0 & 0 \\ u_{m1} & \cdots & u_{mk} & 0 & 0 \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 & 0 & \cdots \\ 0 & \sigma_2 & 0 & \cdots \\ 0 & 0 & \sigma_k & \cdots \\ \cdots & \cdots & \cdots & 0 \end{bmatrix} \begin{bmatrix} v_{11} & \cdots & v_{1n} \\ \cdots & \cdots & \cdots \\ v_{k1} & \cdots & v_{kn} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{2}$$

2.2 Compression ratio

As a result of reducing the rank of an image, less bytes are needed to store image information because we set them to 0. Given a $m \times n$ image, the number of bytes we need to store in the compressed form is: $m \times k$ bytes for U, $k \times k$ bytes for S and $k \times m$ bytes for V. The compression ratio from various values of k is plotted in fig.2 for image sizes used in this project.

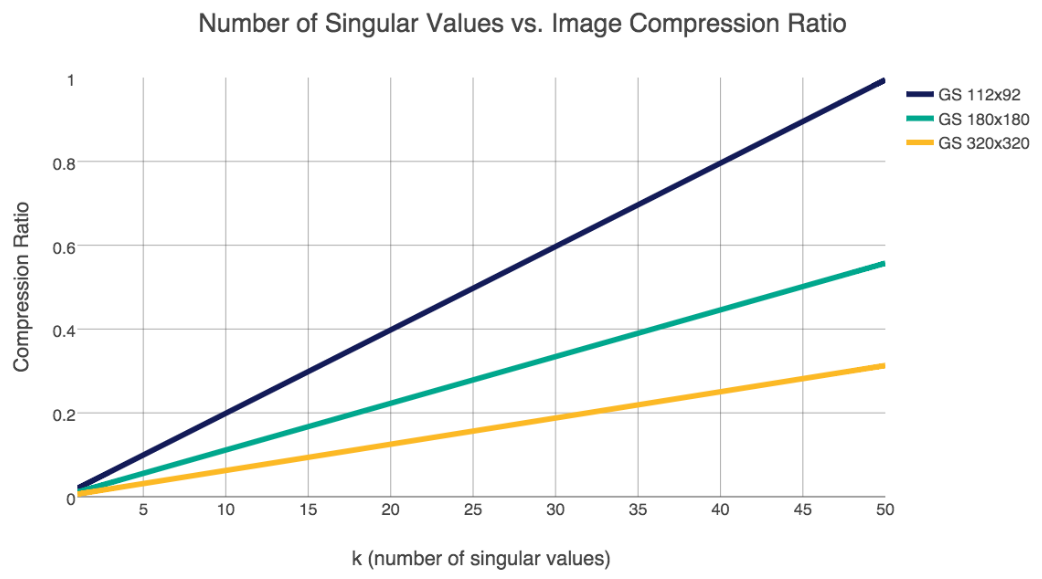


Figure 2: Compression ratio achieved at different k for various gray scale image sizes

$$CR = \frac{k(m+n+1)}{m*n} \tag{3}$$

2.3 Lossy vs. Lossless Compression

There are primarily 2 ways of compressing an image. Lossless compression often has a function that decompresses an image in addition to compressing it. It allows for a compressed imaged to be completely restored to its original state without loss of information. An example of this is the PNG or JPEG image format commonly used today on the web.

Lossy compression, on the other hand, compresses an image but some information is lost. The singular value decomposition is a lossy compression technique because there is no way to recover the zeroed singular values in S or the columns/rows of U or V once they are removed. Because of this, it is more critical that the chosen singular values we remove will result in an image that does not deviate too much from the original.

2.4 Structural Similarity Index

For applications in which images are ultimately viewed by humans, the only "correct" method of quantifying image quality is through subjective evaluation. Since the aim of this project is to achieve an automatic compression algorithm, some standard of evaluating image quality is needed. While the conventional method in image quality estimation has been calculating mean squared error (MSE) between 2 images pixel by pixel, it has been shown that very different images could still have similar MSE values. The structural similarity (SSIM) index [2] tries to counter this problem by taking advantage of the human visual system characteristics. Realizing that we are more adapted for extracting structural information from an image, the SSIM gives a quality score between 0 and 1 by comparing local patterns of pixel intensities normalized for luminance and contrast [2]. The SSIM is used as a standard in this algorithm to decide whether a compressed image is "good" relative to the original.

3 Programming and Design

The following sequence of development steps were taken to arrive at the final algorithm. Each step is built upon methods created in the prior step:

1. Create method to compress gray scale image using the SVD.
2. Create method to compress RGB image using the SVD (reuse method in step 1 for each color channel).
3. Use SSIM to compare quality of compressed image to the original.
4. Incorporate SSIM into the compress algorithm so to help select the optimal compression.

Programming for this project was done using Python due to the available libraries related to linear algebra and image analysis. The singular value decomposition of a matrix is computed using the numpy linear algebra library function **linalg.svd**. The structural similarity index is calculated using the **ssim** function provided by the Python imaging library scikit-image [3].

Fig.2 is a flowchart showing steps that the algorithm goes through. It takes 2 inputs: the image matrix and a target SSIM value. We will show

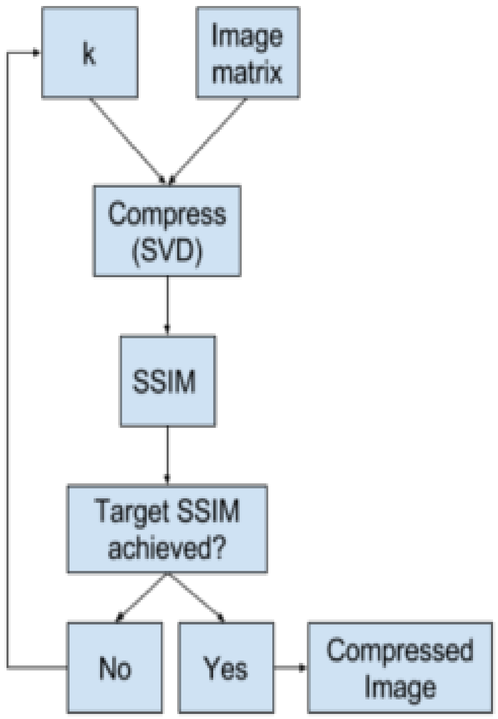


Figure 3: Rank optimization algorithm flowchart

how the target (i.e. optimal) SSIM value is determined in the results section. Since the value of optimum k can be anywhere between 1 and rank of the image matrix, the algorithm repeatedly picks a new k until the result image is as close to the target SSIM as possible. Because the relationship between k and the image quality estimated with SSIM is approximately linear, binary search is used to optimize finding the best k between 1 and rank of the matrix. This is followed by a linear search along the direction that can further minimize the image SSIM (if the new SSIM is closer to target than the last SSIM).

4 Results

Gray scale images of 112x92 (.pgm format) pixels were randomly selected from 10 subjects in the database of faces [4]. For RGB images, 10 subjects were collected from random Facebook profile headshots. All RGB images retrieved are 320x320 in JPG format. They were picked such each one is uniquely different than the others in color and facial structure.

4.1 Optimal SSIM

The structured similarity index is used to estimate quality change when the image is compressed by our algorithm. To test that the ssim is able to distinguish between 2 different facial images, the ssim value between different images as well as with the same image was recorded. Average SSIM between 2 different gray scale images is 0.339 and between 2 of the same image is 1.0 (as expected).

Indeed, judging the image quality is a very subjective task. Although the SSIM provides a consistent evaluation of image quality, what is the "optimal" SSIM would be different under every person's eyes. We know that the lower bound of this optimal value must be greater than the average SSIM found between different images. After some subjective experiments, the lowest SSIM an image can have before it turns blurry is in the range between 0.85 to 0.9 (for a 112x92 gray scale image). To be more conservative, we chose 0.9 to be the minimum cutoff for an "acceptable" image quality.

4.2 Gray scale images

Shown in fig.4 are the results (shown 5 out of 10 images) of compression at k with values 1, 3, 5, 10, 15 and 20 (going from left to right), respectively. It can be seen that there is greater increase in image quality between each added singular value in the beginning than towards the end. This makes intuitive sense because the singular values in the beginning are larger in magnitude than ones in the end. Therefore there is not as much of an impact in image quality by removing the smaller singular values. This effect is shown by the "logarithmic" curve in fig.5.

4.3 RGB images

The RGB compression algorithm feeds the 2D matrix for each color channel into the gray scale compression algorithm discussed previously (fig.6). Interestingly each color channel of the same image had a slightly different SSIM value but deviates by no more than 0.1 points from each other. This

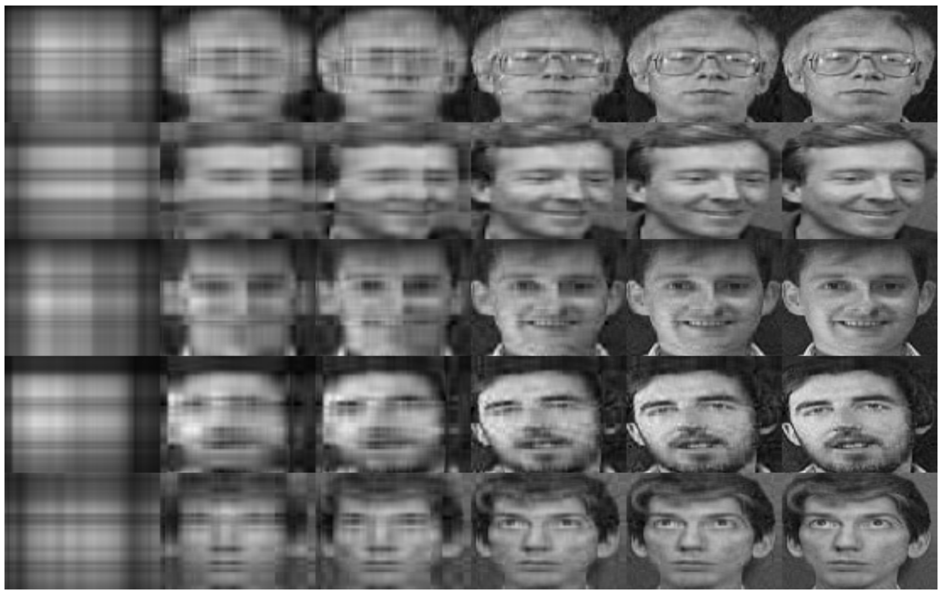


Figure 4: Gray scale image (112x92) compression result for k = 1,3,5,10,15,20 going from left to right

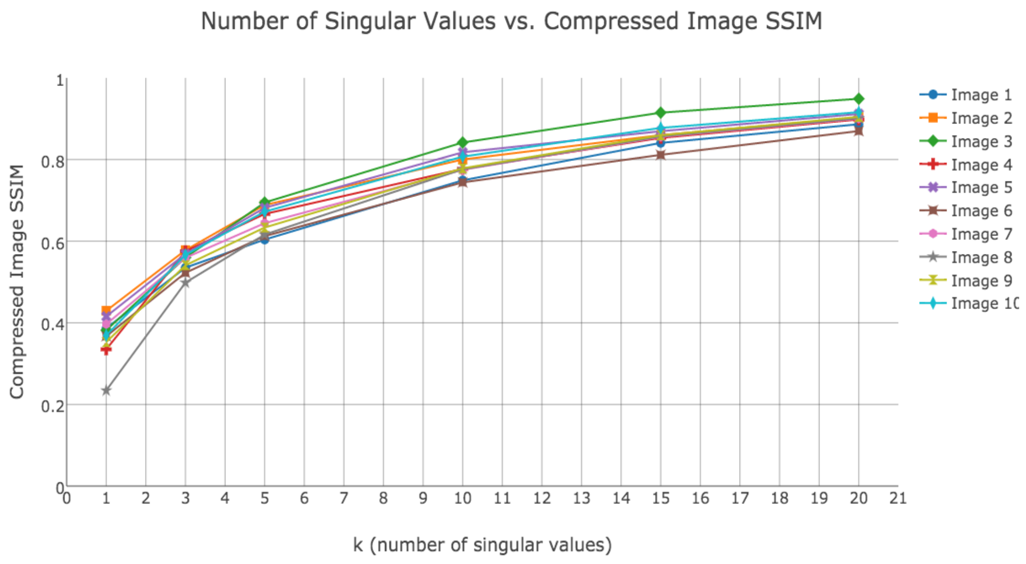


Figure 5: Relationship between rank of compressed grayscale (112x92) images and measured quality.

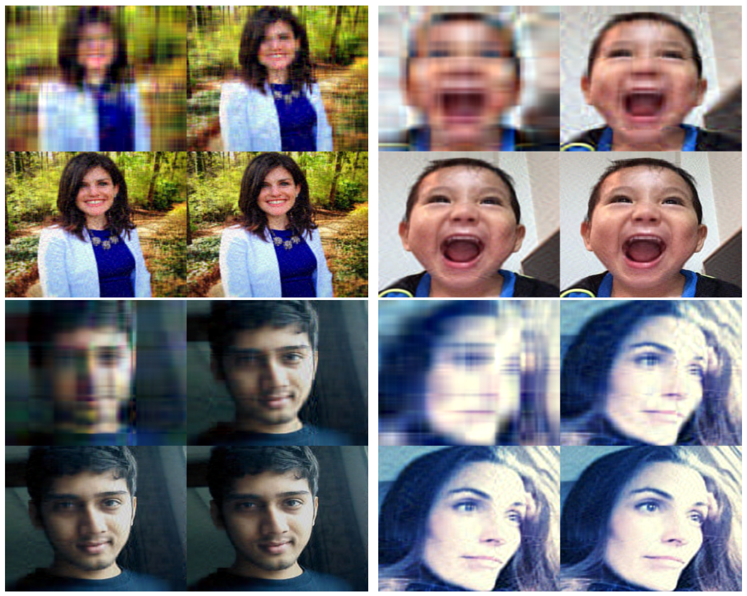


Figure 6: RGB image (320x320) compression result for k = 5,15,25,35 going from left to right, top to bottom

observation makes sense because the matrix we get for each channel is different depending on the image coloration and so would get a slightly different SSIM. Currently the compression algorithm uses the same k value for all 3 color channels. A possible optimization in the future would be to optimize a different k for each color to achieve a better overall image compression quality.

4.4 Auto Rank Optimization

To test performance of the auto rank optimization algorithm, it was ran with various target SSIM inputs while the compressed image rank was recorded. The expected behavior is that given a larger target SSIM (better quality) it should return a compressed image with bigger k because larger number of singular values should have a better image. The data for this test is shown in fig.7. For several images, the algorithm did not return a strictly increasing rank of the compressed image. This could be attributed to the fact that while certain compressed uses fewer singular values, it ”appears” to be more similar to the original image when evaluated by the **ssim** function in the

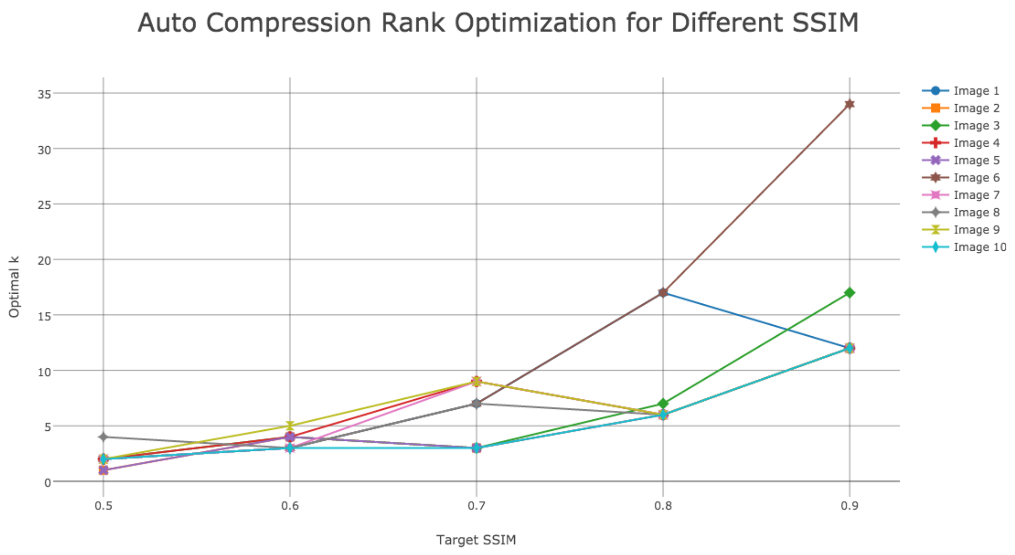


Figure 7: The optimal k discovered by algorithm for some target SSIM

scikit-image library. However, most of the images compressed followed the trend.

5 Discussion

The auto rank optimization algorithm was able to successfully compress the image to some designated target SSIM value provided. As discussed in 4.1, we used a target SSIM of 0.9. While it is debatable on what is the "best looking" SSIM on the images, that is not the purpose of this project. However, the algorithm is able to successfully find the rank that result in target and return a compressed image of that rank.

Note that the compression ratio discussed has been mainly theoretical. It is directly a function of k and original image dimensions m and n. The actual output image is not compressed because we are still saving the full $m \times n$ image matrix as .PNG format. However, the effect of compression can be readily seen by the loss of information. A future work could be to actually implement a file type such that the smaller amount of data is saved as result of removing the zeroed rows/columns from the U, S and V matrices.

Compression performance of the algorithm on 112x92 gray scale images has been on the order of less than a second. This was improved further by doing a binary search on the k space. For 320x320 RGB images, the runtime is about 2 seconds because each color channel was compressed synchronously. Speed of the algorithm could potentially be improved if the compression is done asynchronously across the 3 channels.

6 Conclusion

In this paper we have demonstrated that it is possible to have a compression algorithm that could find an optimal balance between image quality and compression ratio. This is achieved by fine tuning the number of singular values from SVD based on the structural similarity index of the compressed image. With this algorithm it is possible to compress images to a designated image quality that also minimizes the theoretical number of bytes needed to store the image. Since we are computing the SVD, SSIM and multiplying the U, S and V matrices together on each round of the algorithm, its performance is worst than the generally accepted compression techniques today. However, it is a proof-of-concept that it is indeed possible to have a self adjusting algorithm to find an optimal compression. This could possibly be adapted on other higher performance compression techniques in the future.

References

[1] Davis, Ernest, Linear Algebra and Probability for Computer Science Applications, May 2, 2012 by A K Peters/CRC Press

[2] Wang, Z., Bovik, A. C., Sheikh, H. R., Simoncelli, E. P. (2004). Image quality assessment: From error visibility to structural similarity. IEEE Transactions on Image Processing, 13, 600-612.

[3] Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu and the scikit-image contributors. scikit-image: Image processing in Python. PeerJ 2:e453 (2014) http://dx.doi.org/10.7717/peerj.453

[4] The Database of Faces, Cambridge University Computer Laboratory, http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html