Image Compression Using Singular Value Decomposition and Principal Component Analysis

Zongyuan Yuan, Yirui Zhu

Department of Mathematical Sciences, Carnegie Mellon University

Spring 2019

Abstract

The **aim** of this paper is to investigate image compression techniques and more specifically, we focus on exploring Singular Value Decomposition (SVD). In the first half of the paper, we will explain how SVD works and how it can be applied to image compression; In the second half of the paper, we will explore how Principal Component Analysis (PCA) can be used to determine the parameter used in SVD. We find that using SVD and PCA together could compress an image effectively.

1 Introduction

In recent years, rapid growth in social media has produced an increasing amount of data (in the form of file, digital photos or others). Consequently, it poses great challenges to storing these data since storage space is limited. Therefore, compression techniques are developed to save valuable computing resources such as bandwidth and memory. Singular Value Decomposition is one of the most commonly used image compression method. By integrating Singular Value Decomposition and Principal Component Analysis, one could reduce the image size to a maximum scale and preserve the image quality at the same time.

1.1 Literature Review

Thus far there are various approaches to grayscale and color image compression. One is done via the discrete cosine transform, which is used by JPEG. It is currently used for compressing most images available on the Internet. Another technique that could be used to efficiently perform both lossy and lossless image is Haar wavelet compression. QR factorization and wavelet transformation algorithms could also be applied to get a reduced matrix such that the

corresponding image requires much less storage space than the original image. Beside these techniques, Singular Value Decomposition is the most common method used for image compression. Therefore, we will spend the rest of the paper exploring its definition and application.

1.2 Singular Value Decomposition (SVD)

Formal definition Let $A \in C^{m\times n}$ where m and n arbitrary and A not necessarily full rank, a *singular value decomposition* (SVD) of A is a factorization

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

where $U \in C^{m\times m}$, $V \in C^{n\times n}$ are unitary, $\Sigma \in C^{m\times n}$ is diagonal.
Besides, it is assumed that the diagonal entries of σ_j of Σ are non-negative and in non-increasing order, i.e. $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_p \geq 0$ where $p = \min(m, n)$.

1.3 Computation of SVD

The purpose of SVD is to factorize matrix A into the form of $U\Sigma V^T$. The general form is shown as below:

$$A = [u_1 \quad ... \quad u_r \quad ... \quad u_m] \begin{bmatrix} \sigma_1 & & & & \\ & \ddots & & & \\ & & \sigma_r & & \\ & & & \ddots & \\ & & & & 0 \end{bmatrix} \begin{bmatrix} v_1^T \\ \vdots \\ v_r^T \\ \vdots \\ v_n^T \end{bmatrix}$$

Here matrix U contains the left singular vectors, matrix V contains the right singular vectors and the diagonal matrix Σ contains the singular values. Notice that r is the rank of matrix A.

To perform SVD, one should find V, Σ and U in a subsequent order. Here is a detailed explanation of how to compute the matrices:

Step 1. Find V
Multiply both sides of the equation $A = U\Sigma V^T$ by A^T we get

$$A^T A = (U\Sigma V^T)^T (U\Sigma V^T) = V\Sigma^T U^T U\Sigma V^T$$

Since $U^T U = I$, we could simplify the right-hand side and get

$$A^T A = V\Sigma^2 V^T$$

This is equivalent to the diagonalization of $A^T A$. Thus, in order to find V, we find the eigenvectors of $A^T A$ which are the columns of V.

Step 2. Find Σ
From Step 1 we know that Σ^2 is the diagonal matrix constructed from the corresponding eigenvalues of A^TA. So we find the square root of the eigenvalues of A^TA and place them in a decreasing order along the diagonal of Σ.

Step 3. Find U
Multiply both sides of the equation $A = U\Sigma V^T$ by A^T we get

$$AA^T = (U\Sigma V^T)(U\Sigma V^T)^T = U\Sigma V^T V\Sigma^T U^T$$

Since $V^TV = I$, we could simplify the right-hand side and get

$$AA^T = U\Sigma^2 U^T$$

Similar to Step 1, we find the eigenvectors of AA^T which are the columns of U.

Example

Let $A = \begin{bmatrix} 2 & 1 \\ -2 & 1 \end{bmatrix}$

To find V, we need to compute the eigenvectors of A^TA where

$$\begin{aligned} A^TA &= \begin{bmatrix} 2 & -2 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 \\ -2 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 8 & 0 \\ 0 & 2 \end{bmatrix} \end{aligned}$$

To find the eigenvalues, we set

$$\begin{bmatrix} 8-\lambda & 0 \\ 0 & 2-\lambda \end{bmatrix} = 0$$

$$(8-\lambda)(2-\lambda) = 0$$

Therefore, the eigenvalues are 8 and 2.
When $\lambda = 8$,

$$\left[\begin{pmatrix} 8 & 0 \\ 0 & 2 \end{pmatrix} - \begin{pmatrix} 8 & 0 \\ 0 & 8 \end{pmatrix} \right] v_1 = 0$$

$$\begin{bmatrix} 0 & 0 \\ 0 & -6 \end{bmatrix} v_1 = 0$$

$$v_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Similarly (when $\lambda = 2$),

$$v_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Since $V = (v_1 \quad v_2)$, we get

$$V = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

We construct the matrix Σ^2 by placing the eigenvalues along the main diagonal in decreasing order.

$$\Sigma^2 = \begin{bmatrix} 8 & 0 \\ 0 & 2 \end{bmatrix}$$

Therefore,

$$\Sigma = \begin{bmatrix} 2\sqrt{2} & 0 \\ 0 & \sqrt{2} \end{bmatrix}$$

In the same manner as finding V, we find the eigenvector of AA^T to get U.

$$U = (u_1 \quad u_2) = \begin{bmatrix} \frac{-\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix}$$

Note U has an orthonormal basis, each u_i has a length of one.
Therefore,

$$A = \begin{bmatrix} \frac{-\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix} \begin{bmatrix} 2\sqrt{2} & 0 \\ 0 & \sqrt{2} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

1.4 Approximation of a matrix using SVD

Given $A = U\Sigma V^T$, we can write A as a linear combination of its singular values and vectors u_i, v_i^T

$$A = u_1\sigma_1 v_1^T + u_2\sigma_2 v_2^T + ... + u_i\sigma_i v_i^T + ... + u_n\sigma_n v_n^T$$

An approximation of matrix A is achieved by eliminating a number of terms in the linear combination.

2 Application of SVD: Image Compression

We could compress an image by making an approximation of its corresponding matrix as introduced in section 1.3. Since the terms are placed in order of dominance, i.e. $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_n$, we remove the last few terms in the linear combination which are less significant than the first few terms in order to avoid reducing the quality of the image heavily. Let k be the number of terms that we want to keep in the linear combination, matrix A is approximated as follows:

$$A \approx \sum_{i=1}^{k} u_i\sigma_i v_i^T$$

.

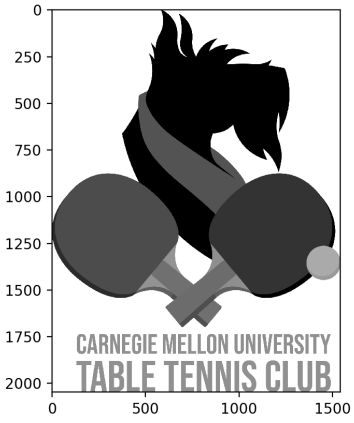
2.1 Implementation

We implemented the SVD in Python using numpy. The code snippet is as follows:

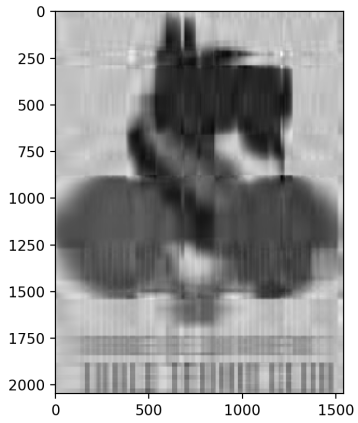
```
import numpy as np
import matplotlib.pyplot as plt

# a black and white image is a matrix
# where each entry represents
# the intensity of the corresponding pixel
def compress(imagematrix, k):
    U, sigma, V = np.linalg.svd(imagematrix)
    image_compressed = np.matrix(U[:, :k]) * np.diag(
    sigma[:k]) * np.matrix(V[:k, :])
    plt.imshow(image_compressed, cmap="gray")
    plt.show()
    return image_compressed
```

2.2 Example

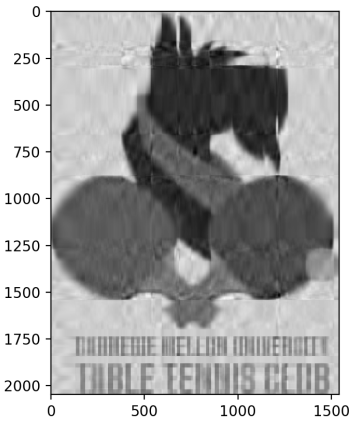
Here we apply SVD on an image to show that the image quality varies according to the number of terms preserved. The more terms we preserve, the closer the image quality is to the original image.



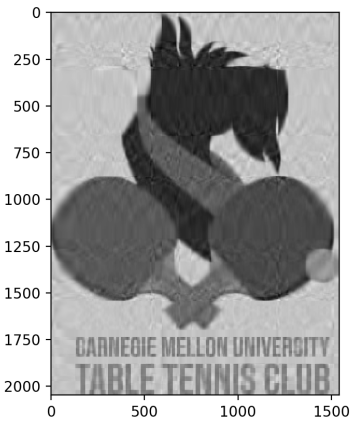
(a) original image.



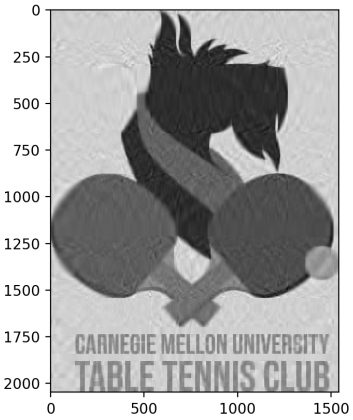
(b) n = 10.



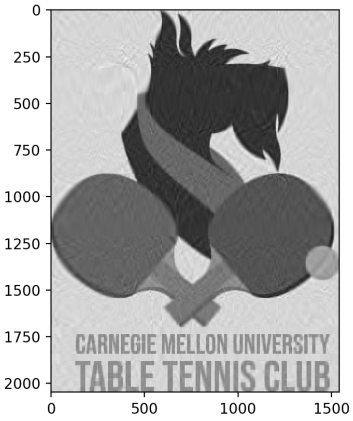
(a) n = 20.



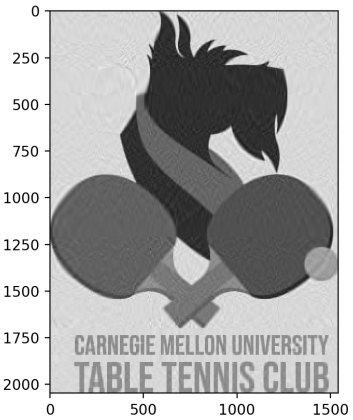
(b) n = 30.



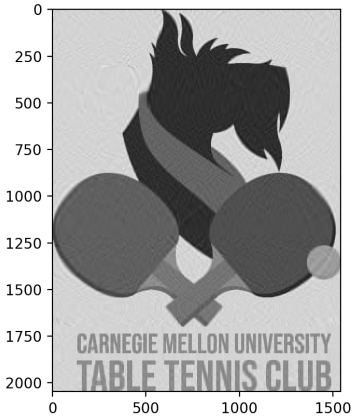
(a) n = 40.



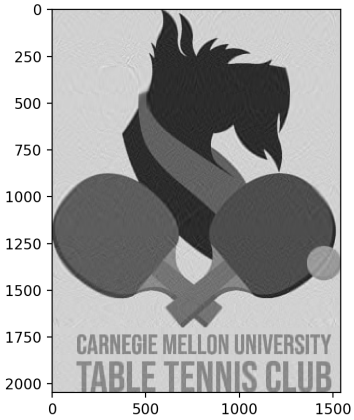
(b) n = 50.



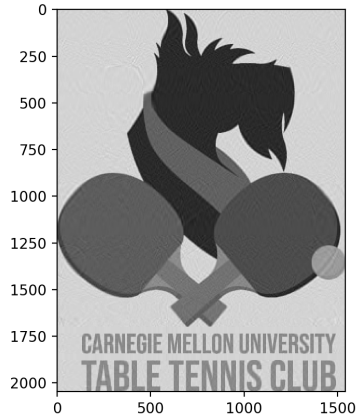
(a) n = 60.



(b) n = 70.



(a) n = 80.



(b) n = 90.

2.3 Determine the range of k

Given $A = U\Sigma V^T$ where $A \in C^{m\times n}$, assume we want to preserve the first k terms in the linear combination. Then we keep the first k rows of matrix U, Σ and V^T. So we have,

$$dim(U) = m \times k, dim(\Sigma) = k \times k, dim(V^T) = k \times n$$

Note that though $dim(\Sigma) = k \times k$, there are only k elements in Σ which lie on the diagonal since the rest entries are zero.

There are mn elements in A and we want to find a k such that the total number of elements in $U\Sigma V^T$ is less than mn. So we have the following inequality:

$$m \times k + k + k \times n < m \times n$$
$$(m + 1 + n)k < m \times n$$
$$k < \frac{mn}{m + n + 1}$$

By solving the inequality, we get the upper bound of k. Therefore, when $k \in (0, \frac{mn}{m+n+1})$, we get a compressed image.

2.4 Determine k with a fixed compression ratio

We could also find out the minimum number of terms to preserve if we already know to what extent we want the image to be compressed. Define the metric as compression ratio and let it be α, where $\alpha = \frac{m\times k+k+k\times n}{m\times n}$.
With a given α, we could figure out how many terms we need to keep.

$$\frac{mk + k + nk}{mn} = \alpha$$
$$k = \alpha\frac{mn}{m + n + 1}$$

For example, given $m = 2500, n = 600$,

if we want to make the compressed image to 80% of its original size, we need

$$k = 0.8\frac{2500 \times 600}{2500 + 600 + 1} \approx 386.97$$

which informs us that we need to keep the first $387th$ terms in the linear combination.

3 Principal Component Analysis

In many real world applications, we often need a default way to choose k that preserves the quality of images and saves memory at the same time. For example, when a user sends large images to other users, it is not realistic to expect the user to decide what k is.

Thus, Principal Component Analysis (PCA) becomes handy in this situation. Principal Component Analysis (PCA) is an algorithm used in machine learning and data mining to reduce high dimensional data to lower dimension data.

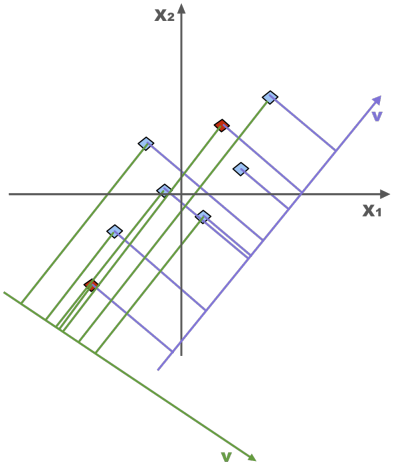
3.1 Ideas behind PCA

Consider an $m \times n$ dataset (m samples, n covariates). Suppose n is very large. Then it is very likely that some of the covariates are correlated. This means that the dataset does not span the entire n-dimensional space. For the dataset, we create an $n \times n$ covariance matrix. We want to choose a subset of the n covariates that are uncorrelated with each other. We also want this subset to preserve as much of the variability in the data as possible. This subset of covariates are called principal components.

3.2 Why more variability?

When we reduce the dimension of the data, we are essentially projecting our dataset onto lower dimensional space. We will illustrate why it is good to have large variability along projection dimension by examining the following example:

We want to reduce 2-dimensional data to 1-d, i.e. $\{x_1, x_2\} \rightarrow v$.



From the figure of above, we can see that the two red points are far away from each other in the original 2-d space. If we project our data onto the green v, the two red points end up being on top of each other. Thus, the green v does not preserve distances in the original space. This is not desirable because distances are very important as they are the basis for all machine learning algorithms. In contrast, the distance between the red points is preserved when we project the original space to the purple v. Since there has to be some extent of information loss when we reduce the dimension of the data, there are always cases where the distances are not preserved. Therefore, we want to choose k to maximize variability $\Longleftrightarrow$ minimize number of cases where the distances are not preserved.

3.3 PCA Algorithm

Algorithm 1: PCA Algorithm

Input: data matrix $\mathbf{X}$
for *each column* x_i **do**
  $x_i = x_i - \bar{x}$;
end
Now we have an adjusted matrix $\mathbf{X}'$
$\mathbf{C}$ = covariance matrix of $\mathbf{X}'$
Compute eigenvalues and eigenvectors of $\mathbf{C}$
Sort eigenvalues by absolute values in a descending order
Output: $\lambda_1, \lambda_2, ..., \lambda_n$ where $\mid \lambda_1 \mid \geq \mid \lambda_2 \mid \geq ... \geq \mid \lambda_n \mid$, eigenvectors

Note that eigenvalue with the largest absolute value indicates that data has the largest variability along its eigenvector.

3.4 Choosing the number of principal components

In section 3.2, we talked about why we need to preserve variability. But we have not formally defined what variability really means here. The formal definition of variability or proportion of explained variance is as follows:

Definition 1. Proportion of Explained Variance along the k-th eigenvector:

$$\frac{\lambda_k}{\sum_{i=1}^{n} \lambda_i}$$

Definition 2. Proportion of Explained Variance along the first k eigenvectors:

$$\frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{n} \lambda_i}$$

Empirically, we choose the minimum k that satisfies the following:

$$\frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{n} \lambda_i} \geq 99\%$$

to be the number of dimension that we want to reduce our dataset to.

3.5 Hypothesis on choosing k in SVD image compression

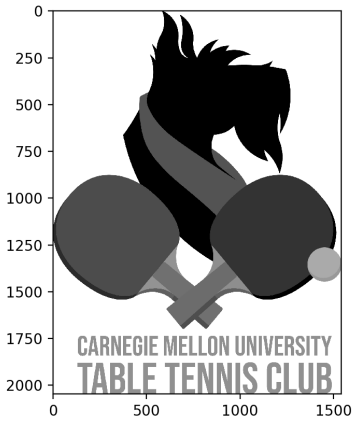
We noticed that there is some similarity between SVD and PCA. Specifically, both SVD and PCA need to compute eigenvectors and eigenvalues of input matrix and then put the eigenvalues in descending order of dominance. Thus, we speculate that if we feed an image matrix to PCA and get the eigenvalues of its covariance matrix (i.e. $\lambda_1, \lambda_2, ..., \lambda_n$) in order of dominance, then the minimum k that satisfies:

$$\frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{n} \lambda_i} \geq 99\%$$

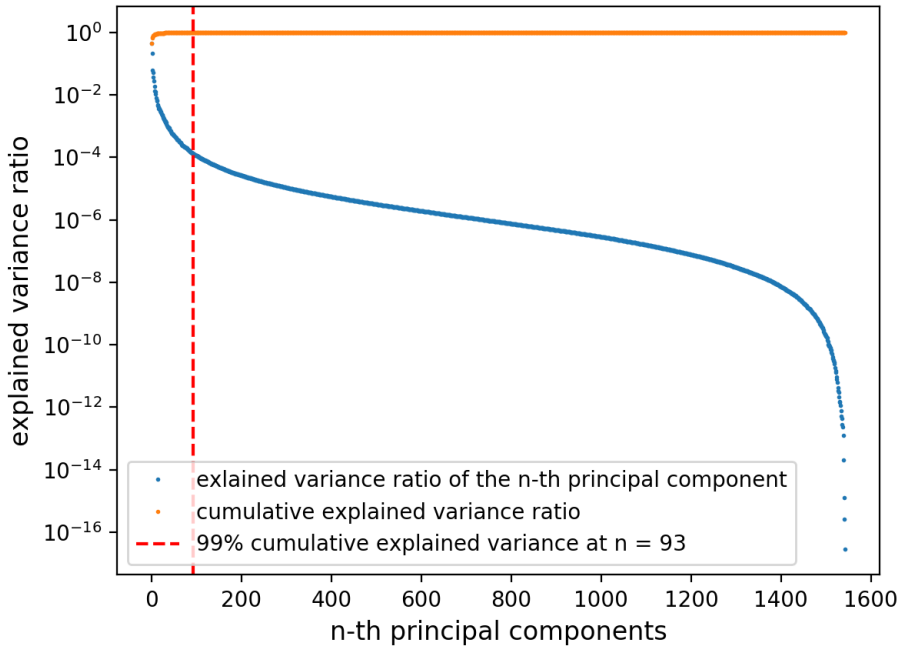
might also be a good k in SVD image compression.

3.6 Experiment

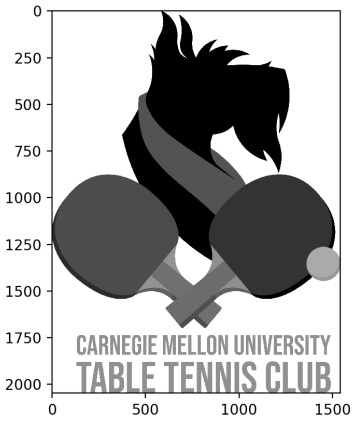
Let's test on the following image:



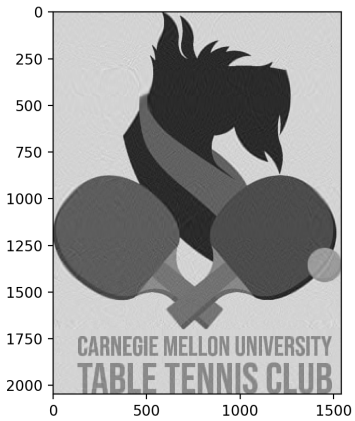
We feed this image to PCA to get the eigenvalues of its covariance matrix in order of dominance. Then we plot explained variance ratio against n-th principal component and get:



PCA tells us to set $k = 93$ for our SVD image compression algorithm. Let's see how it goes:

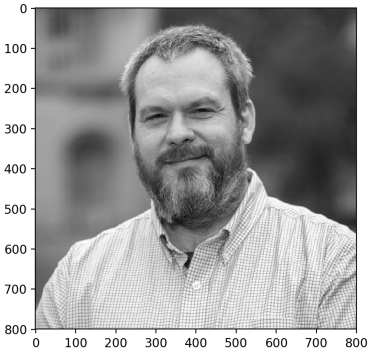


(a) original image.

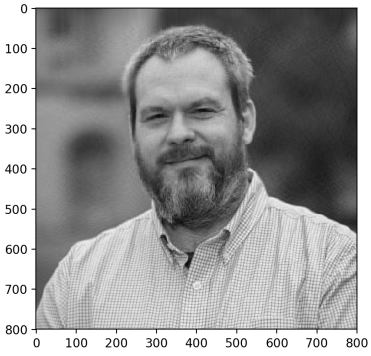


(b) k = 93.

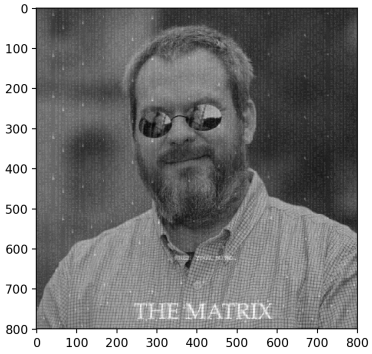
The result looks great. Let's test on more images.



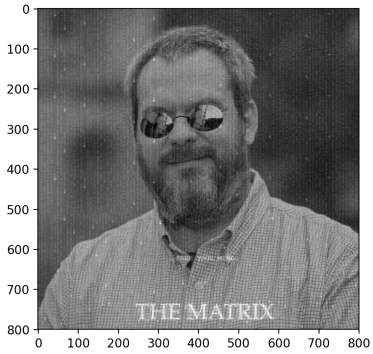
(a) original image.



(b) 99% cumulative explained variance at k = 100.



(a) original image.



(b) 99% cumulative explained variance at k = 170.

From the results above, it seems that PCA is a good way to determine the parameter k in SVD image compression algorithm.

4 Potential Future work

1. Apply SVD image compression algorithm and PCA to colored images.

2. Come up with an objective measure on quality of compressed images compared to original images.

This is very important because so far we can only judge whether the quality of compressed images is good or not by ourselves. If we do not have such measure, it would be impossible for us to test our algorithms against very large dataset of images.

3. Interpretability.

We do not know yet why PCA is a good way to decide the parameter k in SVD image compression algorithm.

4. How to set proportion of cumulative explained variance in PCA.

We empirically set the proportion of cumulative explained variance v in PCA to 99%. But sometimes, setting it to 98% or lower makes very little difference while saving even more memory. Thus, we should consider further exploring how to optimally set v in PCA.

References

[1] Sunny Verma, and Jakkam Phanindra Krishna. *Image Compression and Linear Algebra*. Chennai Mathematical institute. November 15, 2013.

[2] Frank Cleary: Singular Value Decomposition of an Image, `https://www.frankcleary.com/svdimage/`

[3] N. Biranvand, and H. Parvizi Mosaed. *Image Compression Method Based on QR-Wavelet Transformation*. Int. J. Industrial Mathematics. September 6, 2017.

[4] N. Biranvand, and H. Parvizi Mosaed. *Image Compression Method Based on QR-Wavelet Transformation*. Int. J. Industrial Mathematics. September 6, 2017.

[5] Jorge Rebaza: Image Compression, `http://people.missouristate.edu/jrebaza/assets/10compression.pdf`

[6] n.d.: Haar Wavelet Image Compression, `https://people.math.osu.edu/husen.1/teaching/572/image`$_c$*omp.pdf*

[7] Wikipedia: Principal Component Analysis, `https://en.wikipedia.org/wiki/Principal`$_c$*omponent*$_a$*nalysis*

[8] CSDN: PCA - python + numpy, `https://blog.csdn.net/u012162613/article/details/42177327`

[9] Victor Lavrenko: PCA 7: Why we maximize variance in PCA, `https://www.youtube.com/watch?v=6Pv2txQVhxA`