MAT4110 Mandatory Assignment 2

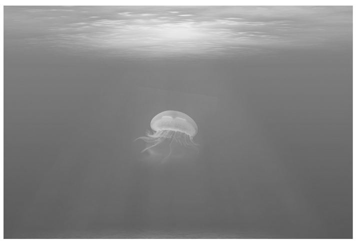
Gabriel Sigurd Cabrera

October 30, 2019

Premise

We are interested in implementing the SVD – or *singular value decomposition* – with the goal of compressing a set of three images, shown in Figure 1:

(a) New York (960×640 px)


(b) Jellyfish (960×640 px)

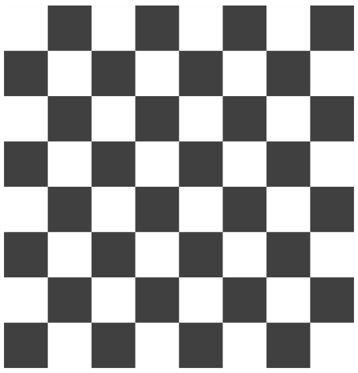

(c) Chessboard (695×720 px)

Figure 1: Three uncompressed, black and white images.

Using the `python` module `imageio`, we will represent each image as a `numpy` array of 64-bit floating-point values in the range $[0, 255]$, where 0 is white, and 255 is black.

Let us call an arbitrary input array $\mathbf{X}$; since we are performing the SVD on each $\mathbf{X}$, we will use the following notation to define the SVD:

$$\mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}} = \mathbf{X} \tag{1}$$

Where $\mathbf{X} \in \mathbb{R}^{m\times n}$, $\mathbf{U} \in \mathbb{R}^{m\times m}$, $\mathbf{\Sigma} \in \mathbb{R}^{m\times n}$, and $\mathbf{V} \in \mathbb{R}^{n\times n}$. It is also worth noting that $\mathbf{\Sigma}$ is a *diagonal matrix*; as such, we define $\mathbf{D} \equiv \mathrm{diag}(\mathbf{\Sigma}) = \begin{bmatrix}\sigma_1 & \sigma_2 & \cdots & \sigma_r\end{bmatrix}$

Compression

Using the SVD to perform compression is relatively simple – one begins by implementing the decomposition as defined in (1) in order to access the singular values σ_i given in $\mathbf{D}$. It is by saving modified versions of $\mathbf{U}$, $\mathbf{D}$, and $\mathbf{V}$ that we intend to reduce the size of our images, though clearly we must shrink them before this becomes worthwhile, as currently this would increase our file size!

To make this process worthwhile, we have three degrees of freedom by which we can reduce our matrix size:

1. Reducing the number of columns in $\mathbf{U}$ and rows in $\mathbf{\Sigma}$ by an amount Δm.

2. Reducing the number of columns in $\mathbf{\Sigma}$ and columns in $\mathbf{V}$ by an amount Δn.

3. Reducing the number of singular values in $\mathbf{\Sigma}$ from n to r.

Since we will not be saving $\mathbf{\Sigma}$, but rather its singular values σ_i, the last item is not of much consequence. In addition, if the number of columns in $\mathbf{U}$ or $\mathbf{V}$ are reduced, then the diagonal of $\mathbf{\Sigma}$ must also be reduced for matrix multiplication to be possible, causing r to have an upper-bound defined by the shapes of $\mathbf{U}$ and $\mathbf{V}$. Therefore, for the sake of conciseness, we will disregard the last item on our list[1], and focus on the first two.

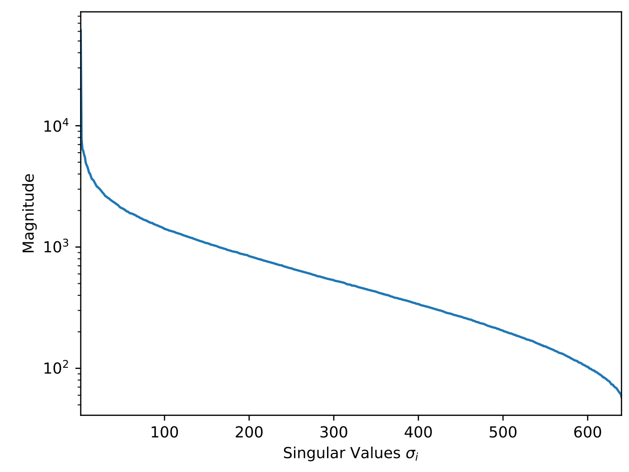
Let us now define our resized matrices $\mathbf{U}' \in \mathbb{R}^{m\times m-\Delta m}$, $\mathbf{\Sigma}' \in \mathbb{R}^{m-\Delta m\times n-\Delta n}$, and $\mathbf{V}' \in \mathbb{R}^{n\times n-\Delta n}$. This in turn implies that $\mathbf{D}' \in \mathbb{R}^{n-\Delta n}$, given the assumptions we made.

It is important to note that $\sigma_1 > \sigma_2 > \cdots > \sigma_r > 0$, since we can assign a higher significance to the largest singular values: $\mathbf{D}'$ will therefore contain the first $n - \Delta n$ elements of $\mathbf{D}$. This in turn implies that we should remove the rightmost columns of $\mathbf{U}$ and $\mathbf{V}$, when creating $\mathbf{U}'$ and $\mathbf{V}'$.

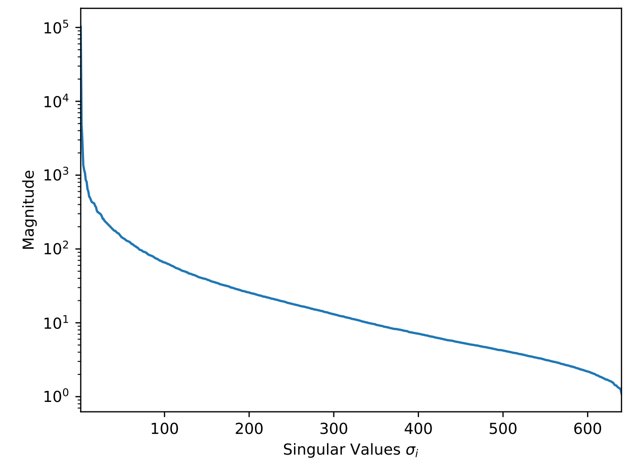
Singular Values

Using `python` to implement the SVD for each image, we see in Figure 2 that the singular values decrease at rates likely dependent on the complexity of the image.

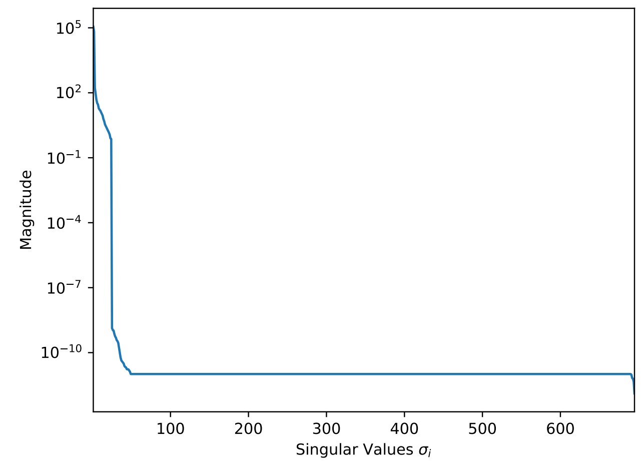
[1] We will assume that the length of $\mathbf{D}$ is equal to the number of columns in $\mathbf{U}$ or $\mathbf{V}$, depending on which one is smallest.



(a) New York



(b) Jellyfish



(c) Chessboard

Figure 2: The singular values of each decomposed image.

We see that the photographs both have very smoothly decreasing singular values (Figures 4a and 4a), while the computer generated chessboard has *very few* large singular values (Figure 4c.) We can therefore conclude that the chessboard image will be far more compressible than the photographs, given that image quality must remain acceptable.

Compression Ratio

The compression ratio R is a measure of how compressed an image has become, with 1 being the uncompressed size, with a larger R being more desirable than a smaller R. It is defined as follows:

$$R = \frac{\text{uncompressed size}}{\text{compressed size}}$$

By summing over the sizes of our decomposed and reduced matrices, we can represent this ratio as a function:

$$R(m, \Delta m, n, \Delta n, r) = \frac{mn}{m(m - \Delta m) + r + n(n - \Delta n)}$$

Though given that r is bound by our other values, it may be best to define $r \equiv \min(m - \Delta m, n - \Delta n)$, since this is the optimal value with regards to both image quality and compression effectiveness. With this in mind, we see in Figure 3 how the compression ratio varies for an image the size of Figure 1a:

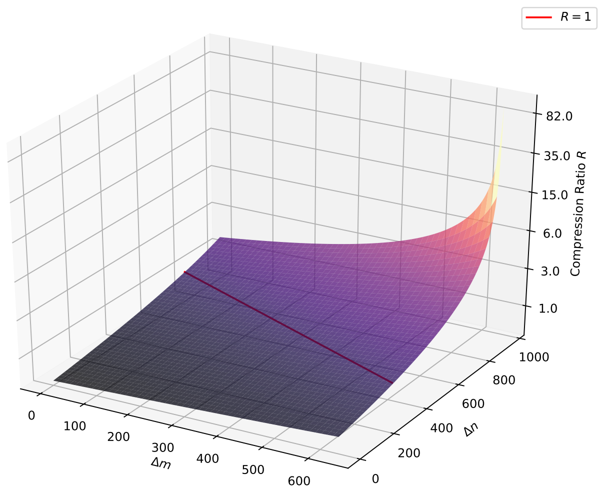


Figure 3: The compression ratio as a function of Δm and Δn, with $m = 640$, $n = 960$.

It is very important to consider, however, that the *image quality* is much more difficult to measure, and will require a manual analysis.

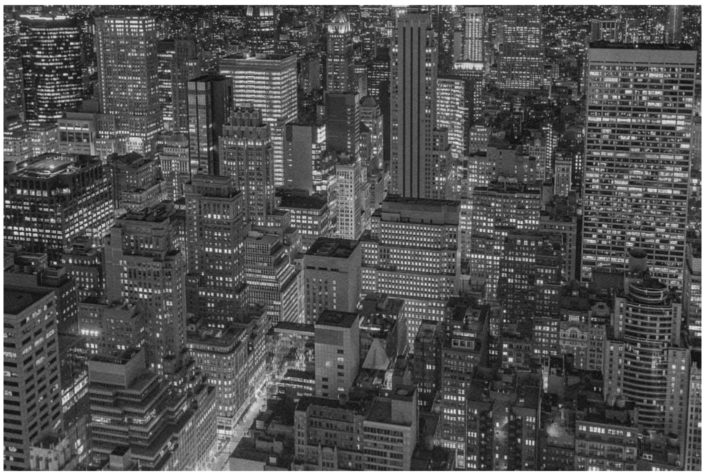
Results

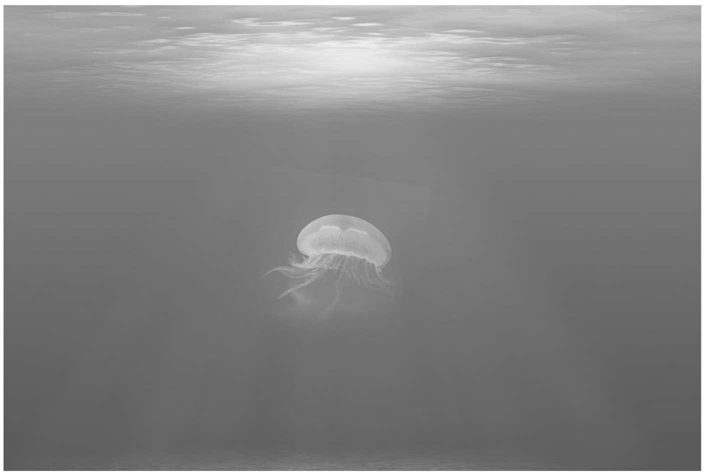
Implementing the methods aligned in the previous sections, we found that each image differed in acceptable compressability. Through trial and error, we found the following values to be acceptable:

Image	**Label**	Δm	Δn	**Compression Ratio**
Fig. 1a	New York	330	700	1.37
Fig. 1b	Jellyfish	500	700	1.81
Fig. 1c	Chessboard	718	691	118.52

Table 1

The compressed images themselves can be seen in Figure 4:


(a) New York


(b) Jellyfish

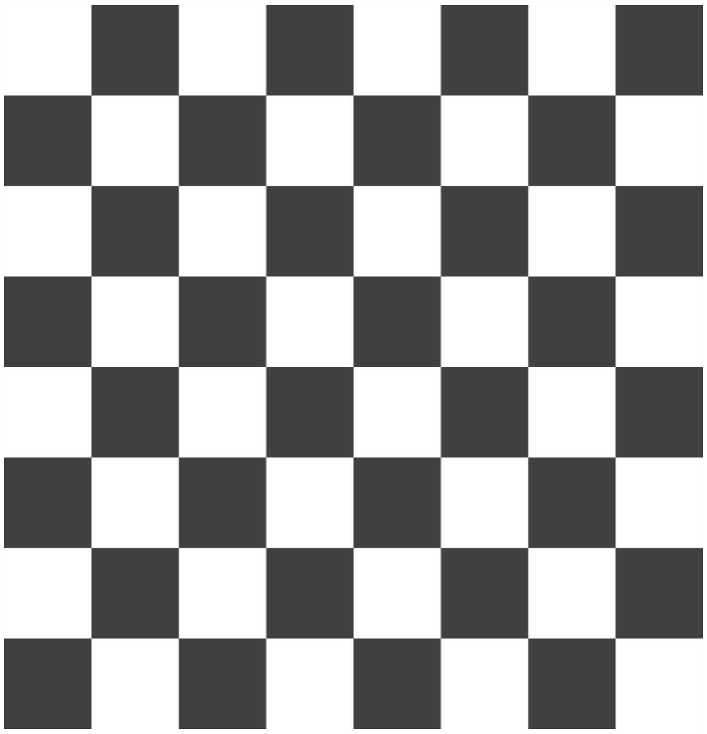

(c) Chessboard

Figure 4: The images from Figure 1, each compressed according to the values given in Table 1.

We can therefore conclude that the SVD is highly effective at compressing images that are simple, such as our digitally rendered chessboard (Figure 1c); we have also seen that photographs containing a lot of detail (such as Figure 1a) will suffer greatly in terms of quality, with only a minor improvement in file size – less detailed photos (as in Figure 1b) can be compressed slightly more effectively.

Appendix

The script used to compress the images is available online at:

https://github.com/GabrielSCabrera/MachineLearning/blob/master/MAT4110/Oblig_2/oblig_2.py