This excellent page **https://timbaumann.info/svd-image-compression-demo/** showing SVD image compression was created by Tim Baumann.

Uncompressed image. Slider at 300.

IMAGE SIZE 600×600
#PIXELS $= 360000$

UNCOMPRESSED SIZE
proportional to number of pixels

COMPRESSED SIZE
approximately proportional to
$600 \times 300 + 300 + 300 \times 600$
$= 360300$

COMPRESSION RATIO
$360000/360300 = 1.00$

Show singular values

10 20 100 200 300 400 500 600

Compressed image. Slider at 20.

IMAGE SIZE 600×600
#PIXELS $= 360000$

UNCOMPRESSED SIZE
proportional to number of pixels

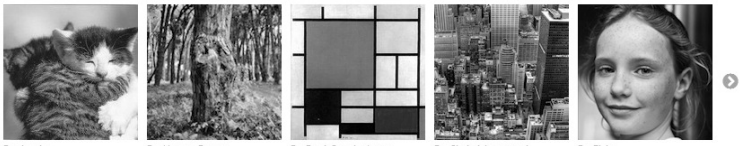
COMPRESSED SIZE
approximately proportional to
$600 \times 20 + 20 + 20 \times 600$
$= 24020$

COMPRESSION RATIO
$360000/24020 = 14.99$

Show singular values

10 20 100 200 300 400 500 600

Change the number of singular values using the slider. Click on one of these images to compress it :



You can compress your own images by using the file picker or by dropping them on this page.

The Application of Singular Value Decomposition to Image Compression

Hanquan (Harry) Qiao

1 Introduction

Singular Value Decomposition (SVD) is one of the most important concepts in linear algebra. For anybody who is taking a linear algebra course, the words eigenvalue and eigenvector should be familiar. For symmetric matrices S, the eigenvalues and orthogonal eigenvectors are not difficult to compute. However, to extend this to a rectangular matrix A requires a different approach.

SVD is a continuation of eigendecomposition that can be used on any rectangular matrix A. Just like the many important factorizations of matrices ($A = CR$, $A = LU$), SVD breaks matrix A into rank-one pieces and in addition puts rank-one pieces in order of importance. The idea of using SVD to compress matrices comes naturally, since the goal is to compress the matrix while maintaining the most important features.

With the advancement of technology and the arrival of the era of big data, the use of images has skyrocketed. Whether it is medical imaging used in patients' treatment [15, 17], data used for recognition in machine learning [2, 6], or other purposes [8, 9], the large amount of digital images not only requires a substantial amount of energy to process, it also occupies valuable memory space. Since images are large matrices, SVD can be a useful tool to compress images while maintaining an acceptable quality.

This paper will examine how SVD is used in image compression. Specifically, it will investigate the effect of using SVD on different matrices and images with code developed by Tim Baumann and published online (http://timbaumann.info/svd-image-compression-demo/) [1].

2 Tim Baumann's Page

Since a large portion of this paper involves using Tim Baumann's webpage, it is necessary to explain how the page functions. Fig. 1 is the setup of the page:

There are 15 images built into this page. Ten of these are stock images, easily found online, and selected by using the right and left arrows and clicking on an image. The other five are identified only with question marks and contain images that can only be seen by using the demo. Users can upload their own images, using the "file picker" button, and experiment on them with SVD. Uploading images that are larger than 1000×1000 pixels is not recommended, because they slow down the computer.

Once an image is selected, users can adjust the slider left or right to experiment how many singular values are required to get a clear image. On the right-hand side of the image, the calculation of compressed size and compression ratio will change as the slider is moved. The line of calculation under "compressed size" also shows how many singular values are selected. There is a checkbox at the bottom right of the displayed image next to "hover to see the original picture." Checking the box enables

comparison of the compressed image to the original to see the difference. The singular values of the image can be shown by hitting the “show singular values” button. Details regarding what happens when the “show singular values” button is selected will be explained later in this paper with an example.

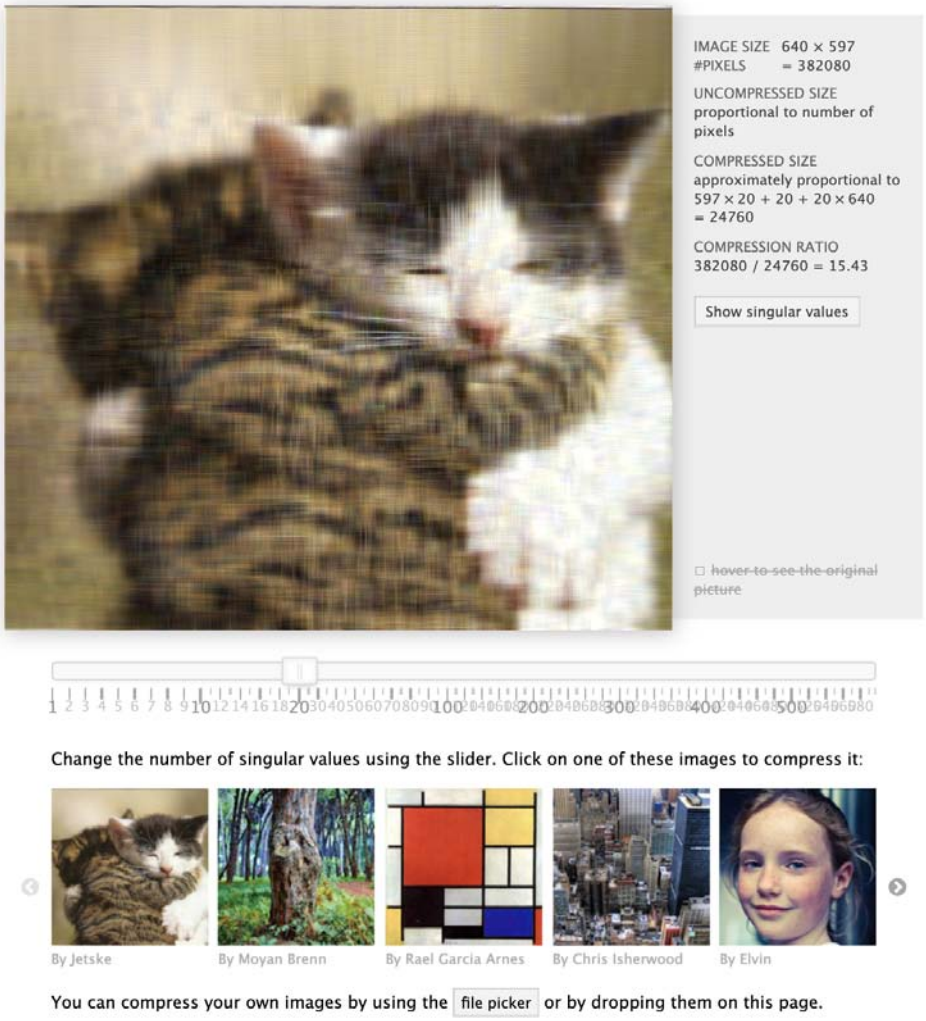


Figure 1. Tim Baumann’s webpage

3 SVD on Low-Rank Matrix

Knowing that the key feature of SVD is to break large matrices into a sum of rank 1 pieces in order of decreasing importance, let’s first look at some simple matrices. SVD works well on low-rank matrices just because of their very nature, as seen in the following examples:

Rank 1 matrix

Matrices of rank 1 are in themselves rank 1 pieces: $A = \sigma_1 u_1 v_1^T$. A rank 1 matrix A has 1 independent column and one independent row. If rank 1 matrix A has dimension $m \times n$, then it is

possible to factor $A = CR = (m \times 1)(1 \times n)$. By doing so, the total number of mn entries in A has been replaced by a reduced number of $m + n$ entries, while still maintaining the values of each entry in matrix A. While there are not many rank 1 matrices in the natural world, there are a few. There are, in fact, more than 35 countries with flags consisting only of stripes, some vertical and some horizontal, with different color combinations [3]. On page 270 of Gilbert Strang's *Linear Algebra for Everyone*, there is a well-crafted example explaining why these pictures can be compressed effectively using SVD. Tim Baumann's published SVD demo can be used on a picture of the Russian flag.

Figure 2a. Russian flag compressed to 1 singular value

Figure 2b. Russian flag compressed to 10 singular values

Figure 2c. Russian flag

The number of singular values can be selected by adjusting the slider. In this specific case, the Russian flag is an image of rank 1. There is only 1 nonzero singular value. That is why Fig. 2a looks exactly like Fig. 2b and Fig. 2c. The main difference here is the compression size shown in the top right corner. The original image has $1200 \times 800 = 960000$ pixels (entries in a 1200×800 matrix). After the image is compressed with 1 singular value selected, it is a product of three matrices which only has $1200 \times 1 + 1 + 1 \times 800 = 2001$ entries, calculated from the equation $A = U\Sigma V^T$. The new compressed image has only $\frac{1}{479.76}$ the size of the original image, while keeping all the features.

Other countries have flags consisting of colored stripes and symbols. For example, the Chinese flag has five yellow stars on the top left corner, the Japanese flag has a red circle on a white background, and the flag of Israel has a blue Star of David (a hexagram) between two blue lines. Those symbols make the rank infinite, but compression is still possible, as will be discussed later in this paper.

Low-Rank Matrix

Although low-rank matrices are a bit more complicated than rank 1, they can be compressed as well. Take the flags of Finland and Greece, for example:

Figure 3. The flag of Finland

Figure 4a. The flag of Greece

Fig. 3 on the left is rank 2. There are two different columns and two different rows. At first glance, Fig. 4a on the right has five different rows, while it has only three different columns. Since the number of independent columns always equals the number of independent rows, the flag must be rank 3. Taking a closer look, though, some rows seem to be the sum of two other rows. If the rows are numbered by looking at the alternating blue and white stripes on the right of the flag, row 1 will appear to be the

first blue stripe, row 2 the first white stripe, and so on. However, row 1 is actually the sum of row 2 and row 3. This means that in the first five rows there are only two independent rows. Though row 6 and row 8 have different colors than row 7 and row 9, the four rows are all full stripes, which means there is only one independent row. This is how it looks with SVD compression:

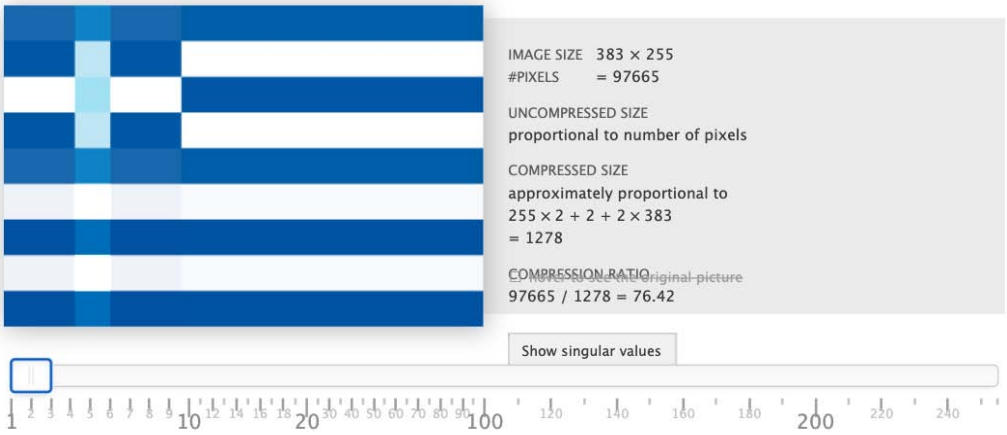


Figure 4b. The flag of Greece compressed to 2 singular values

Clearly, Fig. 4b is not a good compression. Features such as the cross at the top left are lost because the compression is only rank 2. The flag of Greece is rank 3. If three singular values are selected on the slide bar, the compressed image should look just like the original.

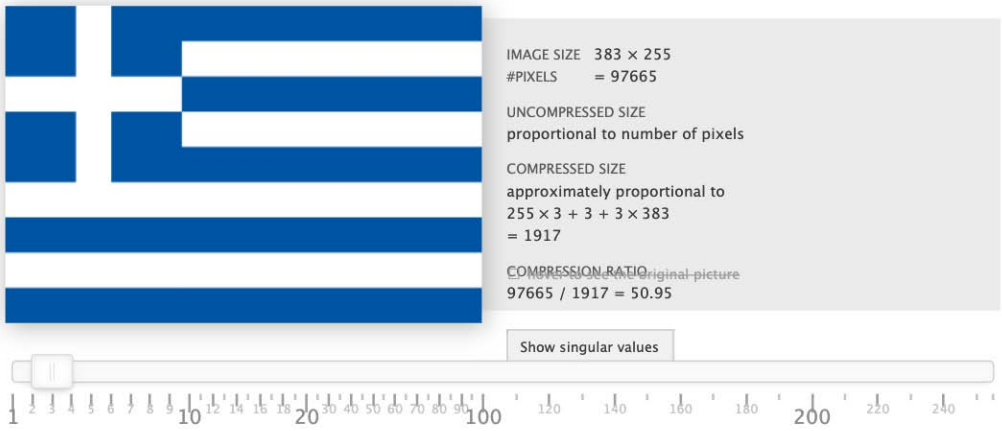


Figure 4c. The flag of Greece compressed to 3 singular values

Just as expected, the compressed image in Fig. 4c is perfect compared to the original. The size (entries) is reduced from 97665 to 1917, $\frac{1}{50.95}$ of the original number.

There are other real-life images besides national flags that behave similarly. Some of them might not be exactly rectangular, but only a few singular values are required for a good SVD. Tartan is a fabric pattern that consists of mostly horizontal and vertical color bands. On Wikipedia, there are more than 80 different tartans, including the well-known Royal Stewart tartan and the Burberry check. Although the color bands are not strictly rectangular (due to the weaving technique), it is very compressible. One of the 15 pictures built in the demo is the Royal Stewart tartan pattern. Applying SVD to this image shows the following:

Figure 5a. Royal Stewart tartan

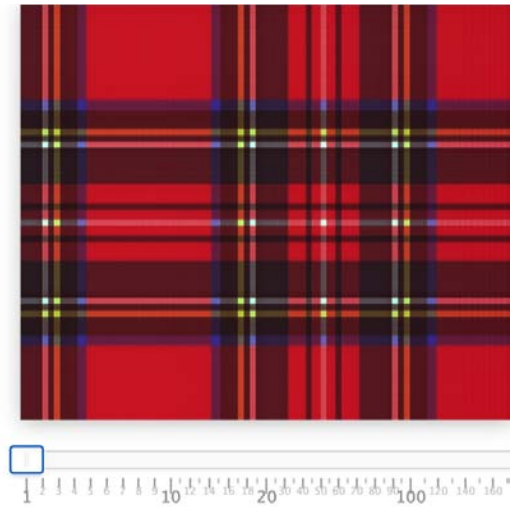


Figure 5b. Tartan compressed to 1 singular value

Figure 5c. Tartan compressed to 3 singular values

Figure 5d. Tartan compressed to 5 singular values

Figure 5e. Tartan compressed to 10 singular values

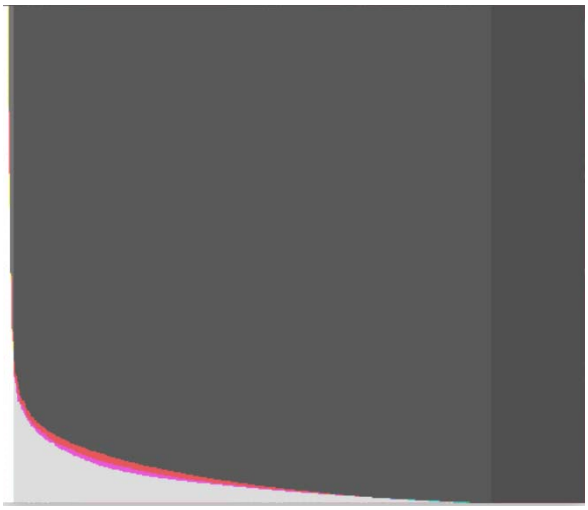


Figure 5f. Tartan singular value graph

From Fig. 5b to Fig. 5e, the tartan image was compressed to 1, 3, 5, and 10 singular values. The arbitrary limit of 10 singular values was selected because it results in a compressed image that is hard to distinguish from the original picture (Fig. 5a) with the naked eye.

Baumann's demo also allows the selection of "show singular value," which displays the calculated singular values. Although the y-axis is not labeled, the curve shows the descending pattern of the singular values. Fig. 5f was made using selected 10 singular values, with the band of grey starting on the left of the graph. The original image has size 481×404 pixels. There are at most 404 singular values possible. The total width of the graph, therefore, is 404, and the grey coloring begins at 10, the number selected. The curve is very steep and begins to turn right around where the grey color begins. This means that even if $\sigma_j u_j v_j^T$ is omitted for $j \geq 11$, the loss of $A = U\Sigma V^T$ would be minimal, because σ_j is so small. The narrow band of magenta between white and dark grey is the result of the demo's design; since the demo is designed to handle all images, it must be able to perform SVD on RGB images. The code first breaks the RGB image into three separate matrices (separate R, G, and B) and performs SVD on each individually. Then it reconstructs the three matrices into a colored image again. In this case, the singular values of the red and blue channel matrices are greater than the singular values of the green channel matrix, hence the thin band of magenta. In other images, for example, narrow bands of cyan, yellow, or even red, green, or blue might appear. What the demo does is graph the singular values of the three different matrices (RGB), sketch three smooth curves through those points, and color the area below the curves accordingly.

4 SVD for Matrices with Geometric Shapes

Now that the full function of Tim Baumann's demo has been described, the investigation of using SVD on matrices and images can proceed in greater depth. As discussed above, national flags, such as the Chinese flag, the Japanese flag, and the flag of Israel, all have symbols that make the rank infinite. Many images in our daily life contain simple shapes such as triangles, circles, stars, or just diagonal lines. These shapes are all full rank, but some might compress more efficiently than others.

Of those three flags mentioned, the Chinese flag has five yellow stars on the top left corner, the Japanese flag has a red circle on a white background, and the flag of Israel has a blue Star of David (a hexagram) between two blue lines. Circle and star are very primitive shapes and the Star of David is formed with horizontal and diagonal lines. It is important to know the result of using SVD to compress primitive shapes before compressing more complex matrices. To better understand the compression of those shapes, a code was written that allowed loading images of primitive shapes and converting the area of the shapes to 0s and the background to 1s. The singular values were calculated and plotted on graphs with regular scale and log scale y axis. Below are a solid and a hollow equilateral triangle, a 45-45-90 triangle, a circle, a solid and a hollow hexagon, and lastly, a star:

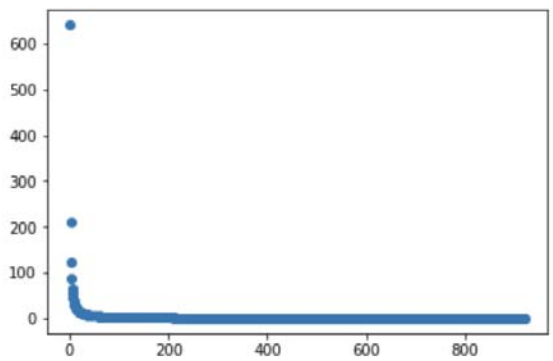


Figure 6a. Solid equilateral triangle

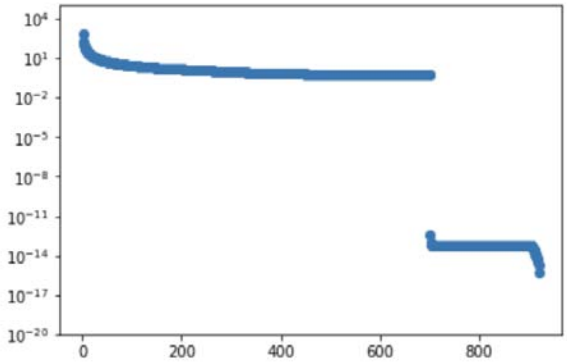


Figure 6b. Solid equilateral triangle

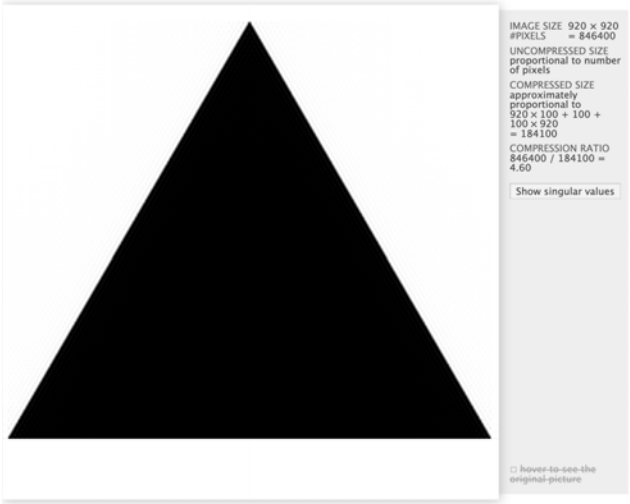


Figure 6c. Equilateral triangle compressed to 100 singular values

Figure 6d. Details of Fig. 6c

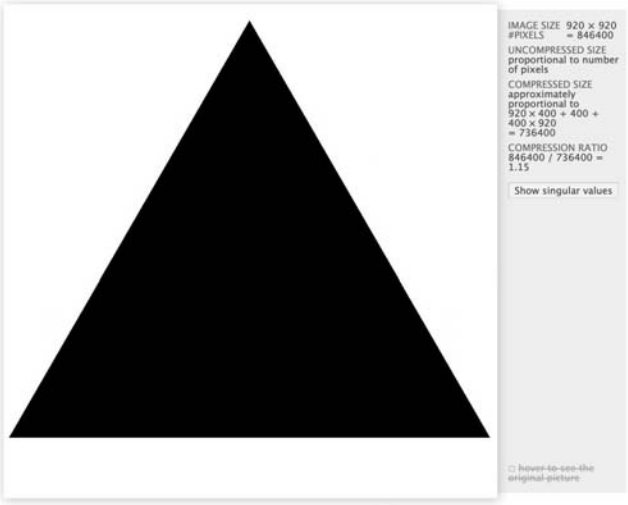


Figure 6e. Equilateral triangle compressed to 400 singular values

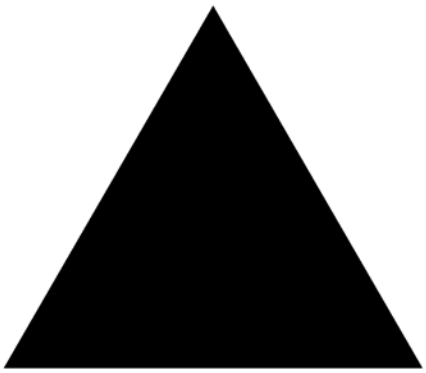


Figure 6f. Equilateral triangle

Fig. 6f is an image of an equilateral triangle. With the code, the image was converted to a 920×920 matrix with the 0s in the triangle area and 1s in the background. The 920 singular values of the matrix are calculated and plotted on a normal scale in Fig. 6a and on a log scale in Fig. 6b. While the normal scale graph shows a pattern of decreasing singular values, the log scale graph suggests that the singular values decrease at a very slow rate between 10^{1} to 10^{-2} until σ_j where j is around 700. This means that with fewer than 700 singular values for an image compression, the image might lose some significant features. In the demo, the original image of the equilateral triangle was uploaded. Driven by the graph of singular values, 100 singular values were selected at first. Although the compression doesn't look bad (Fig. 6c), the image shows many thin and dim stripes of black parallel to the two sides of the triangle (Fig. 6d). It was not until 400 singular values were selected that the tiny stripes became invisible to the naked eye (Fig. 6e).

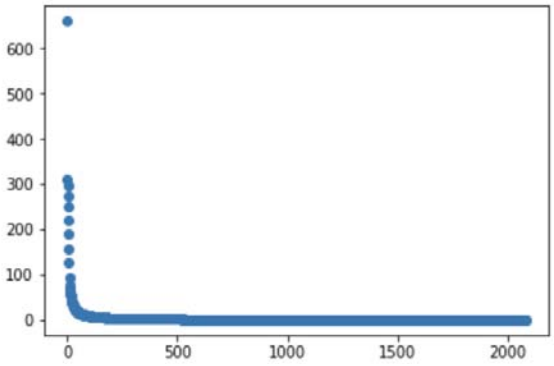


Figure 7a. Hollow equilateral triangle

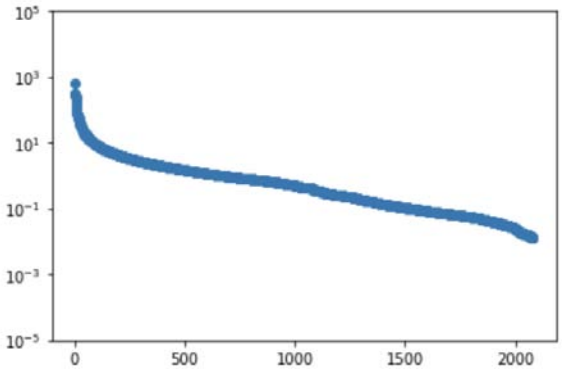


Figure 7b. Hollow equilateral triangle

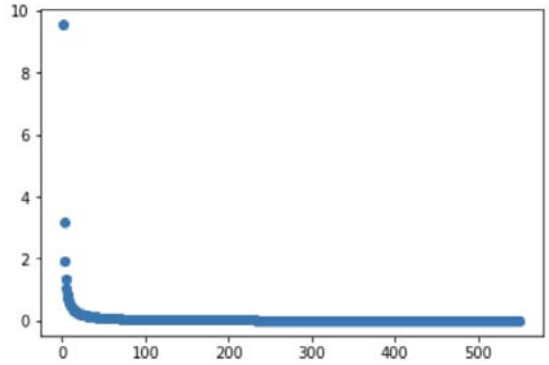


Figure 8a. Solid 45-45-90 triangle

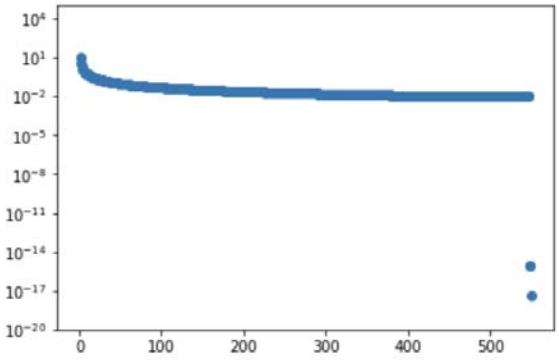


Figure 8b. Solid 45-45-90 triangle

Fig.7a and Fig. 7b depict the singular values of a hollow equilateral triangle, and Fig.8a and Fig. 8b depict the singular values of a solid 45-45-90 triangle matrix with the triangle positioned like that of a lower triangular matrix. Both sets show a pattern of slowly decreasing singular values between 10^{1} to 10^{-2} after the first 10% of all singular values. In fact, the singular values of lower triangular matrix of 1s and 0s with diagonal 1s can be found using the formula $\sigma_k = [2sin(\frac{\pi(2k-1)}{2(2n+1)})]^{-1}$ where n is the size of the matrix [11]. From this formula, we get $\sigma_n \geq \frac{1}{2}$: every singular value of a lower triangular matrix of 1s and 0s with diagonal 1s is greater than $\frac{1}{2}$ [11, 12].

The only difference among the three sets of graphs is the end patterns. For the solid equilateral triangle, there are more than 200 singular values ranging from 10^{-14} to 10^{-17}. However, there are only two data points in that range from the graph of the 45-45-90 triangle, and nothing from the graph of the hollow equilateral triangle. One explanation is that the original images of the equilateral triangle have a large stripe of white above and below the black triangle that might affect the calculation of singular values. The image of the 45-45-90 triangle has a tiny stripe of white below it that was not removed. The image of the hollow triangle has no extra white above and below the triangle. Perhaps those extra rows of 1s in the converted matrix created those very small singular values at the end.

Using the demo, these two images of triangles also show the similar thin stripes parallel to the sides that are not horizontal or perpendicular. In this case, it is possible to conclude that triangles are difficult to compress using SVD. In Fig. 9a, an image of the flag of South Africa of 900×600 pixels requires at least 300 singular values for a good compression (that is to say, no visual difference from the original image). The compression size with 300 singular values is $900 \times 300 + 300 + 300 \times 600 = 450300$, only decreased from the original $900 \times 600 = 540000$ by a factor of 1.20.

Figure 9a. Russian flag compressed to 300 singular values

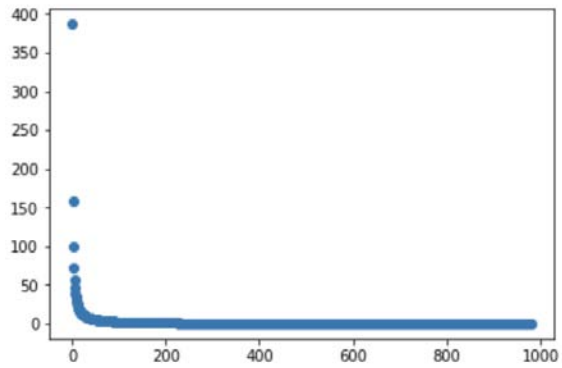


Figure 10a. Circle

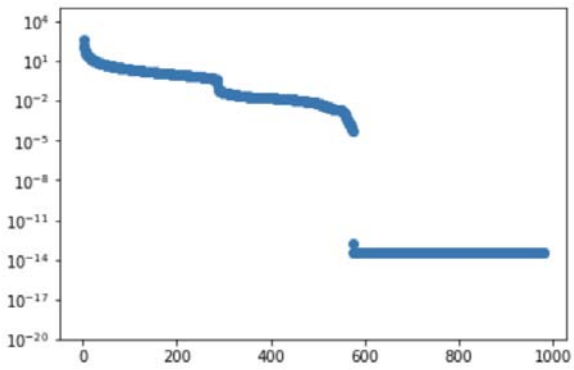


Figure 10b. Circle

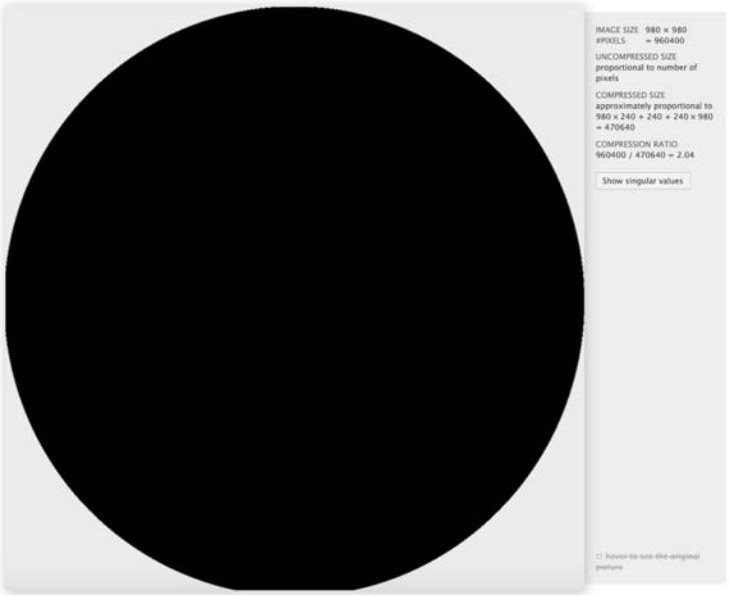


Figure 10c. Circle compressed to 240 singular values

For an image of a circle, the singular value plotted on the log scale graph (Fig. 10b) shows an interesting decreasing pattern. While the first 300 singular values show slow descent around 10^0, there is a sudden drop and then a slower decrease until around the 600 singular value, when the singular values begin to rapidly drop again. From the graphs of the singular values, it is reasonable to guess that the image of a circle compresses considerably better than that of a triangle. MIT Opencourseware, lecture 17, course 18.065 includes a method of finding the rank of a circle which shows that the rank of a circle is $\frac{1}{2}r + 1$ where r is the size of the matrix. This result is confirmed by the graphs of the singular values. Although the singular values in the log graph don't rapidly decrease until around 600, the curve really becomes steep at around 500. Theoretically, $\sigma_j = 0$ where $j > \frac{1}{2}n + 1$, but because the image is not perfect and there are some values between 0 and 1 on the edge of the circle, the result is acceptable. In the demo, a good compression (Fig. 10c) can be achieved with only 240 singular values. SVD compresses the image by a factor of 2.04 and potentially higher if more error is allowed.

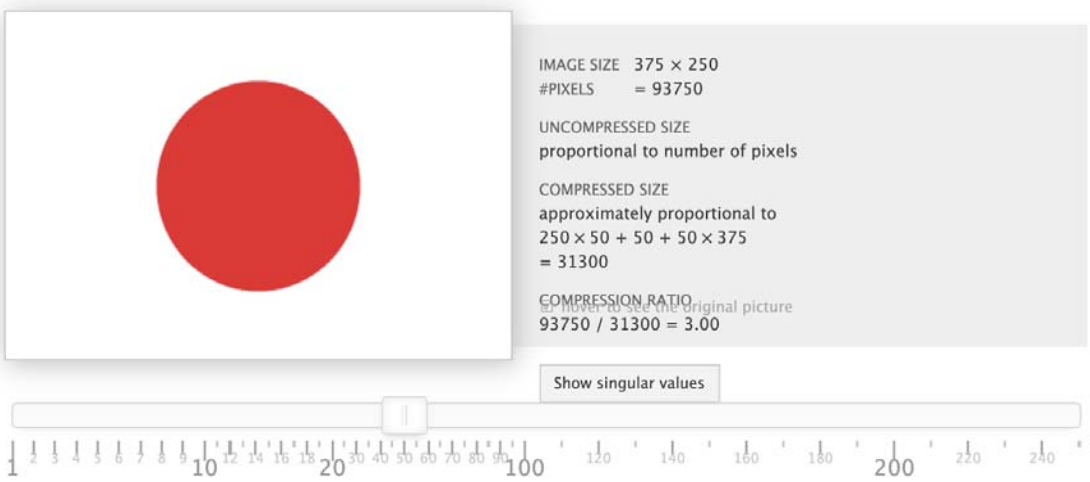


Figure 11a. The Japanese flag compressed to 50 singular values

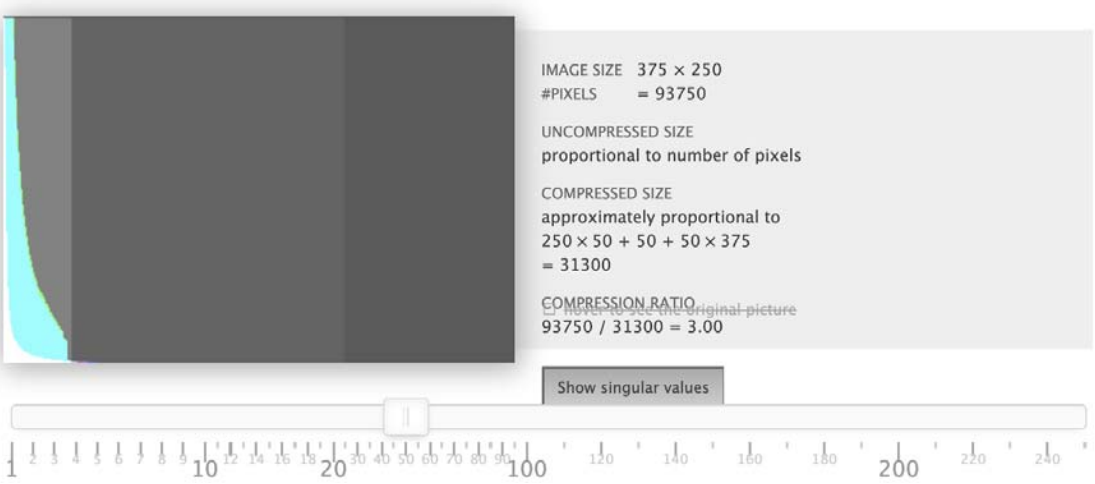


Figure 11b. The Japanese flag singular value graph

An almost perfect compression of the Japanese flag, a red circle on a white background, requires only 50 singular values. Better, only about one third of the 93750 entries from the matrix of the original image has to be stored.

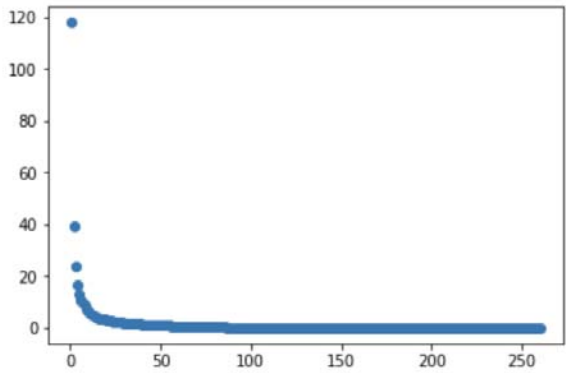


Figure 12a. Solid hexagon

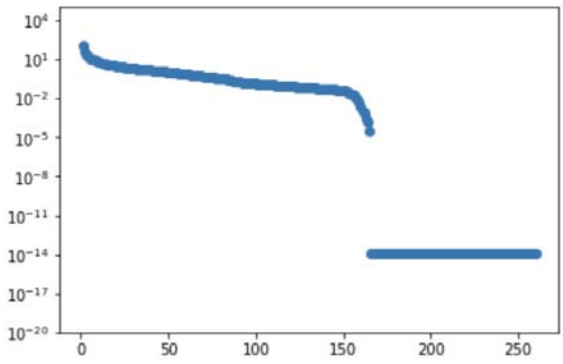


Figure 12b. Solid hexagon

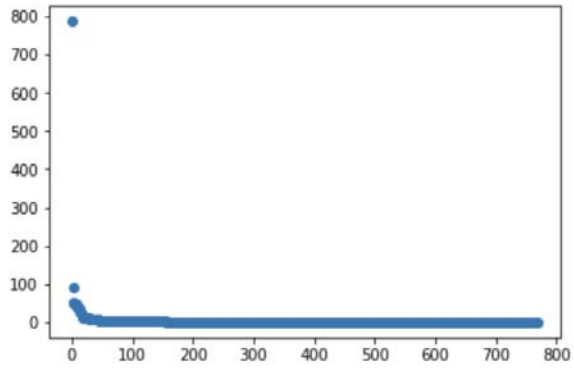


Figure 13a. Hollow hexagon

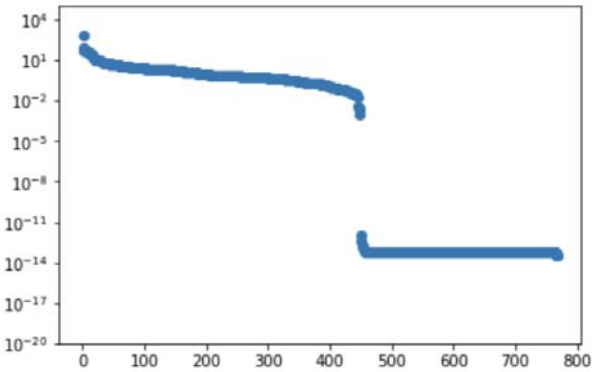


Figure 13b. Hollow hexagon

Fig. 12a and Fig. 12b depict the singular values of a solid regular hexagon, and Fig. 13a and Fig. 13b depict a hollow regular hexagon. From the conclusion above about triangles, it is reasonable to anticipate that a hexagon would not be very easy to compress, since it has four diagonal lines. The two

sets of graphs of the solid and hollow regular hexagon show a similar decreasing pattern. The pattern of decreasing is slow between 10^{1} to 10^{-2}. Rapid decreasing doesn't occur until about the last 40% of the singular values.

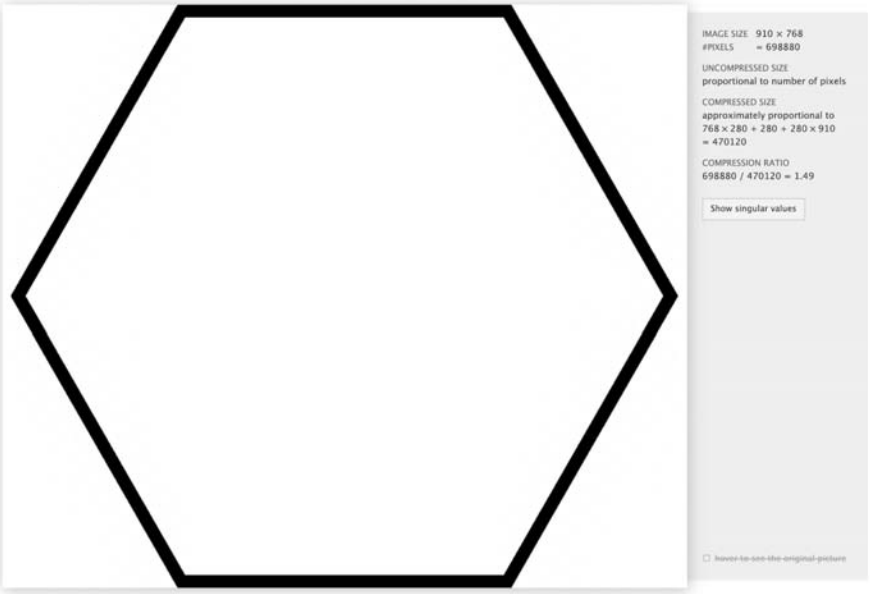


Figure 13c. Hollow regular hexagon compressed to 280 singular values

In Fig. 13c, the hollow regular hexagon needs about 280 of the 768 singular values to achieve a very good compression. Still, the compression size with 280 singular values is $768 \times 280 + 280 + 280 \times 910 = 470120$, decreased from the original $768 \times 910 = 698880$ by a factor of 1.49. In conclusion, hexagons are inefficient to compress using SVD: more efficient than triangles but less efficient than circles. Moreover, after using the demo on dozens of convex polygons, including regular n-gons and irregular n-side polygons, a conclusion can be made. Convex polygons with more horizontal or vertical edges, such as a regular octagon, are more efficient to compress with SVD, whereas shapes such as random triangles, regular pentagons, regular heptagons, and enneadecagons, on the other hand, are inefficient to compress with SVD.

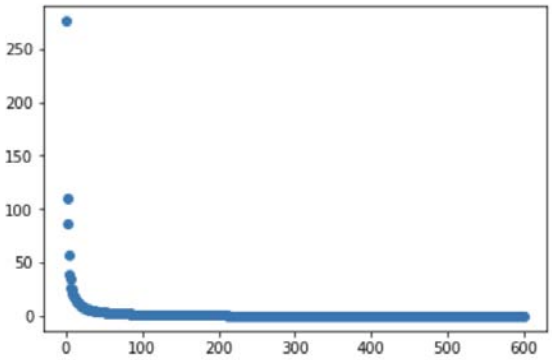


Figure 14a. Star

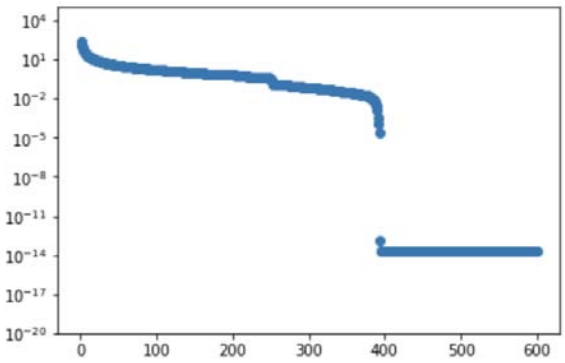


Figure 14b. Star

The last shape investigated is a star, an example of a concave polygon. Just like that of convex polygons, the singular values of the star (Fig. 14a and Fig. 14b) show a slow initial decrease. The

concavity of the polygon isn't a deciding factor here at all. The demo shows similar results. The image of the star is not really efficient to compress with SVD.

5 SVD on Random Matrix

Random matrices are widely studied by mathematicians and physicists. Random matrices are very intriguing in the sense that all the entries in the matrix are randomly generated. The rank of random matrix and random binary matrix has been investigated before [5, 7]. It is possible for a random matrix to have dependent columns and rows. In the section below, the efficacy of SVD on random matrices will be examined. To better understand the compression of a random matrix, a code was written that can generate a random square matrix, plot the singular values and convert the matrix into an image for Tim Baumann's demo. In the section below, random binary matrices and random matrices will be analyzed. The first random matrices analyzed are random binary square matrices of different sizes. The singular values are plotted with regular scale and log scale y axis.

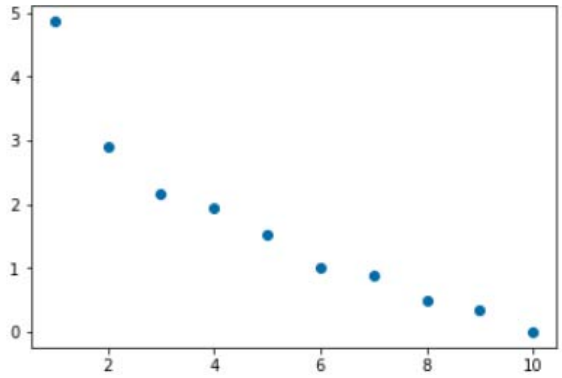


Figure 15a. Random binary matrix of size 10

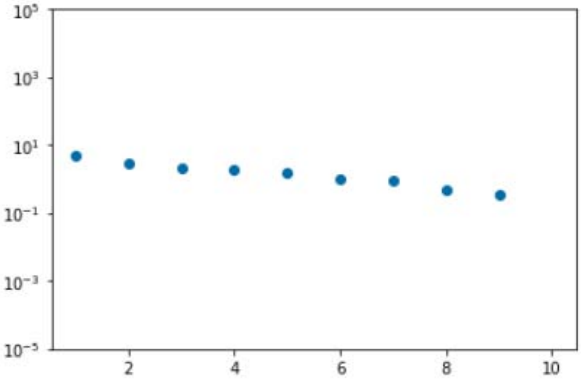


Figure 15b. Random binary matrix of size 10

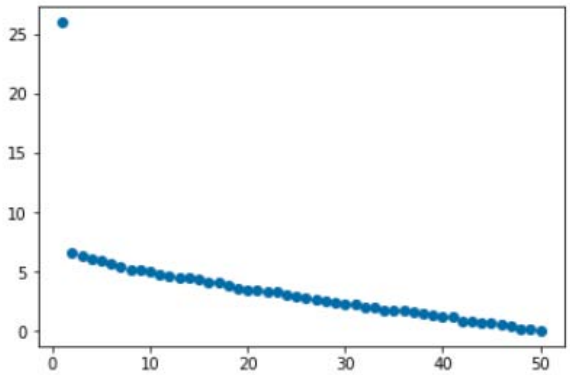


Figure 16a. Random binary matrix of size 50

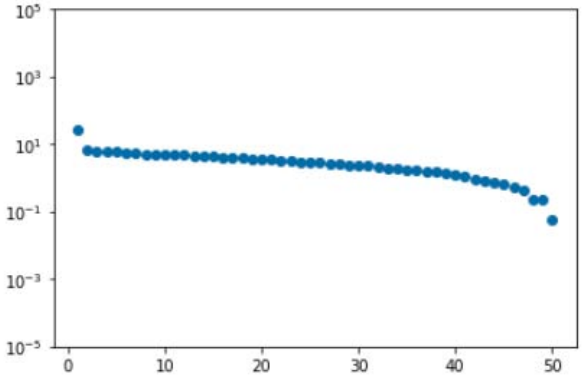


Figure 16b. Random binary matrix of size 50

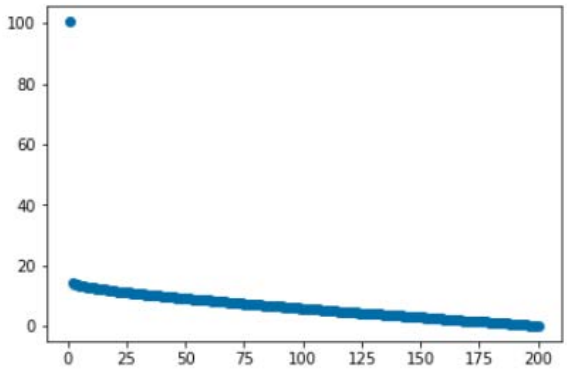


Figure 17a. random binary matrix of size 200

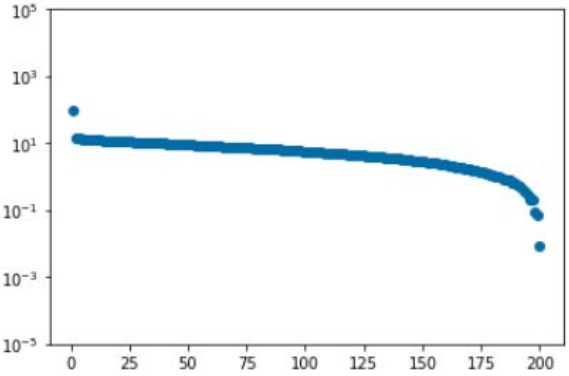


Figure 17b. random binary matrix of size 200

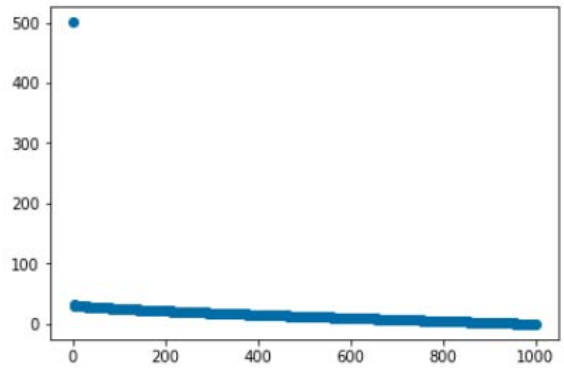


Figure 18a. Random binary matrix of size 1000

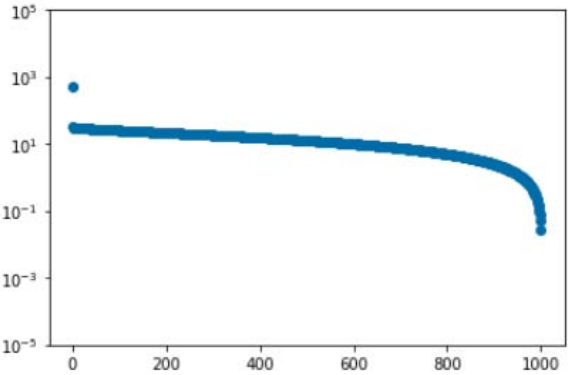


Figure 18b. Random binary matrix of size 1000

Here, the singular values of four random binary square matrices of different sizes are calculated and plotted from Fig. 15 to Fig. 18. As the size of the matrix increases, the descending pattern of the singular values becomes highly consistent, especially in Fig. 17b and Fig. 18b. The descending pattern is almost exactly the same. On the other hand, every time a new random binary matrix of size 10 is generated, the singular value graph has visible differences. Occasionally, the final singular value or singular values is 0 because there are dependent columns and rows in the matrix. Although the descending curve in the log scale graphs is very flat, it is decreasing faster and faster, unlike the singular value graphs of a triangle (Fig. 6, 7, and 8). Now the question is how effective it is to apply SVD onto a random binary matrix. For better visualization, the entry of a random binary matrix of size 200 is multiplied by 255 and the size of the image is enlarged by a factor 2.

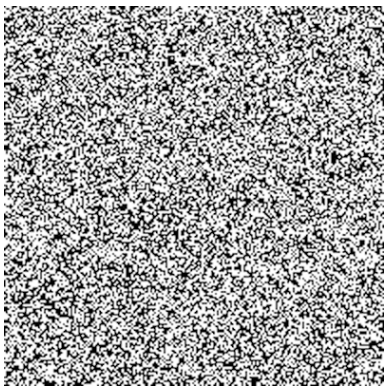


Figure 19a. Image of random binary matrix of size 200

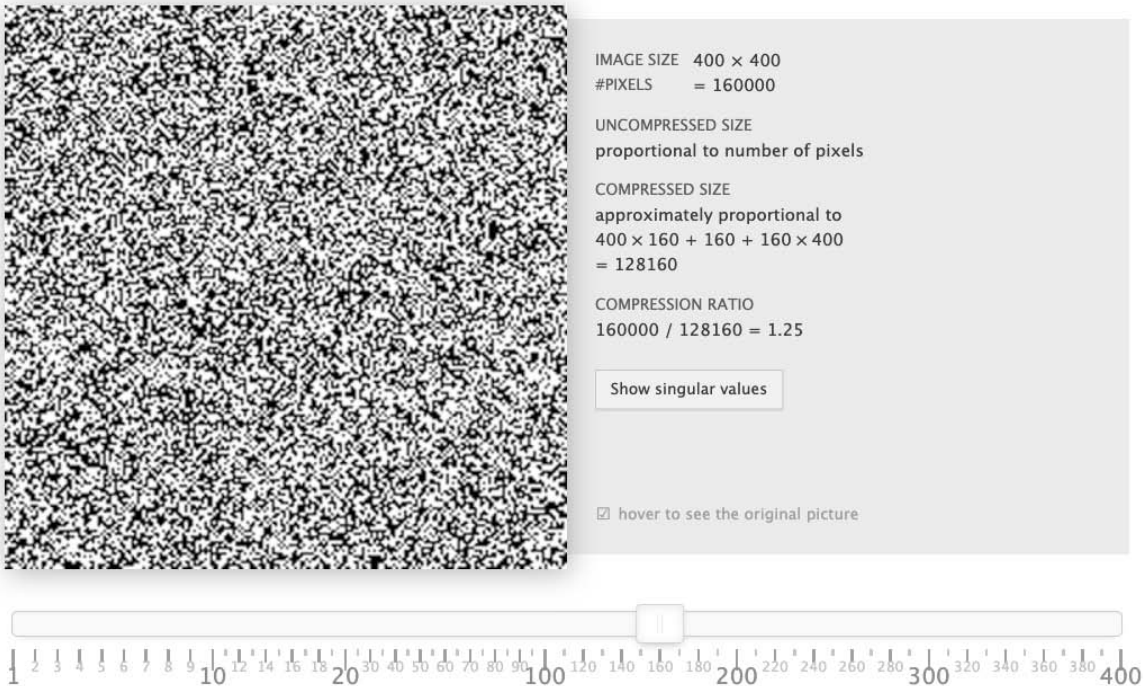


Figure 19b. Image of random binary matrix of size 200 compressed to 160 singular values

For the image in Fig. 19a, a good compression (no visual difference) can be obtained when the image is compressed to 160 singular values. The original image has $400 \times 400 = 160000$ pixels. The image that is compressed to 160 singular values has size $400 \times 160 + 160 + 160 \times 400 = 128160$, decreased from the original by a factor of 1.25. SVD is not really efficient on this image. On other random binary matrices, SVD also works similarly or worse. On a random binary matrix of size 1000, it has to compress to at least 400 to 500 singular values to get a good compression. This means that the U, Σ, and V^T matrices have more entries than the original matrix. After multiple trials, it can be concluded that SVD works ineffectively on random binary matrices.

The image of a random binary matrix simply consists only of random black and white pixels. Random matrices that contain randomly generated entries between 0 and 255 are constructed and analyzed. After converting to an image file, it will show pixels of random colors on the greyscale. First, the singular values are plotted with regular scale and log scale y axis and analyzed.

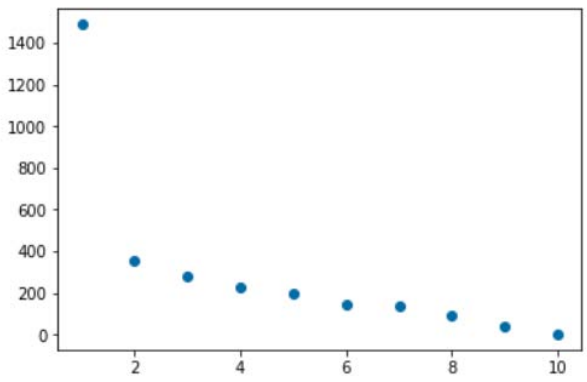


Figure 20a. Random matrix of size 10

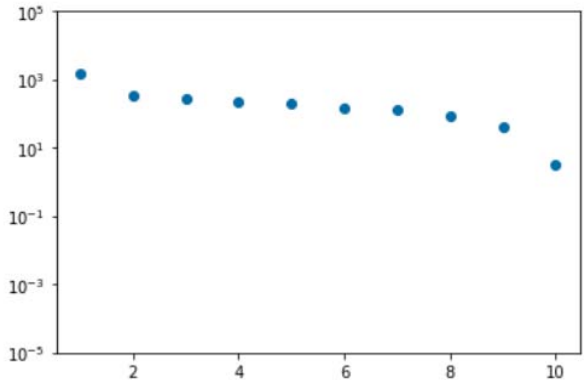


Figure 20b. Random matrix of size 10

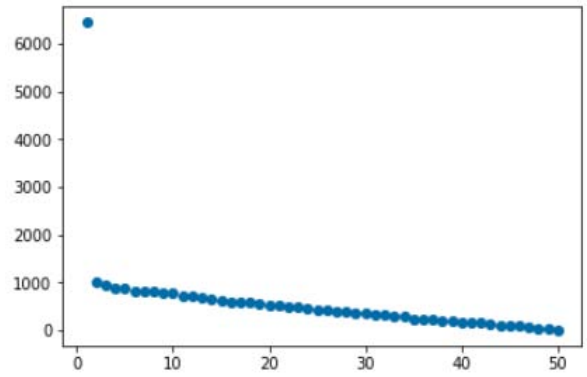


Figure 21a. Random matrix of size 50

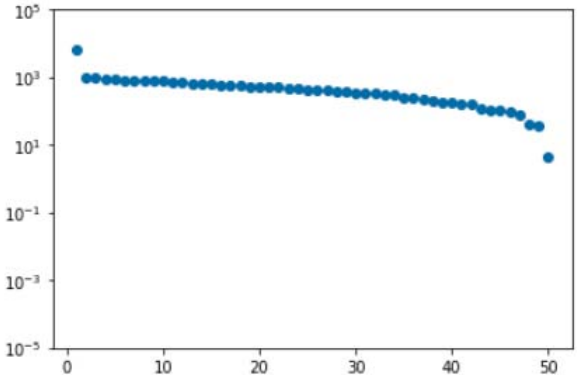


Figure 21b. Random matrix of size 50

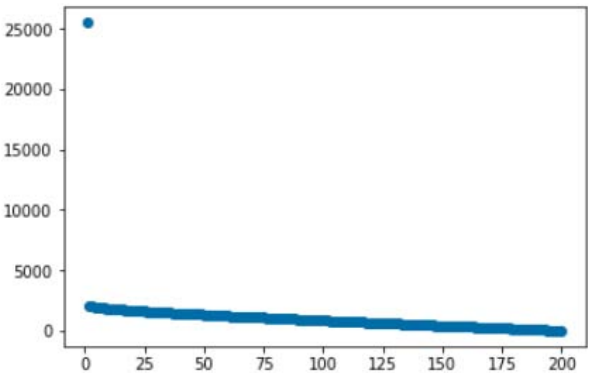


Figure 22a. Random matrix of size 200

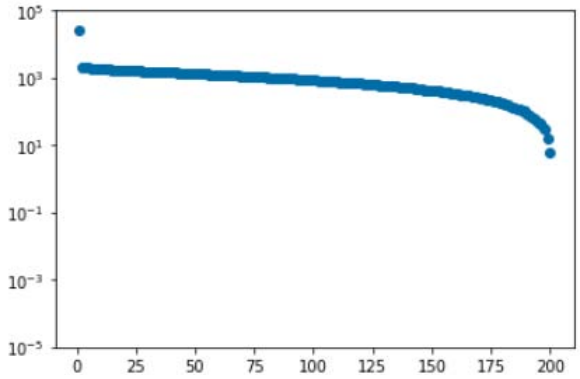


Figure 22b. Random matrix of size 200

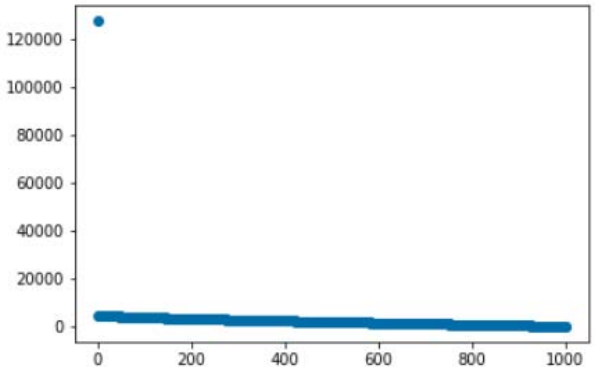


Figure 23a. Random matrix of size 1000

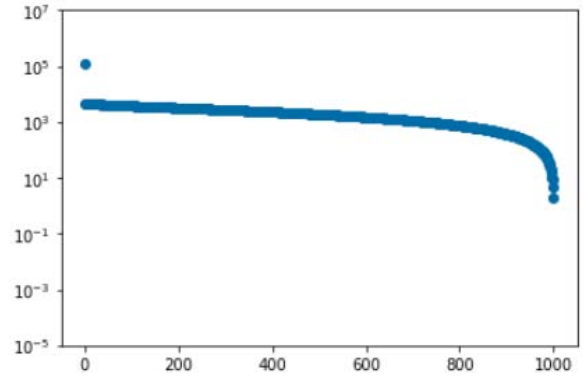


Figure 23b. random matrix of size 1000

Here, the singular values of four random binary square matrices of different sizes are calculated and plotted from Fig. 20 to Fig. 23. Similar to a random binary matrix, as the size of the matrix increases, the descending pattern of the singular values becomes highly consistent (Fig. 22 and Fig. 23). More importantly, the descending pattern looks very similar to that of a random binary matrix. Similar to the procedure on random binary matrices, a random matrix of size 200 is converted to an image and enlarged by a factor of two for better visualization.

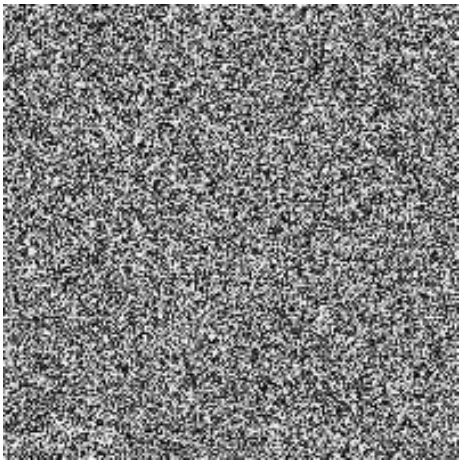


Figure 24a. Image of random matrix of size 200

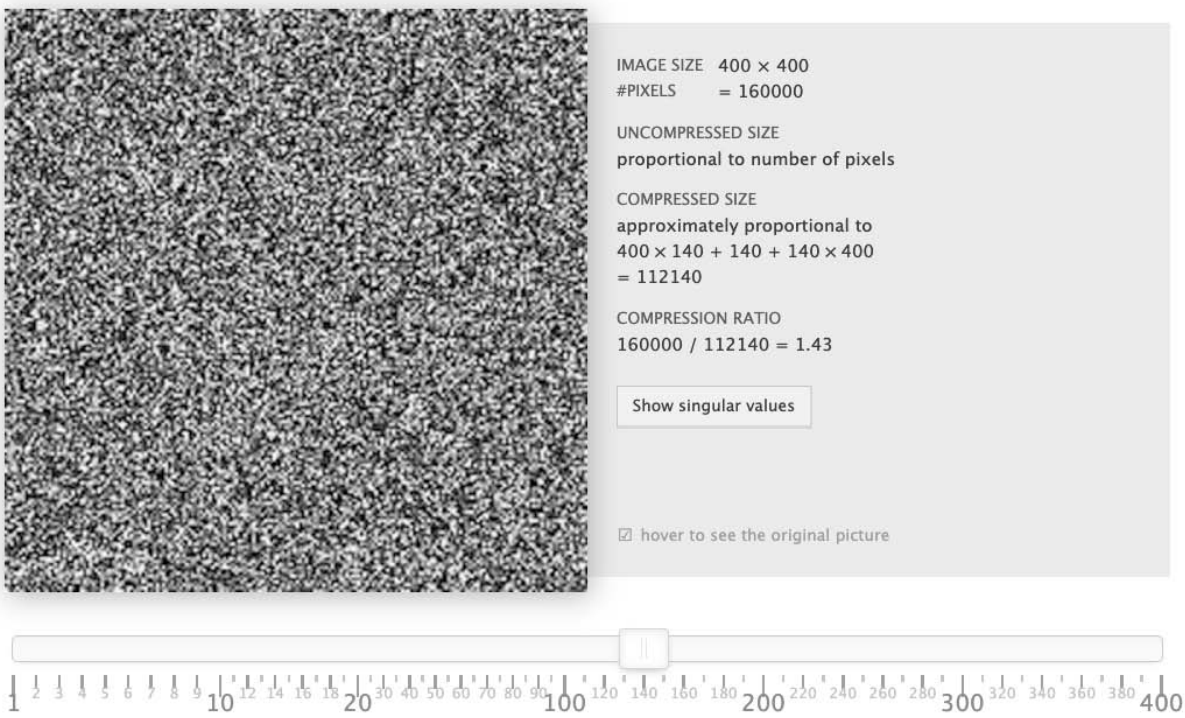


Figure 24b. Image of random matrix of size 200 compressed to 50 singular values

From the experience with a random binary matrix, we anticipate that a high number of singular values has to be selected. For the image in Fig. 24a, a good compression can be obtained when the image is compressed to 140 singular values. In comparison with Fig. 20a, Fig. 24a is slightly more efficient to compress with SVD. Tim Baumann's demo yields similar results for other random matrices. For a random matrix of size 1000, a good compression requires around 400 singular values. After experimenting with images converted from random matrices on the demo, a conclusion can be made that random matrices are inefficient to compress using SVD.

6 SVD on Matrix of Numerical Low-Rank

After examining SVD on all types of matrices, it is important to look back at what SVD is. The key to SVD is that it breaks a matrix into rank-one pieces and puts those rank-one pieces in order of importance. This is the reason why this paper started by discussing rectangular shapes, because the images are low-rank. The vast majority of images are not matrices of low rank, in fact, mostly are full rank. However, some matrices are numerical low-rank.

The definition of numerical rank of a matrix is very similar to the definition of rank. The main difference is when numerical rank of a matrix is defined, tolerance ε is allowed. Tolerance can be interpreted as "wiggle room." The numerical rank of matrix A is defined as $rank_{\varepsilon}(A) = k$ when $\sigma_{k+1}(A) \leq \varepsilon \times \sigma_1(A)$ and $\sigma_k(A) > \varepsilon \times \sigma_1(A)$ [11]. In this section, rumerical low-rank will be investigated. Two of the most well-known numerical low-rank matrices are the Hilbert matrix and the Vandermonde matrix. Below, the singular values of the two matrices are plotted and analyzed.

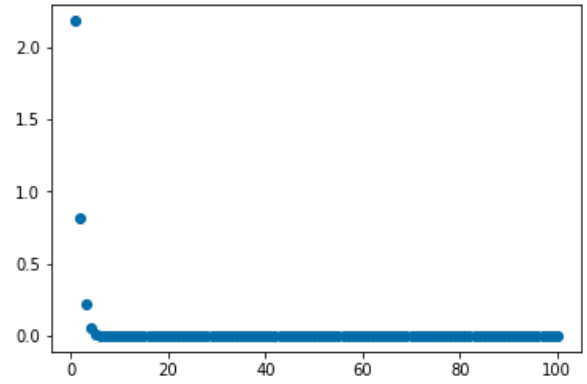


Figure 25a. Hilbert matrix

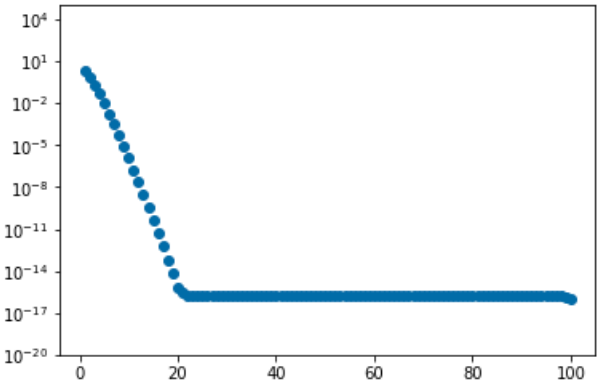


Figure 25b. Hilbert matrix

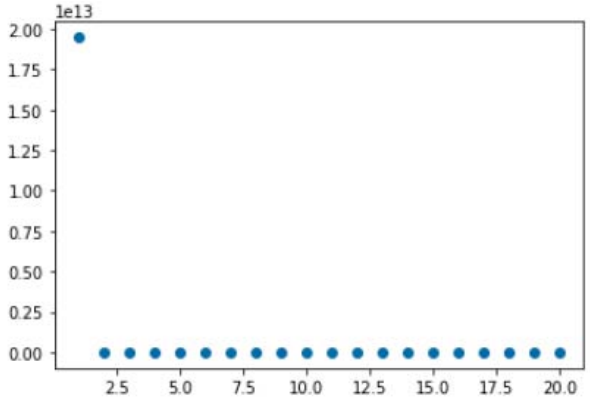


Figure 26a. Vandermonde matrix

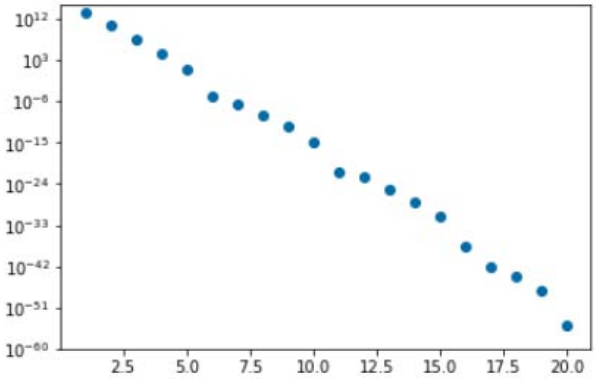


Figure 26b. Vandermonde matrix

A sharp difference can be noticed immediately when examining the singular values graphs. In Fig. 25b and Fig. 26b, the singular values decrease rapidly, unlike any other singular value graph shown in the previous sections. The decreasing pattern singular values of these two matrices almost look linear in the log scale graphs. The previous experience relating singular value graphs with image compression efficacy shown on Tim Baumann's site has shown that the rapidly decreasing singular value can be the

perfect setup for effective singular value decomposition. Eckart-Young states that the singular values can tell us how well the matrix can be approximated with a low-rank matrix. Specifically, $\sigma_{k+1}(A) = \left|\left|x - x_k\right|\right|_2$ [11]. In other words, the Hilbert matrix and the Vandermonde matrix can be approximated with low-rank matrices with a very small tolerance at the same time. Unfortunately, the ways the two matrices are defined limit the visualization and the use of Baumann's demo.

7 SVD on Real-Life Image

Last but not least, the efficacy of applying SVD on real-life images was investigated. On Tim Baumann's page, there are 15 images that he built in. Of those 15 images, five are identified only with question marks and contain images that can only be seen by using the demo. One of them is Vincent van Gogh's S*tarry Night*. The singular values of the image of *Starry Night* are calculated and plotted below:

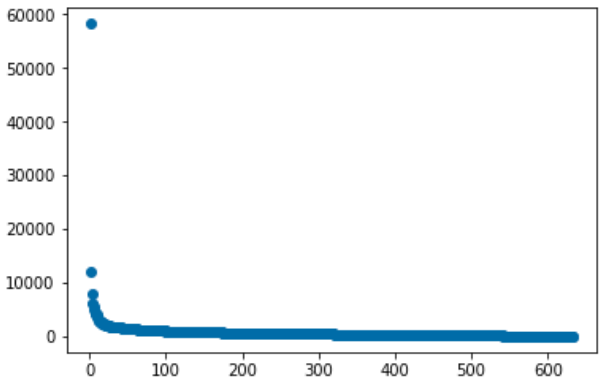


Figure 27a. *Starry Night*

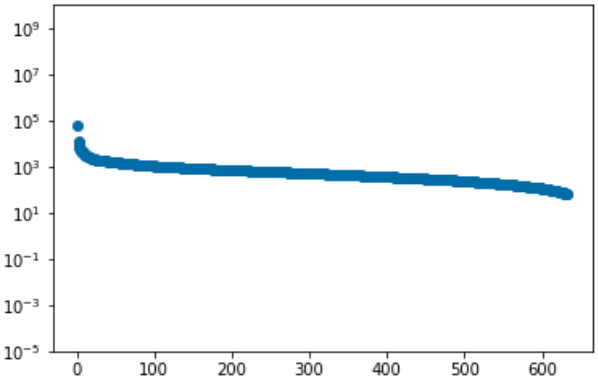


Figure 27b. *Starry Night*

Figure 27c. *Starry Night*

Fig. 27a and Fig. 27b depict the singular values of the *Starry Night* matrix. Fig. 27b shows a pattern of slowly decreasing singular values between 10^5 to 10^1. In comparison with graphs of some of the hexagon and triangular matrices, the singular values of the *Starry Night* matrix decrease at about the same rate. Surprisingly, the singular values seem to diminish at a slightly slower rate compared to that of a random matrix. However, for the random matrices, the singular values begin to decrease faster towards the end, whereas the curve in Fig. 27b remains smooth and flat. The features of the singular value graphs make sense because the painting is so complex and consists mostly of small diagonal strokes, as shown in Fig 27c. Based on Fig. 27a and Fig. 27b, it can be anticipated that the image *Starry Night* is not very effective to compress with SVD.

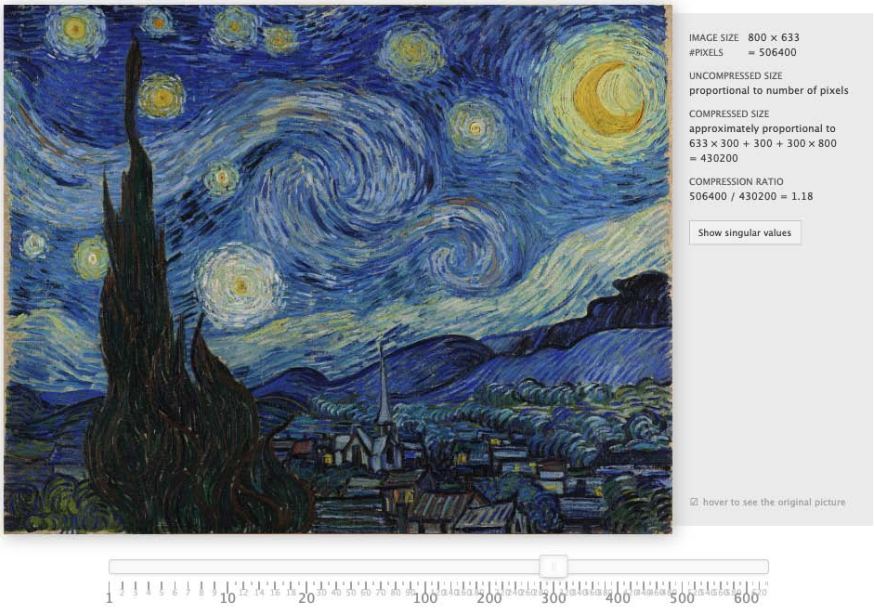


Figure 27d. *Starry Night* compressed to 300 singular values

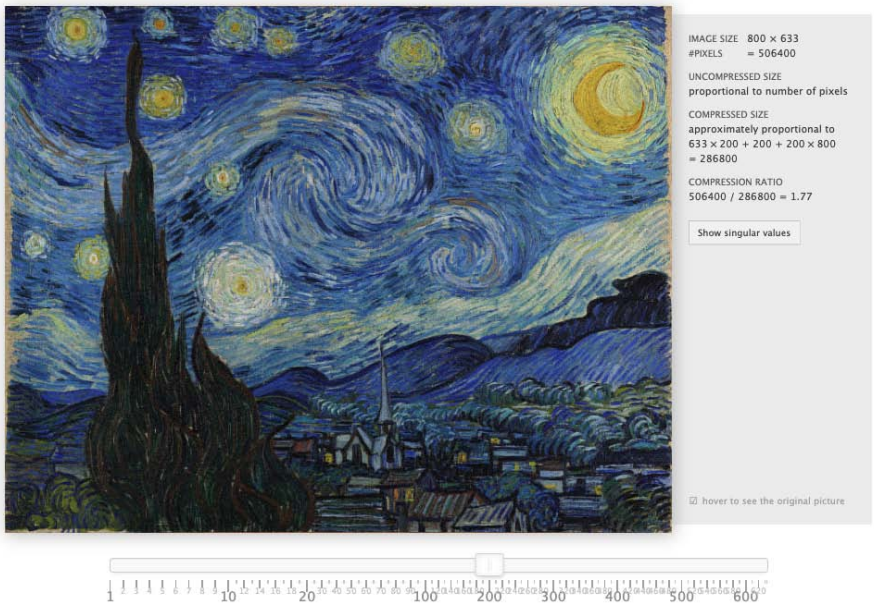


Figure 27e. *Starry Night* compressed to 200 singular values

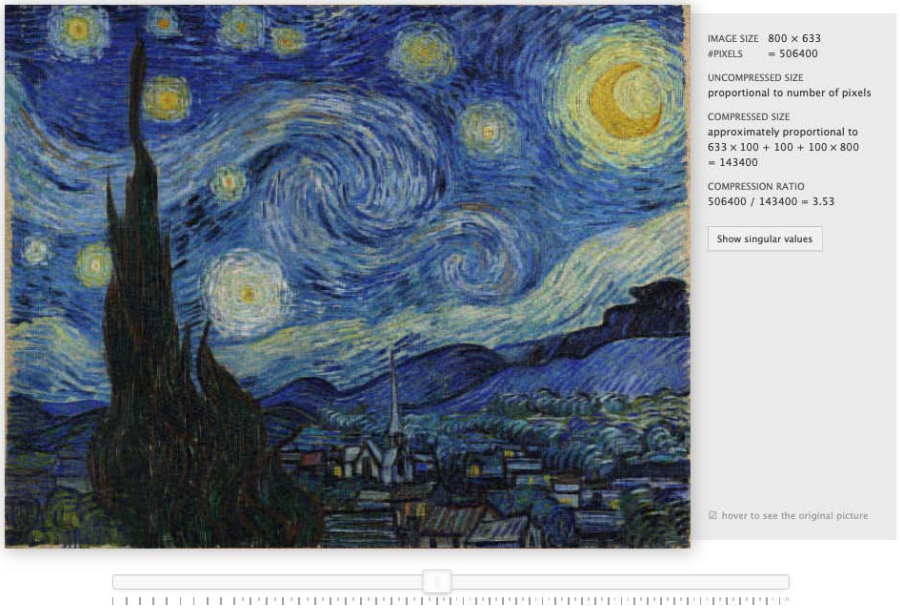


Figure 27f. *Starry Night* compressed to 100 singular values

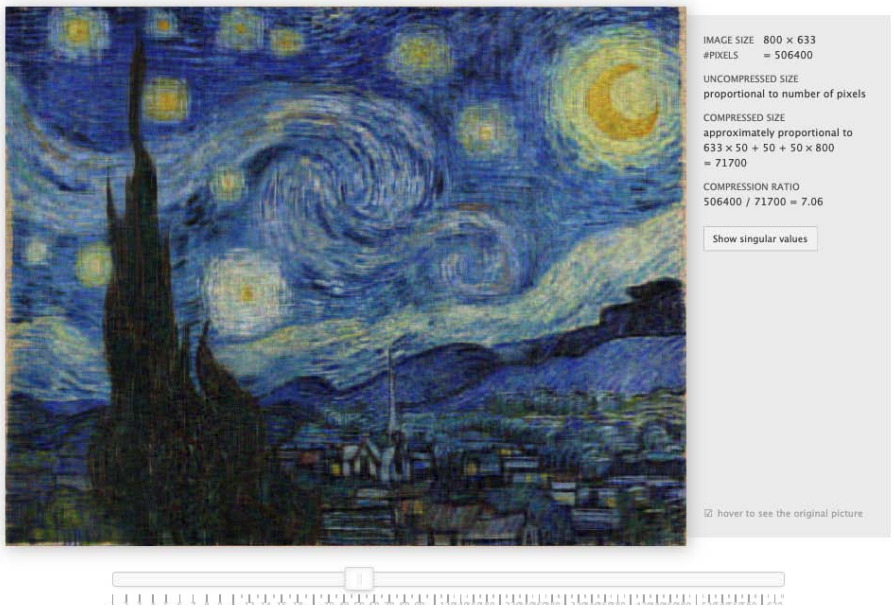


Figure 27g. *Starry Night* compressed to 50 singular values

For the image in Fig. 27c, a good compression (no visual difference) can be obtained when the image is compressed to 300 singular values. The original image has $800 \times 633 = 506400$ pixels. The image that is compressed to 300 singular values has size $800 \times 300 + 300 + 300 \times 633 = 430200$, decreased from the original by a factor of 1.18. SVD is not really efficient on this image. Nevertheless, Fig. 27e and Fig. 27f show how good an approximation can result when fewer number singular values are selected. Although some features of the original image are lost, the overall structure and the most significant details remain. When the image of *Starry Night* is compressed to 200 singular values and 100 singular values, the compression size reduces from the original size by a factor of 1.77 and 3.53. It is really about what quality of the approximation is considered acceptable. For a real-life image, the curve of

the decreasing singular values is mostly flat and smooth. The efficacy of using SVD differs from image to image, but not by a great deal. For example, an image of a Broadway musical advertisement can have a better approximation with lower rank than an image of a rainforest, because it is simpler and often contains English letters with vertical and horizontal edges, sometimes curved edges. However, SVD can never be used as effectively on real-life images as on numerical low-rank matrices, because the singular values of real-life images never decrease rapidly.

8 Conclusion

After investigating the effect of SVD on different matrices and images using Baumann's code, the following observations can be made. While matrices of low-rank and numerical low-rank compress well with SVD, matrices with geometric shapes, such as triangles and hexagons, and random matrices do not. Real-life images cannot be approximated with low-rank matrices with a very small tolerance at the same time. Nevertheless, in our daily lives, there are many big data matrices that turn out to be numerical low-rank [14]. For example, they appear in movie preferences, text documents, survey data, medical records, etc. Since these datasets are often very large, it is important to know when and how they can be approximated with a low-rank matrix. SVD can be a useful tool for those approximations.

9 Reference

[1] Baumann, T. Image Compression with Singular Value Decomposition. SVD-Demo: Image Compression. http://timbaumann.info/svd-image-compression-demo/.

[2] Bermeitinger, B., Hrycej, T., & Handschuh, S. (2019). Singular Value Decomposition and Neural Networks. Lecture Notes in Computer Science Artificial Neural Networks and Machine Learning – ICANN 2019: Deep Learning, 153–164. https://doi.org/10.1007/978-3-030-30484-3_13

[3] Central Intelligence Agency. (2015). References :: Flags of the World. In The world factbook. essay.

[4] Cleary, F. Singular Value Decomposition of an Image¶. SVD Image Compression. https://www.frankcleary.com/svdimage/.

[5] Coja-Oghlan, A., Ergür, A. A., Gao, P., Hetterich, S., & Rolvien, M. (2020). The rank of sparse random matrices. Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, 579–591. https://doi.org/10.1137/1.9781611975994.35

[6] Color Image Compression using Singular Value Decomposition and Back Propagation Neural Network. (2020). Journal of Xidian University, 14(4). https://doi.org/10.37896/jxu14.4/370

[7] Cooper, C., Frieze, A., & Pegden, W. (2019). On the rank of a random binary matrix. Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, 946–955. https://doi.org/10.1137/1.9781611975482.58

[8] Ding, C., & Ye, J. (2005). 2-Dimensional Singular Value Decomposition for 2D Maps and Images. Proceedings of the 2005 SIAM International Conference on Data Mining. https://doi.org/10.1137/1.9781611972757.4

[9] Fingerprint Compression using Singular Value Decomposition. (2016). International Journal of Science and Research (IJSR), 5(4), 884–887. https://doi.org/10.21275/v5i4.nov161874

[10] Image Compression via the Singular Value Decomposition. (2007). Wolfram Demonstrations Project. https://doi.org/10.3840/000401

[11] Massachusetts Institute of Technology. (2018). Lecture 17: Rapidly Decreasing Singular Values. MIT OpenCourseWare. https://ocw.mit.edu/courses/mathematics/18-065-matrix-methods-in-data-analysis-

signal-processing-and-machine-learning-spring-2018/video-lectures/lecture-17-rapidly-decreasing-singular-values/.
[12] Strang, G. (2020). Linear Algebra for Everyone. Wellesley-Cambridge Press.
[13] Townsend, A., & Wilbert, H. (2018). On the singular values of matrices with high displacement rank. Cornell Department of Mathematics . http://pi.math.cornell.edu/~ajt/publications.html.
[14] Udell, M., & Townsend, A. (2019). Why Are Big Data Matrices Approximately Low Rank? SIAM Journal on Mathematics of Data Science, 1(1), 144–160. https://doi.org/10.1137/18m1183480
[15] Virmani, J., Kumar, V., Kalra, N., & Khandelwal, N. (2013). SVM-based characterisation of liver cirrhosis by singular value decomposition of GLCM matrix. International Journal of Artificial Intelligence and Soft Computing, 3(3), 276. https://doi.org/10.1504/ijaisc.2013.053407
[16] Yao, Q., Kwok, J. T., & Zhong, W. (2015). Fast Low-Rank Matrix Learning with Nonconvex Regularization. 2015 IEEE International Conference on Data Mining. https://doi.org/10.1109/icdm.2015.9
[17] Yu, Y., Hong, M., Liu, F., Wang, H., & Crozier, S. (2011). Compressed sensing MRI using Singular Value Decomposition based sparsity basis. 2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society. https://doi.org/10.1109/iembs.2011.6091419