Ryan Pearson

# Singular Value Decomposition and Discrete Cosine Transform of Images

## Introduction:

Digital cameras have revolutionized pictures. Now if you want to take a photo you can pull out your phone and take a photo and instantly see it and have it stored. This is a huge shift from the days of film and having to send in the film to be developed and getting it weeks after you took the photo. The storage of photos digitally is pretty costly though. Each photo is made up of a matrix storing numbers 0-255 for each color red green and blue. Meaning an 8-megapixel photo has 8 million pixels or 24 million values to store. This is why images are usually compressed to reduce the amount of storage it takes to hold the image and for easier transmitting of images.

## Singular value decomposition method:

Singular value decomposition (SVD) is the idea that matrix A can be turned into 3 matrices that which are commonly labeled as U, $\Sigma$ and $V^T$. If A has m rows and n columns then U is an m by m matrix, $\Sigma$ is an m by n matrix and $V^T$ is an n by n matrix. At first, it seems counterproductive to divide an image up into 3 separate matrices since that's roughly 3 times the space to be taken up from storing the original image. However, if only some of the rows and columns are used when multiplying the matrix back out you can still get a good approximation of what A originally was.

$$\begin{aligned}(mxm)*(mxn)*(nxn) &= (mxn)*(nxn)\\ &= mxn\end{aligned}$$

Now let's say H is an arbitrary number less than n and m

$$\begin{aligned}(mxH)*(HxH)*(Hxn) &= (mxH)*(Hxn)\\ &=(mxn)\end{aligned}$$

So now you can see that if we reduced the size of these matrices U, $\Sigma$, and $V^T$ after multiplying them together you'll end up with a matrix of the same size. For image compression, this means there won't be any loss in pixels after it has been compressed down into these matrixes. However, SVD does not guarantee that the resulting A will have the same pixel values, but generally they will be close.

Before I go into why Multiplying smaller matrixes from the SVD together give a good approximation I need to explain how to get the SVD. First, you find $A^T * A$ and its eigenvalues and eigenvectors.

A = $\begin{bmatrix} 2 & 4 \\ 1 & 2 \end{bmatrix}$

$A^T$ * A = $\begin{bmatrix} 2 & 1 \\ 4 & 2 \end{bmatrix} \begin{bmatrix} 2 & 4 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} 5 & 10 \\ 10 & 20 \end{bmatrix}$

To find the eigenvalues solve determinant of ($A^T$ * A – ƛ*I) set equal to 0

$\begin{bmatrix} 5 & 10 \\ 10 & 20 \end{bmatrix} - ƛ\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = 0$

$\begin{bmatrix} 5 - ƛ & 10 \\ 10 & 20 - ƛ \end{bmatrix} = 0$

Then take the Determinate

$\begin{vmatrix} 5 - ƛ & 10 \\ 10 & 20 - ƛ \end{vmatrix} = 0$

(5- ƛ)(20 – ƛ) - 100 =0

$ƛ_1 = 25$
$ƛ_2 = 0$

Then we need to take the square root of the eigenvalues to get the singular values. We'll call them $\sigma$. These values will be the diagonal of $\Sigma$.

$\sigma_1 = 5$
$\sigma_2 = 0$

Next find the nullspace of ($A^T$ *A – ƛ*I) for each eigenvalue found to find the eigenvectors then normalize the vectors. Call these vectors V.

$\begin{bmatrix} 5 - 25 & 10 \\ 10 & 20 - 25 \end{bmatrix} V_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

$x1 = 1/2t,\ x2 = t$

$V_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$

$\begin{bmatrix} 5 - 0 & 10 \\ 10 & 20 - 0 \end{bmatrix} V2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

$x1 = -2t,\ x2 = t$

V$_2$ = $\begin{bmatrix} -2 \\ 1 \end{bmatrix}$

Now Normalize these Vectors

V$_1$ = $\frac{1}{\sqrt{5}}\begin{bmatrix} 1 \\ 2 \end{bmatrix}$ V$_2$ = $\frac{1}{\sqrt{5}}\begin{bmatrix} -2 \\ 1 \end{bmatrix}$

These are the vectors that will make up the matrix V

Now we need another set of vectors U to complete the singular value decomposition. This is found the same way as finding $\Sigma$ and V but by using A* A$^T$

A * A$^T$ = $\begin{bmatrix} 2 & 4 \\ 1 & 2 \end{bmatrix}\begin{bmatrix} 2 & 1 \\ 4 & 2 \end{bmatrix}$ = $\begin{bmatrix} 20 & 10 \\ 10 & 5 \end{bmatrix}$

The eigenvalues end up being the same as above so we must solve for the eigenvectors.

$\begin{bmatrix} 20-25 & 10 \\ 10 & 5-25 \end{bmatrix}$ U$_1$ = $\begin{bmatrix} 0 \\ 0 \end{bmatrix}$

$x1 = 2t,\ x2 = t$

U$_1$ = $\begin{bmatrix} 2 \\ 1 \end{bmatrix}$

$\begin{bmatrix} 20-0 & 10 \\ 10 & 5-0 \end{bmatrix}$ U$_2$ = $\begin{bmatrix} 0 \\ 0 \end{bmatrix}$

$x1 = 2t,\ x2 = t$

U$_2$ = $\begin{bmatrix} -1 \\ 2 \end{bmatrix}$

Now we have all the numbers to fill in matrixes U $\Sigma$ and V$^T$. $\Sigma$ is constructed by taking all the $\sigma$ and putting them in the diagonal of a matrix in decreasing order with size m by n and then filling the rest in with zeros. U is formed by taking the spanning the orthonormal U vectors and then V matrix can be found by spanning the orthonormal V vectors.

U = $\frac{1}{\sqrt{5}}\begin{bmatrix} 2 & -1 \\ 1 & 2 \end{bmatrix}$, $\Sigma$ = $\begin{bmatrix} 5 & 0 \\ 0 & 0 \end{bmatrix}$ V = $\frac{1}{\sqrt{5}}\begin{bmatrix} 1 & -2 \\ 2 & 1 \end{bmatrix}$

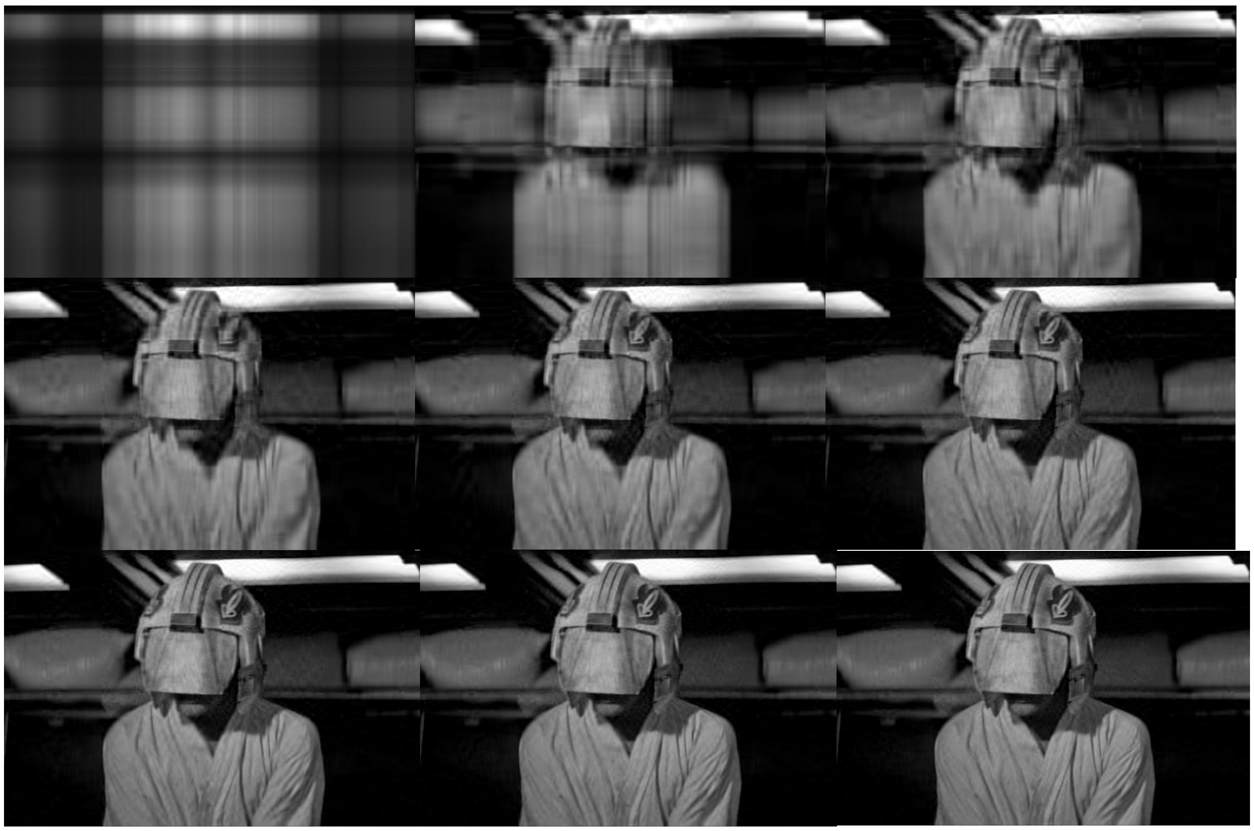
V then must be transposed to V$^T$

$V^T = \frac{1}{\sqrt{5}}\begin{bmatrix} 1 & 2 \\ -2 & 1 \end{bmatrix}$

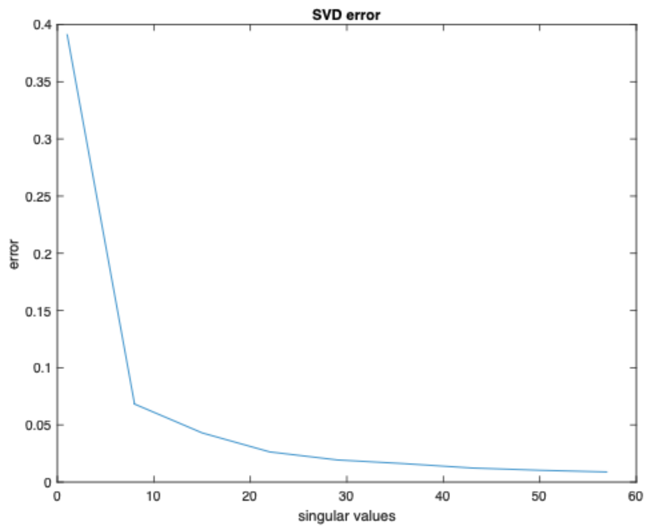
Now multiplying this back out will give you the original A. We can ignore small singular values and the corresponding vectors in $V^T$ and U these new matrixes and end up with a good approximation of A. In this example it is easy to see that using the second singular value of 0 and its vectors would be pointless since you will end up with the zero vector anyways so it can be thrown away. This is the real reason it can be used as image compression to turn one matrix into 3.

Using a program to do the SVD then selectively ignoring the smaller singular values in the new matrixes we can see how a good approximation is given using this 548 by 825 pixel image below.

Now if we compress it using SVD and reconstruct it by only using some of the singular values we can see a good approximation


Number of Singular values used and compression ratio in order of left to right top to bottom (1 : 0.00304, 8 : 0.02436, 15 : 0.04568, 22 : 0.06700, 29 : 0.08832, 36 : 0.10965, 43 : 0.13097, 50 : 0.15229, 57 : 0.17361)

The first image is using just 1 singular value and doesn't look like much of anything but using just 8 singular values like in the second and it begins to take shape. Using 43 singular values the image is almost identical to the original and is much smaller.



This shows the error between the original photo vs. the number of singular values used. Even at 8 singular values this error is relatively low, but the image is hard to made out. At around 29 singular vales the error stops decreasing sharply and is about where the photo starts to look very similar to the original even though it is using just 8.832% of the space that the original is. At 43 singular values the image is nearly identical to the original both in terms of looks and the error.

Discrete Cosine Transform method:

Discrete Cosine Transform (DCT) is the method of compression that JPEG file format uses to compress photos. At a very abstract level it will transform the image into a signal that the high frequency values of can be considered unimportant since the coefficients are generally so small and then can be thrown away. The general way to compress and image is to divide it into 8 by 8 matrixes and then apply DCT to the chunks of the image. The spatial equation (F(i,j)) to turn each chunk into a signal equation (F(u,v)) is show here:

$$F(u,v) = \left(\frac{2}{N}\right)^{\left(\frac{1}{2}\right)} * \left(\frac{2}{M}\right)^{\left(\frac{1}{2}\right)} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} A(i)A(j) \cos\left(\frac{\pi * u}{2N}(2i+1)\right) \cos\left(\frac{\pi * v}{2M}(2j+1)\right) F(i,j)$$

$$A(x)=\begin{Bmatrix} \frac{1}{\sqrt{2}}\ for\ x = 0 \\ 1\ for\ x\ \neq 0 \end{Bmatrix}$$

A 1-dimensional version of this equation would be

$$F(u) = \left(\frac{2}{N}\right)^{\left(\frac{1}{2}\right)} \sum_{i=0}^{N-1} A(i) \cos\left(\frac{\pi * u}{2N}(2i+1)\right) F(i)$$

$$A(x)=\begin{Bmatrix} \frac{1}{\sqrt{2}}\ for\ x = 0 \\ 1\ for\ x\ \neq 0 \end{Bmatrix}$$

Now to show and example I will use the simpler 1-dimensional version. Say we have a Matrix that is 1 by 6 like this

F(x) =$[84 \quad 6 \quad 20 \quad 33 \quad 68 \quad 48 \quad 15 \quad 12]$

After we do DCT to these values we get

F(u) = $[101.1 \quad 20.3 \quad -11.3 \quad 54.7 \quad 38.2 \quad 7.1 \quad 20.8 \quad 15.0]$

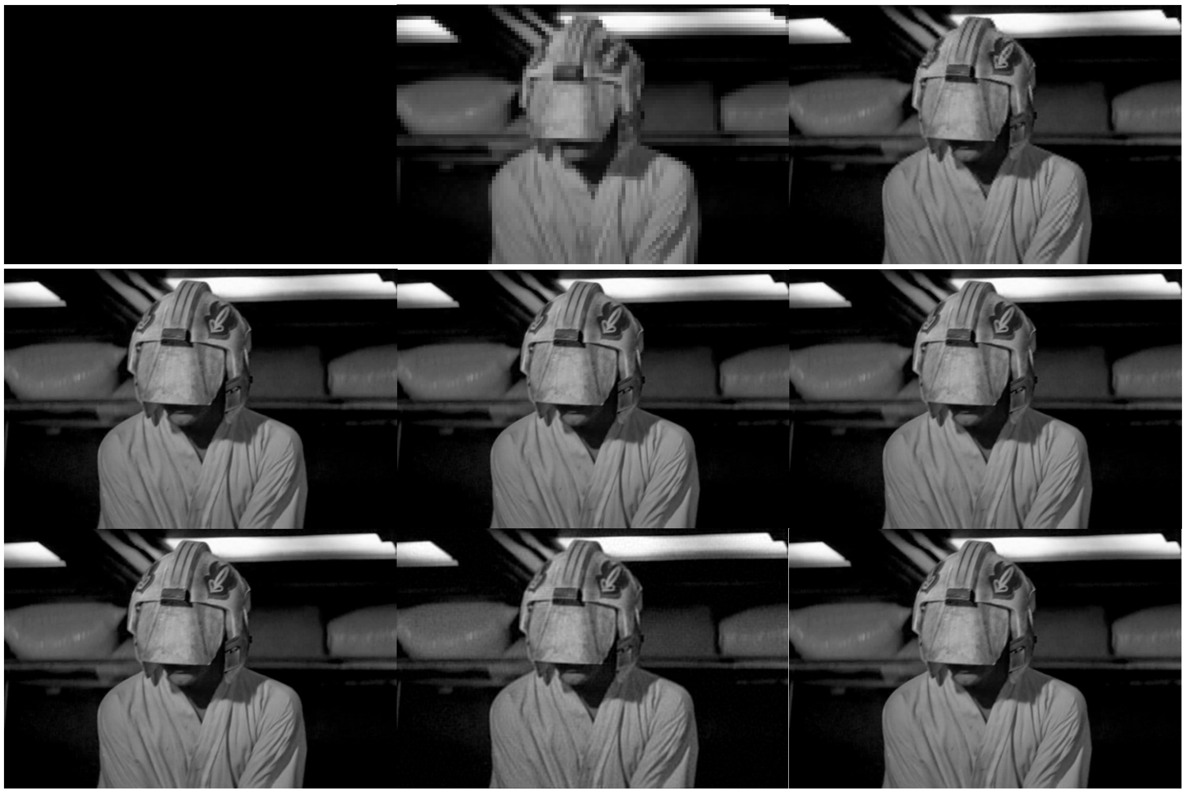
Then the numbers at the end can be ignored and the inverse DCT can be applied to this set to give you an approximation of the original F(x). In this case only the first 4 numbers were used to compute the inverse DCT and the rest were set to zero which is a pretty extreme example, but it still manages to get pretty close for only having half of the data.

$$F(x) \approx [63.1 \quad 36.7 \quad 16.9 \quad 27.9 \quad 54.1 \quad 58.8 \quad 30.4 \quad -2.0]$$

For a 2-dimensional version of this like how a photograph would be compressed the high frequency numbers won't be at the end of the list since DCT will produce a coefficient matrix instead. The numbers that can be selectively ignored will still be the high frequency values, but these are now in the bottom right of the matrix.

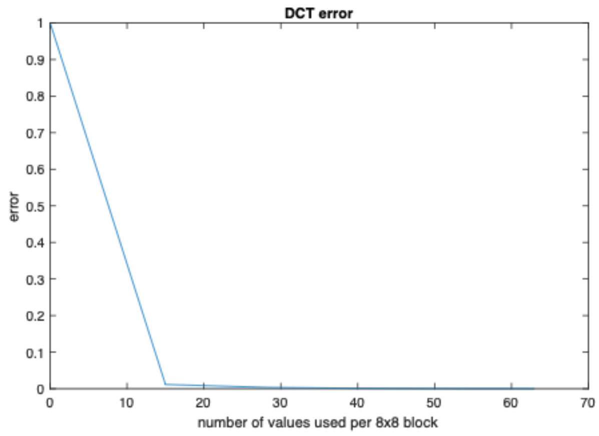
Using a program to compute the DCT of an image and then reconstructing it and ignoring the high frequency values we can see how DCT can compress an image.

This is the uncompressed Image for reference to the compressed images below.



Number of values in each 8 by 8 block and compression ratio in order of left to right top to bottom (0 : 0.0, 1 : 0.016, 15 : 0.234, 28 : 0.438, 39: 0.609, 48 : 0.75, 55 : 0.860, 60 : 0.937, 63 : 0.984)

The top row of photos are the only ones the show any degradation and using only the top 15 lowest frequency values it is almost impossible to tell the difference. All the rest of the photos look almost identical even though they use a faction of data the original image uses. JPEG images end up being compressed more by doing Huffman compression (a form of lossless compression) on the DCT Matrix to further reduce the size of the image.



The error for the DCT compressed photos is much steeper. Until the value of 15 per 8 by 8 block where is it nearly 0 and this image cannot be distinguished from the original. Any values past using 15 per 8 by 8 block have a unnoticeable error between the original even though they use a faction of the space the original does.

Conclusion:

Both of these compression algorithms were done on black and white images since it was easier to compress just one matrix of black and white than 3 matrices of red, green, and blue. For a colored image the SVD or DCT would be applied to the colors red, green, and blue separately and then the matrices will be “stacked” on top of each other.

These algorithms are both very useful to compress photos to a very high level but since they can compress and image so much the resulting image may not look like the original. The main problem with these is finding a balance between reducing the number of bits it takes to store the image and the quality of the resulting compressed image.

References:

Strang, Gilbert. “Singular Value Decomposition (the SVD).” YouTube, MIT OpenCourseWare, 6 May 2016, www.youtube.com/watch?v=mBcLRGuAFUk.

The Singular Value Decomposition (SVD). MIT, math.mit.edu/classes/18.095/2016IAP/lec2/SVD_Notes.pdf.

Marshall, Dave. *The Discrete Cosine Transform (DCT)*. 4 Oct. 2001, http://users.cs.cf.ac.uk/Dave.Marshall/Multimedia/node231.html.