An Exploration of Transformations into Different Color Spaces and Their Implications for Image Processing and Compression

Image compression is an integral part of computer science, and has been so as long as the need has existed to transmit detailed images while facing the challenges of infrastructure limits, restricting means of transmission, and the need for efficiency in storage. A simple model for image compression consists of constructing a singular value decomposition (SVD) of the image, which can then be altered in such a way as to reduce the amount of data that must be transmitted, while retaining important features.

A singular value decomposition of an *m x n* matrix *A* is represented by the equation

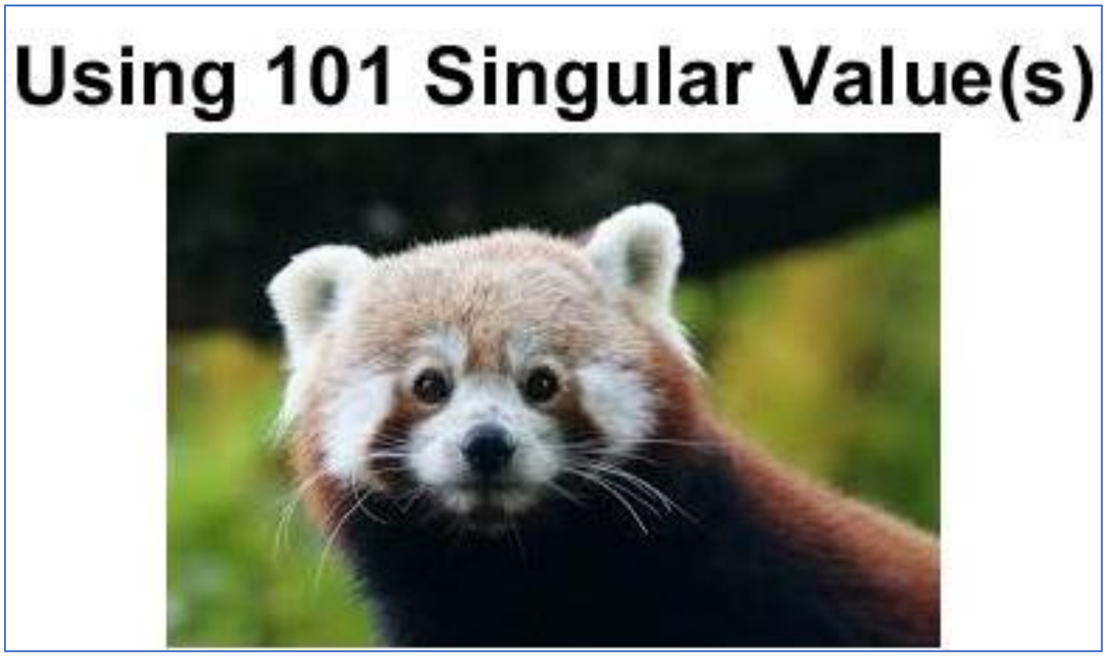
$$A = U\Sigma V^T$$

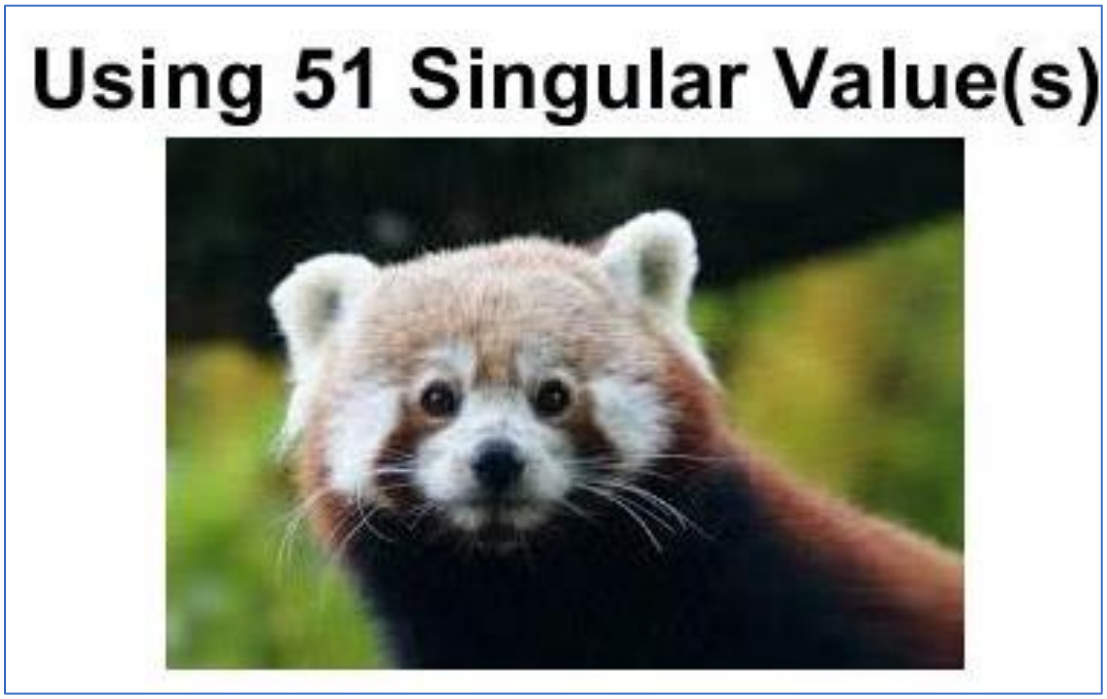
where Σ is an *m x n* diagonal matrix with the entries being the sorted eigenvalues of AA^T, largest to smallest, the columns of V are the unit eigenvectors associated with the singular values, and each column of *U* $\mathbf{u}_i = \frac{A\boldsymbol{v}_i}{\sigma_i}$, with any further needed columns of *U* expanded as part of an orthonormal basis.

Basic compression replaces all but the first *k* selected singular values with zeros. Because the singular values are sorted, the result of this is a matrix multiplication $U\Sigma' T$ which, if represented as a sum, contains only the *k* most influential terms, and is therefore easier and more efficient to transmit. This process is fairly easy to perform on greyscale images, and can be extended without too much difficulty to color images.

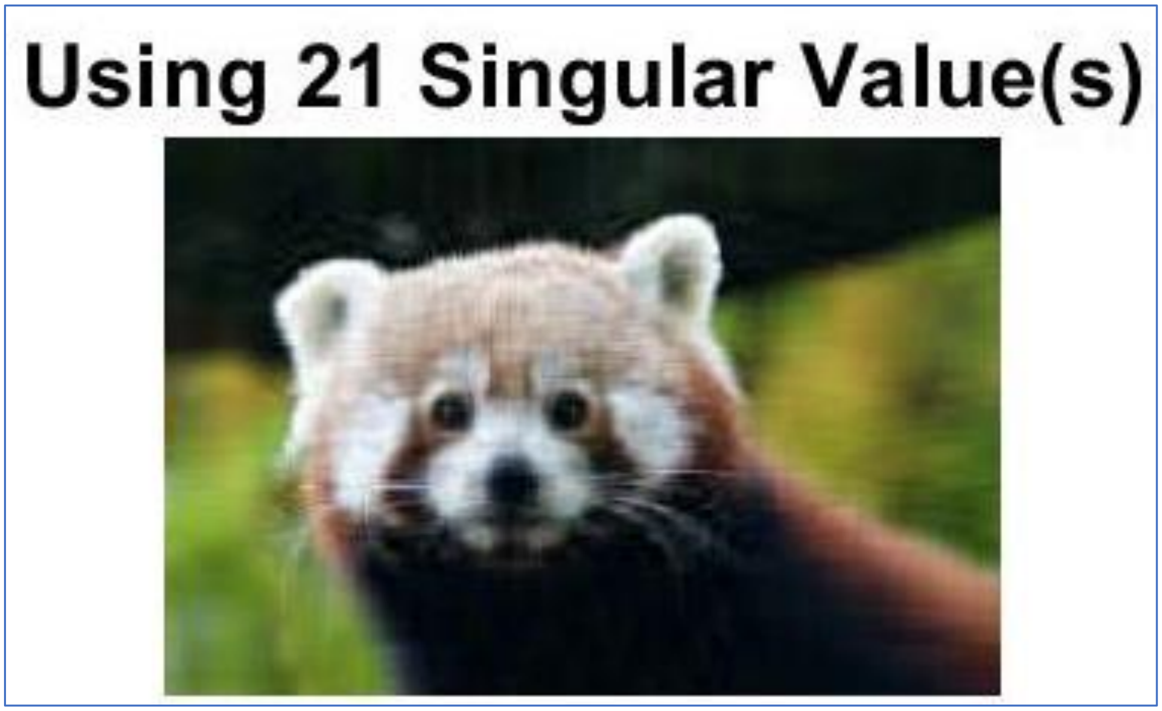
Above is an image of a red panda (*Ailurus fulgens*). On screen, the image is represented as the combination of the R, G, and B matrices associated with the photo, each value of which represents, on a scale of 0 to 255, the proportion of red, green, or blue that the associated pixel contains.

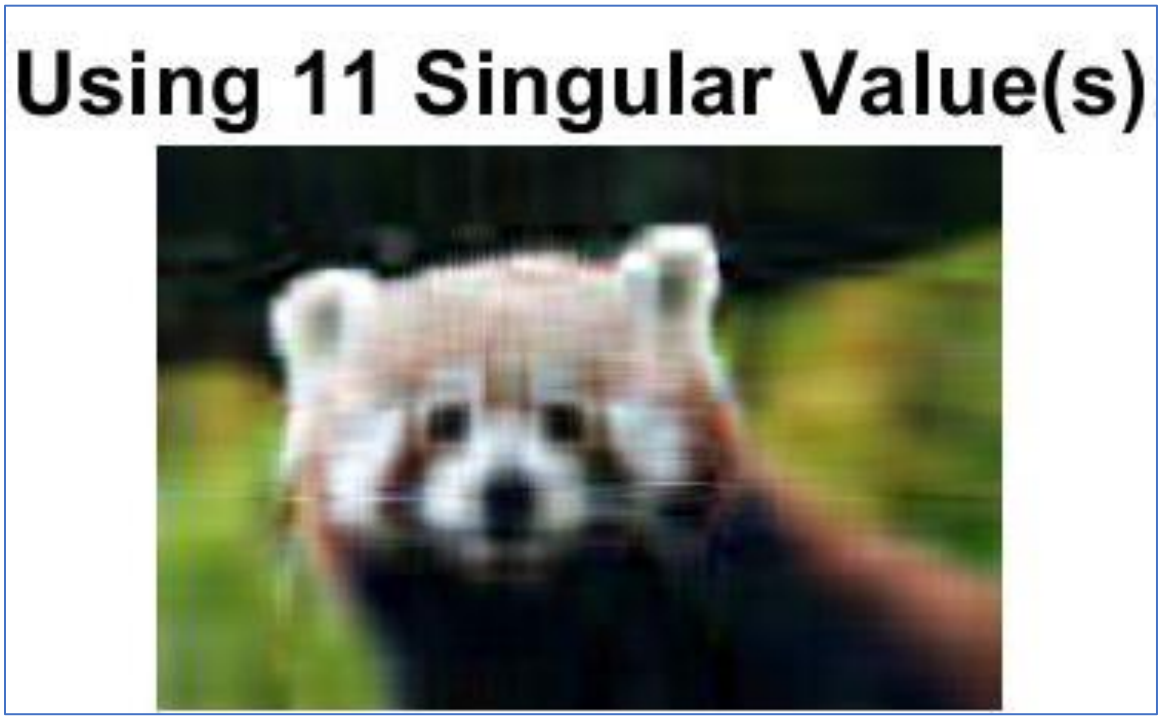
The image was compressed by SVD using code derived from Stack Overflow comments[1], modified to compress the R,G,and B matrices separately. Selected results appear below, with the number of retained singular values accompanying the compressed images.





[1] https://stackoverflow.com/questions/13614886/using-svd-to-compress-an-image-in-matlab





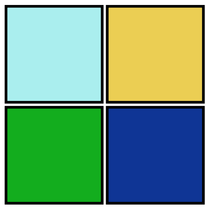
The immediate problem with this method of compression is that SVD compression is fairly inflexible: each matrix involved is degraded by the same amount, and the process of compression doesn't allow for much finetuning or identification of which elements can be discarded without significantly impacting the resulting image.

In fact, the RGB color space itself is not a good tool for this. Each color value is necessary for the image to look correct; an attempt to simplify only one or two of the matrices is immediately noticeable and counterproductive. To identify elements that can be removed without significant impact to the visual appearance of the image, standard compression algorithms transform the RGB color space into the Y′CbCr color space, where Y′ represents the luma or greyscale information, and Cb and Cr are the red and blue chrominance information[2].

Human perception is less sensitive to changes in Cb and Cr, and more sensitive to changes in Y′, so a typically compressed image will, for every 4 pixels of information retained in the Y′ matrix, only retain 2 pixels' worth of information in the Cb and Cr matrices, and then adjust sizes to match. The image is then converted back into RGB, and the compression is complete. This method is efficient enough that it forms the basis of jpeg compression.

The discussion above is fairly abstract, so we illustrate with an example. Suppose we have a 2x2 pixel image (approximation below) with color matrices:

$$R = \begin{bmatrix} 170 & 235 \\ 19 & 15 \end{bmatrix}, \qquad G = \begin{bmatrix} 238 & 206 \\ 173 & 53 \end{bmatrix}, \qquad B = \begin{bmatrix} 238 & 83 \\ 30 & 149 \end{bmatrix}$$



The transformation from RGB to Y′CbCr is accomplished with the following matrix transformation[3], with R, G, and B being the values for each pixel, *not* the entire matrices:

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Ranges:
R/G/B [0 ... 255]
Y/Cb/Cr [0 ... 255]

www.equasys.de

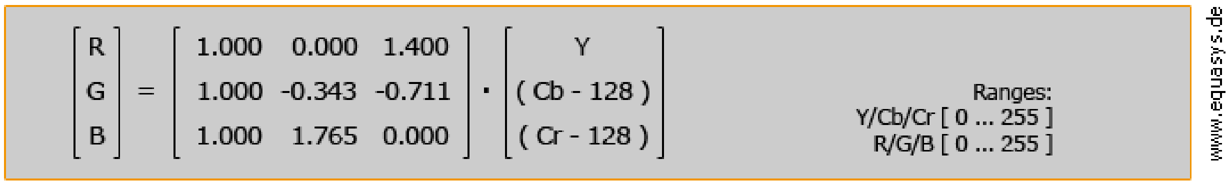
RGB to full-range YCbCr color conversion

After applying these equations, the image has color matrices:

$$Y' = \begin{bmatrix} 217.668 & 200.649 \\ 110.652 & 52.82 \end{bmatrix}, \qquad Cb = \begin{bmatrix} 139.492 & 61.599 \\ 82.526 & 182.422 \end{bmatrix}, \qquad Cr = \begin{bmatrix} 94 & 152.463 \\ 62.583 & 101.224 \end{bmatrix}$$

It's important to note that this is the exact same image. It is even still displayed with RGB values because of hardware. The only change is the way we discuss the colors of the image. Image compression software would manipulate these values to produce a result that, while smaller in size, would be fairly similar to human eyes (or at least, this would be the case if our example were larger than 2x2). The transformation back to RGB is produced by the following equation[3]:

[2] https://www.ma.utexas.edu/users/davis/reu/ch1/signals/chroma.pdf
[3] http://www.equasys.de/colorconversion.html



$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.000 & 0.000 & 1.400 \\ 1.000 & -0.343 & -0.711 \\ 1.000 & 1.765 & 0.000 \end{bmatrix} \cdot \begin{bmatrix} Y \\ (Cb - 128) \\ (Cr - 128) \end{bmatrix}$$

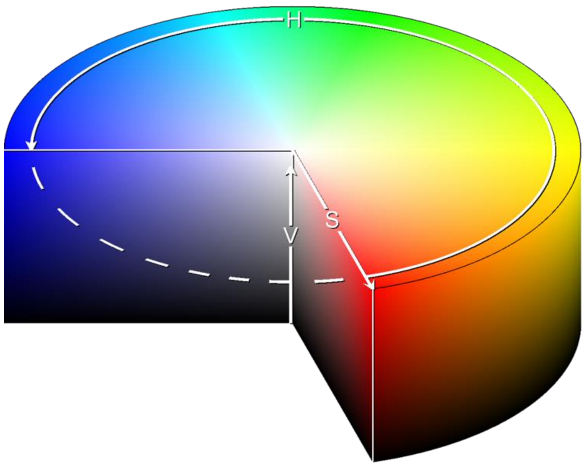
Ranges:
Y/Cb/Cr [0 ... 255]
R/G/B [0 ... 255]

www.equasys.de

Full-range YCbCr to RGB color conversion

Some rounding error will likely be observed.

As a side note, Y′CbCr isn't the only color system that tries to align with visual perception. A representation of the Hue, Saturation, Value (HSV) space is reproduced below.[4]



While similar in concept, Y′CbCr is less computationally complex than HSV, and is less prone to errors and confusion which result from the cylindrical shape of the HSV space. Nonetheless, HSV is useful in specific applications (an example: use of HSV to measure how "green" leaves are). The math for transforming RGB to HSV is complex enough that the reader is encouraged to find it themselves. In brief, it involves a transformation RGB→YIQ, followed by a hue rotation transform and other calculations.[5]

To sum up, while greyscale and RGB color spaces are useful for understanding image processing and can be used for rudimentary compressions through SVD, modern image compression relies on transformations into color spaces which attempt to mimic visual perception, and then exploits the qualities of these spaces in order to destroy information that carries little perceptual significance. The images are then translated back into RGB space for display, having undergone the compression process.

[4] (3ucky(3alls/CC-BY-SA-3.0, https://upload.wikimedia.org/wikipedia/commons/e/e0/HSV_cylinder.png

[5] https://beesbuzz.biz/code/hsv_color_transforms.php