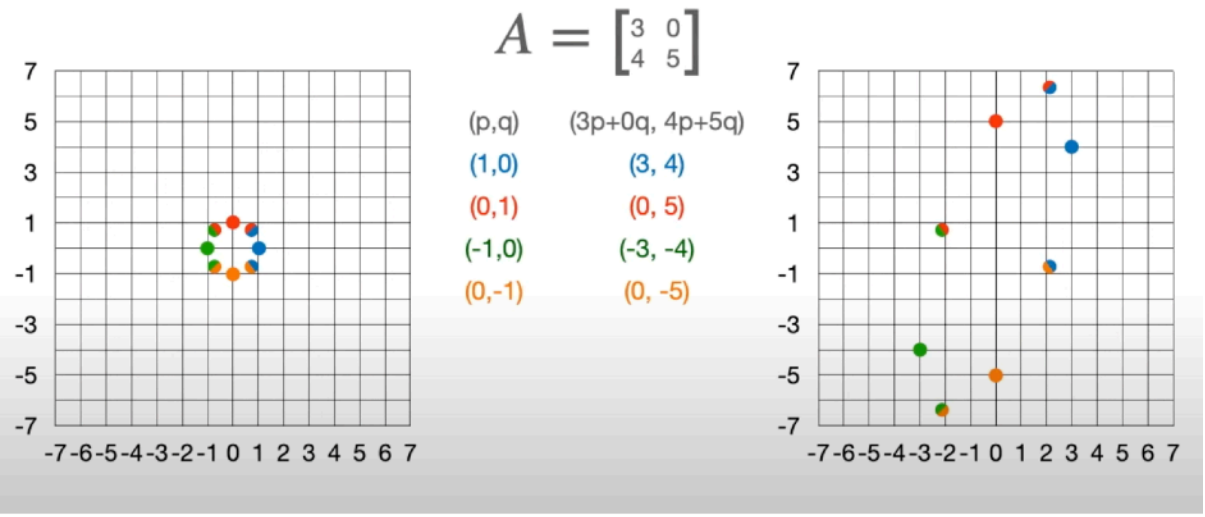
IMAGE COMPRESSION USING SVD

Aditya Chatterjee

Linear Transformations

To understand SVD, let us first look at what exactly is a Linear Transformation and the effect it has on a graph. Lets take 4 points {(1,0), (0,1), (-1,0), (0,-1)} on



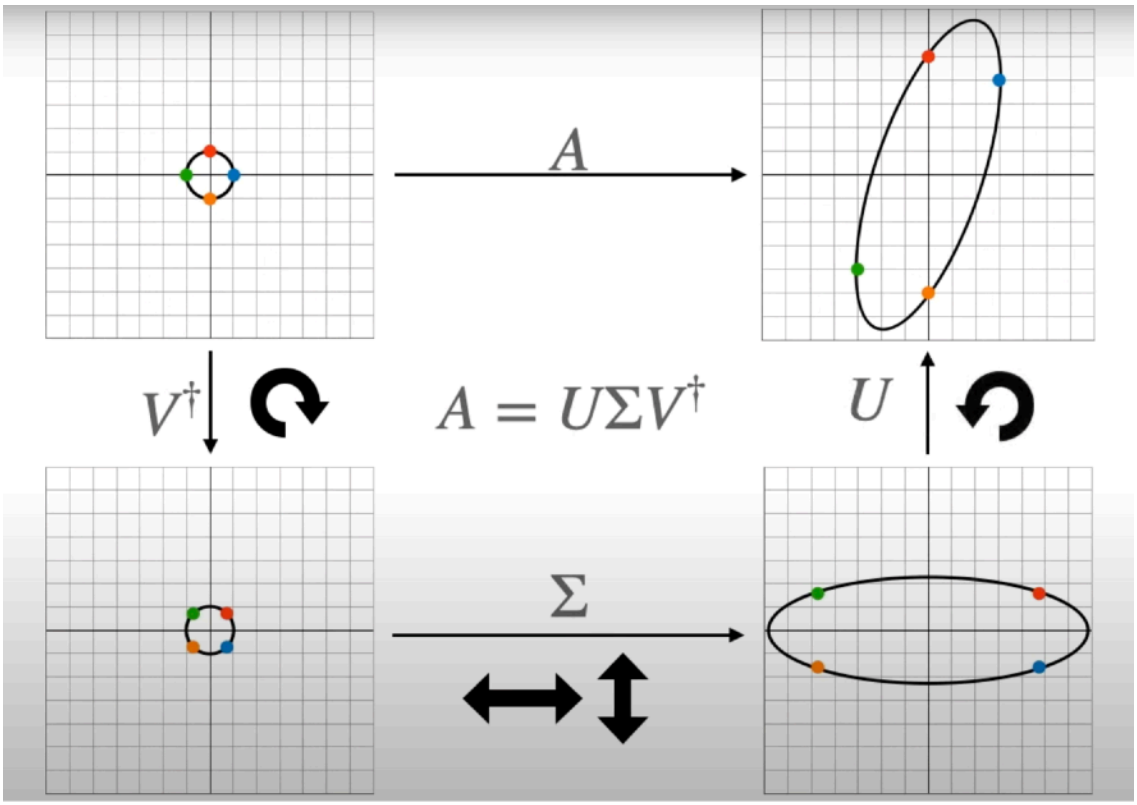
a 2D space. And let us take any 2x2 matrix say $\begin{bmatrix} 3 & 0 \\ 4 & 5 \end{bmatrix}$. As we can see from the above figure, each and every point gets moved (transformed) to a different position when we multiply our given set of points to the matrix. This is how linear transformation works.

Now some matrices are special - they can rotate a given figure to a particular angle but not change the dimensions of the figure. These matrices are always comprised of orthonormal vectors (orthogonal and unitary) so as to not stretch our given figure. An example of this can be $\begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix}$. Depending on θ our graph will be rotated to a particular angle with respect to original orientation.

Another kind of matrix is also available - those which can stretch our given figure but not rotate them. These are always diagonal matrices. An example will be $\begin{bmatrix} \sigma 1 & 0 \\ 0 & \sigma 2 \end{bmatrix}$. This matrix will stretch our x-axis by σ1 and our y-axis by σ2.

Singular Value Decomposition (SVD)

Now using the knowledge we gained above, we can come to the conclusion that any matrix can be re-written in the form of rotation matrices and stretching matrices. That is where SVD comes in to the picture. This theory states that any matrix A can be written as $A = U\Sigma V^T$ where U and V are our rotation matrices and Σ is our stretching matrix. Any vector in the column



space of A is in the column space of U and the same goes for the row space of A and row space of V. The above diagram is an easy way to understand what SVD is trying to do.

Say A is a 6x4 matrix, U will be a 6x6 matrix consisting of column vectors, Σ will be a 6x4 diagonal matrix and V^T will be a 4x4 matrix consisting of row vectors.

Some points to keep in mind about SVD :-

1) V must diagonalise $A^T A$ while U must diagonalise $A A^T$

2) v_i and u_i are eigenvectors of $A^T A$ and $A A^T$ respectively

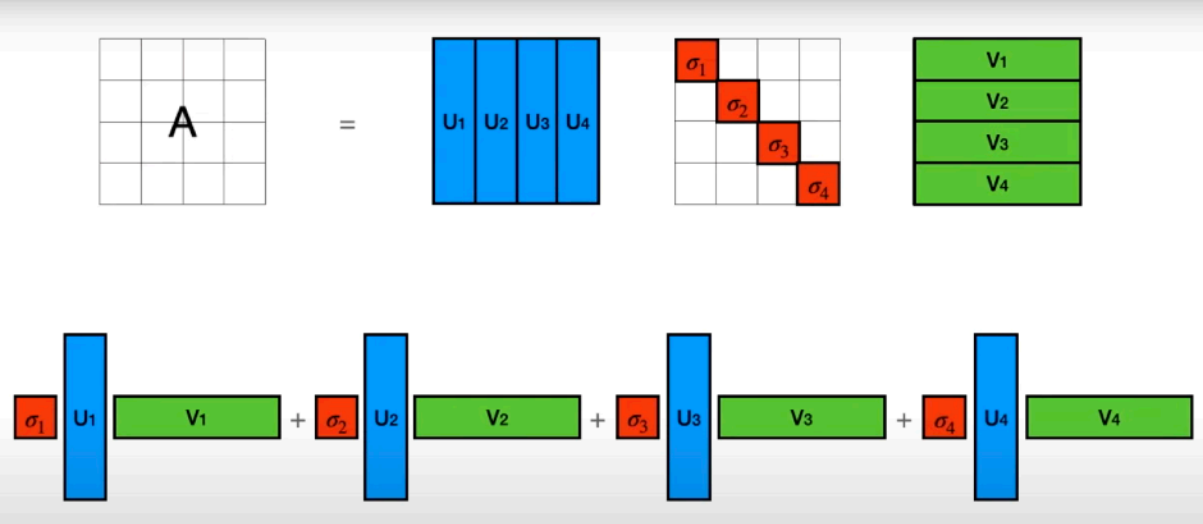
3) If A has rank r then v_i .... v_r forms orthonormal basis for range of A^T and u_i .... u_r forms orthonormal basis for range of A

Image Compression Using SVD

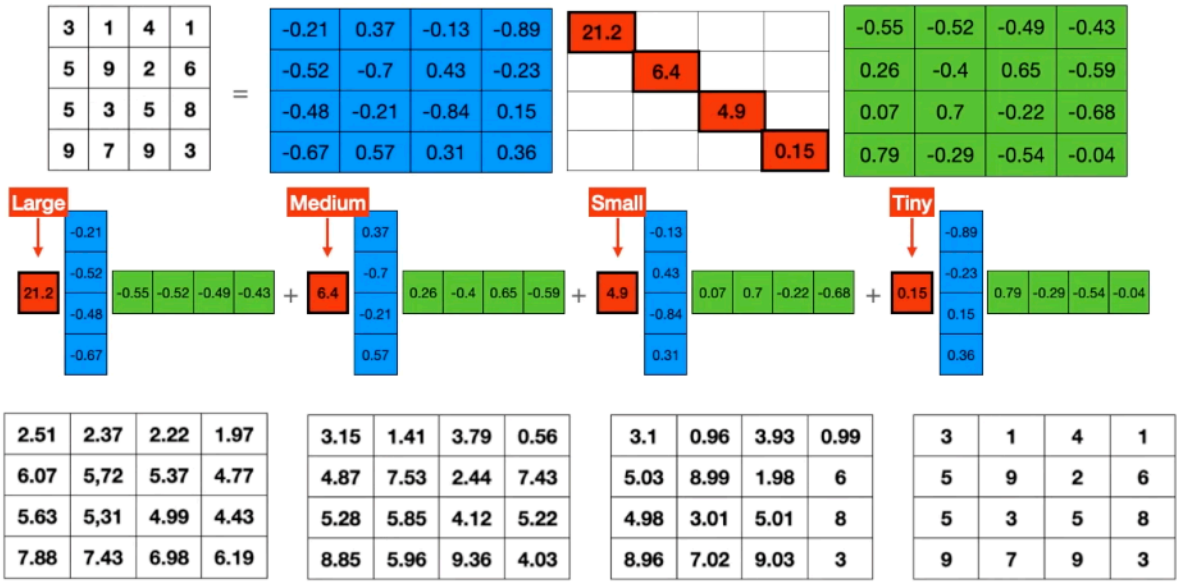
SVD is essentially trying to reduce a rank R matrix to a rank k matrix.

Lets say our image matrix is A and let $k \in N$ where $k \leq \text{rank}(A)$. What we are looking for is a matrix A_k having rank = k which is the best approximation of A among the matrices that have rank equal to k. This can be obtained by performing SVD of A and keeping the first k values of Σ the same and making the rest zero.

In other words, every matrix A can be rewritten in the form of multiple rank 1 matrices : $A = \Sigma_1^r \sigma_i u_i v_i^T$



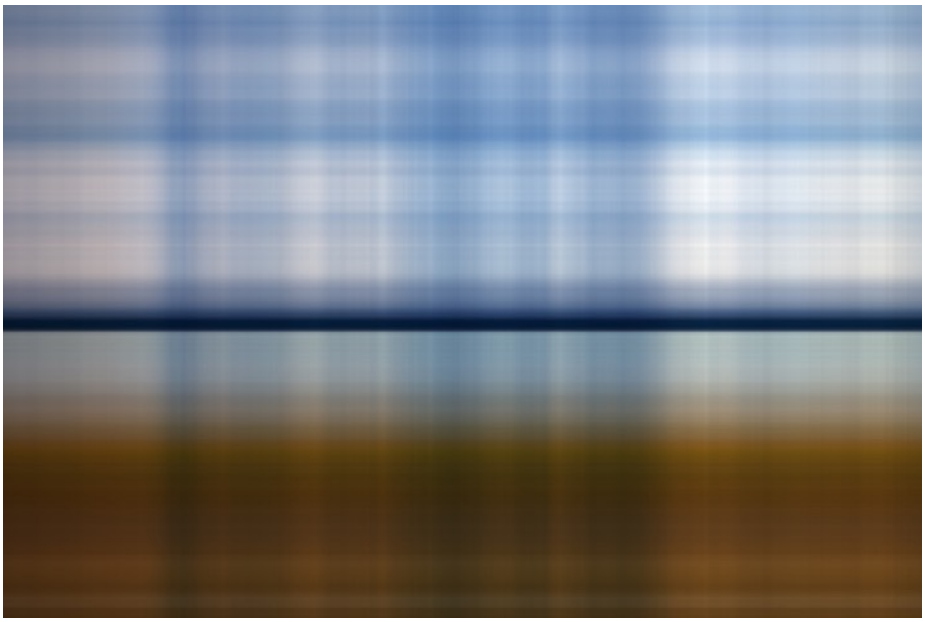
In this figure, the σ are called 'singular values' and those are the values by which we can approximate the matrix A by making them zero. Every successive singular value is less significant to our original matrix composition.



So from what we understood above, we can achieve image compression by reducing the number of singular values we are taking and hence reduce the number of unique row vectors.

Lets see how it looks at different limits:

1) LIMIT = 1



This first image is when we take our singular value limit as 1. Here, each row of pixels is the same, but has different brightness. Essentially each row can be written as $R_i = c_i * SV_i$, where c_i is the scaling factor and SV_i is the singular value vector with the highest contribution to the data.

2) LIMIT = 5

3) LIMIT = 25

4)LIMIT = 50

5) LIMIT = 100

6) LIMIT = 250

7) LIMIT = 400

8) ORIGINAL

Code used for executing image compression using SVD :

```
from PIL import Image
import numpy

#open the image and return 3 matrices, each corresponding to one channel (R, G, B)
def openImage(imagePath):
    imOriginal = Image.open(imagePath)
    im = numpy.array(imOriginal)

    aRed = im[:, :, 0]
    aGreen = im[:, :, 1]
    aBlue = im[:, :, 2]

    return [aRed, aGreen, aBlue, imOriginal]


#compress the matrix of a single channel
def compressSingleChannel(channelDataMatrix, singularValuesLimit):
    uChannel, sChannel, vhChannel = numpy.linalg.svd(channelDataMatrix)
    aChannelCompressed = numpy.zeros((channelDataMatrix.shape[0],
channelDataMatrix.shape[1]))
    k = singularValuesLimit

    leftSide = numpy.matmul(uChannel[:, 0:k], numpy.diag(sChannel)[0:k, 0:k])
    aChannelCompressedInner = numpy.matmul(leftSide, vhChannel[0:k, :])
    aChannelCompressed = aChannelCompressedInner.astype('uint8')
    return aChannelCompressed

#main program
print('*** Image Compression Using SVD ***')
aRed, aGreen, aBlue, originalImage = openImage('/Users/apricuz/Creative Cloud Files/
IMG_8062.jpg')


imageWidth = 512
imageHeight = 512

singularValuesLimit = 400

aRedCompressed = compressSingleChannel(aRed, singularValuesLimit)
aGreenCompressed = compressSingleChannel(aGreen, singularValuesLimit)
aBlueCompressed = compressSingleChannel(aBlue, singularValuesLimit)

imr = Image.fromarray(aRedCompressed, mode=None)
img = Image.fromarray(aGreenCompressed, mode=None)
imb = Image.fromarray(aBlueCompressed, mode=None)

newImage = Image.merge("RGB", (imr,img,imb))

originalImage.show()
newImage.show()
```