

Joe Narus
u0498279

Image Compression and Quality

Introduction:

Over the years, digital photography is becoming a larger part of our lives. We view hundreds of photos every day from the palm of our hands! Millions of these digital photos are stored all over the internet and take up a vast amount of memory, which is where image compression comes in handy. The main goal of image compression is to limit the amount of redundancy in an image so that it can be saved in a relatively efficient way that doesn't take up a lot of memory while also maintaining as much of the original image quality as possible. There are two categories of image compression: **lossless** and **lossy**. Lossless image compression, found in GIF, PNG, and TIFF file formats, is generally used for important images and online storage. Lossy compression, such as the Discrete Cosine Transform (DCT) algorithm and Singular Value Decomposition (SVD) compression, is mostly found within applications where bit rates in file transfers need to be low and quality isn't entirely necessary.

Abstract:

Using a grayscale image and the matlab coding language, I will explore two different types of image compression: DCT and SVD and determine what values best for maintaining quality, while also achieving a sufficient level of compression for both methods. Lastly, I'll briefly touch on other methods of compression, such as GIF and PNG, as a means of comparison for these two methods.

SVD compression:

Singular Value Decomposition image compression is achieved by using the decomposition to create an approximation of the original matrix. As taught in chapter 7, SVD is expressed as the equation:

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

Σ is the matrix containing diagonal entries of the non-zero singular values of A. A singular value is computed by taking the square root of an eigenvalue of the matrix $A^T A$. Σ is the same size as A, and zeros are filled in everywhere that does not contain a non-zero singular value. V is the matrix of eigenvectors corresponding to the appropriate eigenvalues of $A^T A$. Luckily Matlab makes this image compression fairly easy through the use of built in functions to handle finding singular values of matrices and rewriting image data.

The original image I used for my project is this batch of fruit:







My matlab function first grayscales the image to result in this image: (Makes rewriting easier)





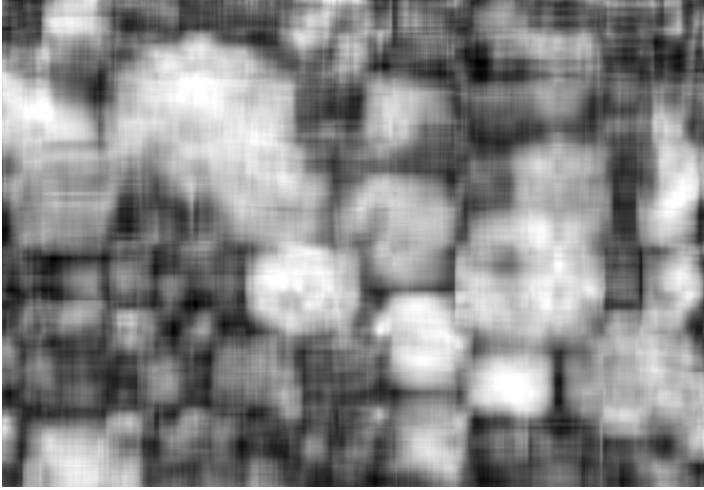
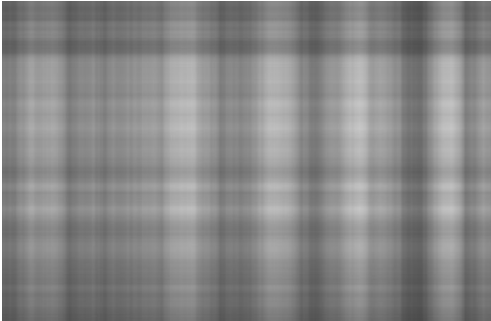
My driver function makes the call `image = SVD('mixed-fruits.jpg',value);`

Value is a float percent value that determines how many of the singular values (compr are used to create the approximation matrix. The following table shows the results of the different value numbers and their impact on image compression. A value of 0 results in the original image. A value of 1 causes it to use 0 of the singular values and makes a blob.

Value	# of singular vals	Image	Compression ratio
0	462 / 462		0 (195 KB)
.001	318 / 462		201 KB 1.0307

<p>.005</p>	<p>132 / 462</p>		<p>196 KB 1.005</p>
<p>.0075</p>	<p>91 / 462</p>		<p>187 KB .9589</p>



<p>.01</p>	<p>67 / 462</p>		<p>178 KB .9128</p>
<p>.03</p>	<p>21 / 462</p>		<p>146KB .7487</p>


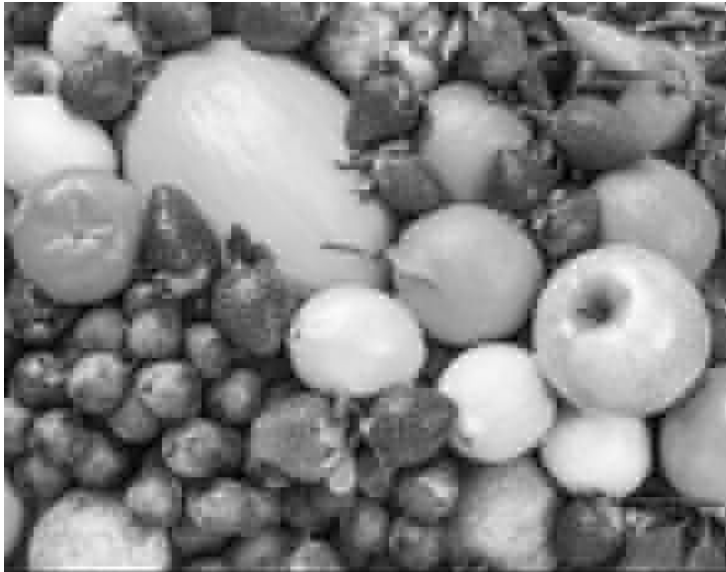
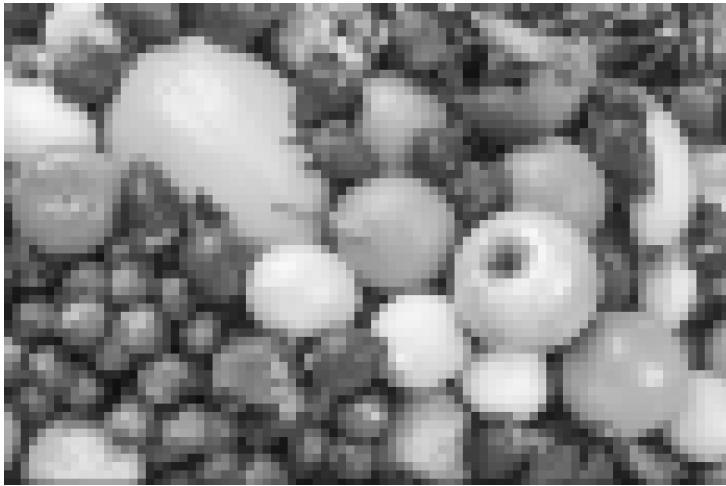
.06	11 / 462		129 KB .6615
1	0 / 462		75 KB .3846

Based on these results, the change in quality doesn't become super apparent until fewer than 100 of the singular values are being used. The image begins to lose very little quality and becomes slightly granulated. After dropping below 50 singular values, the image begins to lose the majority of its quality and it becomes harder to determine what the original image looked like. At 0 singular values, the image is entirely lost and it becomes a gray plaid pattern. My conclusion is that a value of .01-.015 (values are greater than 1%-1.5% of the maximum singular value), results in the best image compression that does not completely cut quality. The image may be granulated, but it's possible to tell what the image still is.

DCT Compression:

Discrete Cosine Transformation compression is achieved by finding the DCT coefficients for each 8-by-8 pixel block of the input image and finding the inverse DCT of each block, then multiplying them together with a mask to create the new, compressed image. Matlab has built in DCT functions that make computing these exceptionally easy. In the below table, the Mask Value column indicates the number of 1's in the upper left part of the 8-by-8 mask matrix.

Mask Value	Image	Compression Ratio
15		.9128 178KB
10		.8615 168KB

6		.77948 152KB
3		.5948 116KB
1		.05128 10KB

Based on these results, DCT manages to maintain image quality fairly well through lower compression ratios. Even at a mask value of 3, the image is still fairly clear and it's obvious that it's an image of fruit. Even at 1, the only really problem is how blurry the image is, but it still isn't absolutely awful. And the compression ratios are pretty impressive.

GIF and PNG:

GIF and PNG are file formats that use lossless compression to store pixel data. GIF compresses files by LZW encoding. This causes a partial loss in pixel data as the 15 bytes used to store pixel data (ARGB) is changed to 12 bytes. This decreases the size of large files significantly, but maintains a good portion of image quality. Similar to DCT.

PNG uses a combination of filtering followed by the DEFLATE compression algorithm. Filtering is the process of predicting the values of pixels based on previous pixels and then subtracting the predicted value from the actual value. This is done because a filtered line is generally easier to compress than the original line of data. Some PNG encoders pre-process images in a lossy way to make compression even easier, but results in data loss, so the practice isn't common.

PNG is one of the most common file types in web development because the images can be hosted on servers and rendered back to near their original quality. GIF is commonly used to store multiple frames of videos as a series of images that can be played like a video. Storing the image data is important for these so the story of the "video" won't be lost in bad compression.

CONCLUSION:

Based on the data collected, both of these algorithms are effective for image compression, but it seems as though DCT takes the cake in its ability to maintain the image quality with a higher compression ratio. How the change in image quality changed was the most obvious difference in the two compression styles. SVD led to a much more pixelated, blurry image and eventually became a gray plaid pattern. DCT on the other hand only became slightly more granulated/pixelated and when the mask value was taken to 0, the image became completely black rather than a gray plaid design as SVD displayed.

The compression ratios on DCT were much higher than SVD. SVD doesn't really gain a lot of compression as the number of singular values used are decreased. This means that the compression of SVD isn't very extensive and DCT wins in a contest of compression. Though this project focused mainly on grayscale images, I'm sure that the results would be relatively similar with the added layers of color.

References

DCT code based on matlab tutorial at:

<http://www-rohan.sdsu.edu/doc/matlab/toolbox/images/transfo8.html>

SVD code based on matlab documentation at:

<https://www.mathworks.com/help/matlab/ref/svd.html>

GIF:

https://en.wikipedia.org/wiki/GIF#File_format

PNG:

https://en.wikipedia.org/wiki/Portable_Network_Graphics#Compression