

Linear Algebra and Image Compression

Esmeralda Bess
May 7, 2012
University of Utah
Math 2270

I have researched two ways to use linear algebra in image compression. The first method that I will illustrate is using discrete cosine transformation (DCT). The second method is singular value decomposition (SVD).

DCT

The idea behind using DCT is to find the low frequencies in an image and discard unnecessary data. This will reduce the file size while preserving necessary data to allow the human eye to visualize a very similar image.

I did not create my own Matlab code and instead used this:

<http://www.mathworks.com/help/toolbox/images/f21-16366.html>

MathWorks DCT and Image Compression

```
I = imread('cameraman.tif');
I = im2double(I);
T = dctmtx(8);
dct = @(block_struct) T * block_struct.data * T';
B = blockproc(I,[8 8],dct);
mask = [
1 1 1 1 0 0 0 0
1 1 1 0 0 0 0 0
1 1 0 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0];
B2 = blockproc(B,[8 8],@(block_struct) mask .* block_struct.data);
invdct = @(block_struct) T' * block_struct.data * T;
I2 = blockproc(B2,[8 8],invdct);
imshow(I), figure, imshow(I2)
```

From the Matlab code, the "I" matrix has the image information that is turned into a matrix of RGB codes. The "T" in DCT is calculated using the third line in the code. This matrix is shown below. It is an 8x8 matrix that will be used to compress an image. The "mask" matrix from the code is what will control the compression ratio. In the sample code above, only 10 of the 64 elements in the matrix are 1's, and everything else is a 0. This tells us that about 10/64 amount of image data will be preserved, which we will check later. Finally, the "B2" matrix is the image that is decompressed from the calculated compressed DCT coefficient matrix "B". We will be creating compressed images such as "B2" using different values for "mask" to compare the amount of compression and image quality.

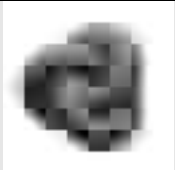








Calculated "T" matrix:

0.3536	0.3536	0.3536	0.3536	0.3536	0.3536	0.3536	0.3536
0.4904	0.4157	0.2778	0.0975	-0.0975	-0.2778	-0.4157	-0.4904
0.4619	0.1913	-0.1913	-0.4619	-0.4619	-0.1913	0.1913	0.4619
0.4157	-0.0975	-0.4904	-0.2778	0.2778	0.4904	0.0975	-0.4157
0.3536	-0.3536	-0.3536	0.3536	0.3536	-0.3536	-0.3536	0.3536
0.2778	-0.4904	0.0975	0.4157	-0.4157	-0.0975	0.4904	-0.2778
0.1913	-0.4619	0.4619	-0.1913	-0.1913	0.4619	-0.4619	0.1913
0.0975	-0.2778	0.4157	-0.4904	0.4904	-0.4157	0.2778	-0.0975

For all of my tests I will be using the original image 'knot.bmp' (size 5,174 bytes) taken from the class webpage:



From the code, since the "B" and "B2" matrices are 64x64, I have cut out an 8x8 piece from the same part of the image to compare the differences.

Discrete Cosine Transformation		
 mask: 3/64 ~ 5% 559 bytes	 mask: 6/64 ~ 9% 665 bytes	 mask: 10/64 ~ 15% 763 bytes
 mask: 15/64 ~ 23% 830 bytes	 mask: 21/64 ~ 32% 920 bytes	 mask: 28/64 ~ 43% 968 bytes
 mask: 36/64 ~ 56% 1,020 bytes	 mask: 43/64 ~ 67% 1,050 bytes	 mask: 49/64 ~ 76% 1,067 bytes

I feel that using a mask with 28 "1" values gives a good image quality. When I used a mask of all "1" values, I expected to see a very good image quality with a large file size. I was surprised by the results:



Clearly, it is a very good quality but it is not 100%. The file size of this image is 1,765 bytes. You can see a little distortion compared to the real original knot.bmp. I figured out that this is because Matlab further compresses the images shown, so it is not accurate to judge by the file size alone.

For the picture I chose as the ideal compression, calculating the compression ratio by the mask values keeps 36/64 amount of data which is about 56%. We discarded over 43% of data and still retained a very good idea of what the image is supposed to look like. In reality we discarded more than that, since the file size is 1,020 bytes compared to the original 5,174 bytes. This means that the image is 1020/5174, about 20% that of the original picture.

I was interested in seeing what the DCT coefficient matrices look like. Since they are 64x64, I decided to cut out an 8x8 piece from the same part of the image to compare the results.

DCT coefficient matrices

8x8 piece from original image:



Its DCT coefficient matrix:

1	1	1	1	0.996078431	0.996078431	0.992156863	0.71372549
1	1	1	0.996078431	1	0.980392157	0.584313725	0.411764706
1	1	1	1	0.984313725	0.545098039	0.384313725	0.470588235
1	1	0.996078431	1	0.576470588	0.349019608	0.439215686	0.443137255
1	0.992156863	1	0.71372549	0.305882353	0.4	0.407843137	0.411764706
0.996078431	1	0.91372549	0.337254902	0.341176471	0.376470588	0.380392157	0.380392157
0.992156863	1	0.576470588	0.254901961	0.345098039	0.349019608	0.349019608	0.349019608
1	0.933333333	0.28627451	0.278431373	0.31372549	0.321568627	0.321568627	0.309803922

8x8 piece from the "ideal" compression image (mask = 36/64)

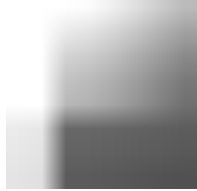


Its DCT coefficient matrix:

1.000031614	1.000038966	0.970032566	1.001874028	1.040501586	1.036792848	0.8949185	0.71075437
0.999805173	0.999929079	0.962319854	1.055024113	1.030298536	0.862746501	0.621899191	0.491361088

1.000466491	1.000345298	1.013676812	1.038985615	0.867345529	0.617558848	0.422981436	0.387407849
1.00102636	0.998688784	1.032036035	0.906870591	0.604893772	0.431119975	0.404761227	0.412767035
0.998688784	0.993164934	0.983664239	0.732887597	0.358972268	0.298547953	0.428891405	0.453667524
0.994126895	1.001793353	0.892820746	0.377464107	0.309208562	0.395886971	0.371389192	0.382397918
1.011604365	0.995965262	0.57364678	0.277292625	0.310016346	0.370926031	0.341186731	0.351506053
1.0268536	0.915159885	0.302340725	0.242201921	0.332554445	0.331338066	0.305837078	0.318516085

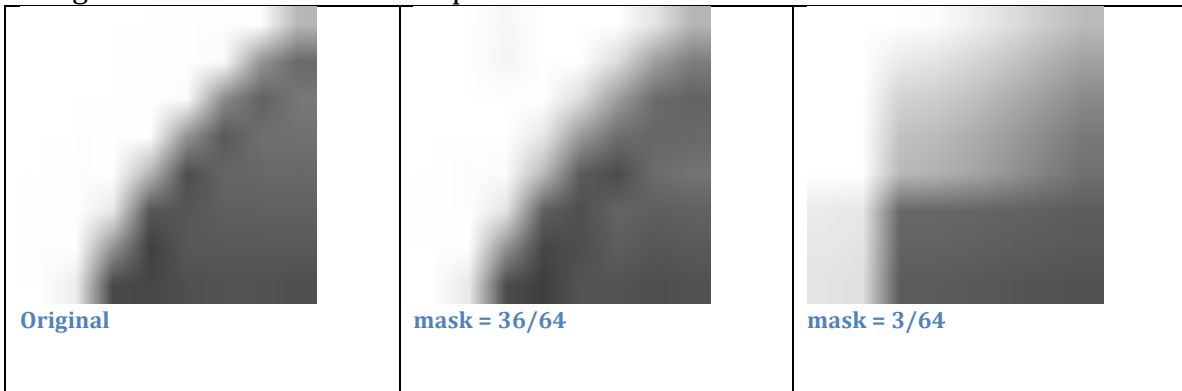
8x8 piece from the "bad" compression image (mask = 3/64)



Its DCT coefficient matrix:

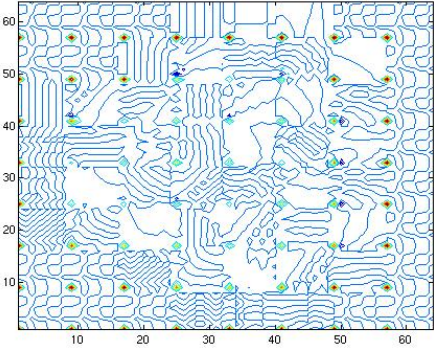
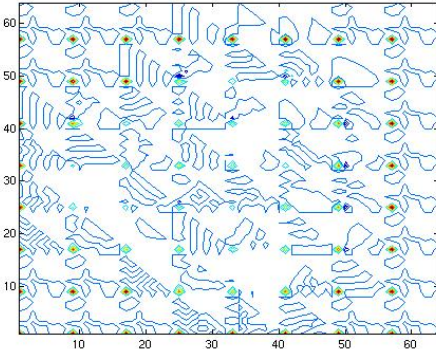
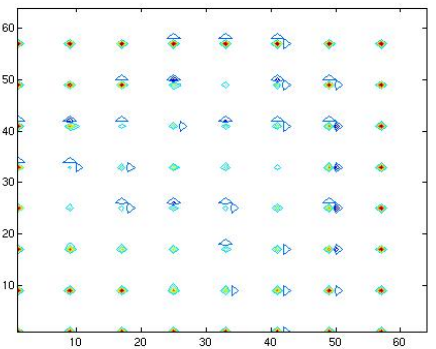
0.999724473	0.999688579	1.016012681	0.988016922	0.936287504	0.868699763	0.795543318	0.727955577
0.999630678	0.999594784	0.919784306	0.891788547	0.840059129	0.772471388	0.699314943	0.631727202
0.999544022	0.999508129	0.83088088	0.802885121	0.751155703	0.683567961	0.610411517	0.542823775
0.999477699	0.999441806	0.762837144	0.734841384	0.683111966	0.615524225	0.542367781	0.474780039
0.999441806	0.999405912	0.726012139	0.698016379	0.646286962	0.57869922	0.505542776	0.437955034
0.92036653	0.904497754	0.410514181	0.406156943	0.398105816	0.387586513	0.3762005	0.365681196
0.910678779	0.894810002	0.39327978	0.388922541	0.380871415	0.370352111	0.358966098	0.348446794
0.892778149	0.876909372	0.361434758	0.35707752	0.349026393	0.338507089	0.327121076	0.316601772

Images stretched out to see the pixels better



As you can see from the zoomed in pictures, the middle one strongly resembles the original knot image, and the last one has lost a lot of information.

I wanted to also compare the entire matrices, and since they are very large I decided to create plots instead.

	<p>Original DCT coefficient matrix.</p> <p>You can see a lot of data in this plot.</p>
	<p>mask = 36/64</p> <p>The amount of data is not as large as in the original, but there is still a lot of information in this plot.</p>
	<p>mask = 3/64</p> <p>You can clearly see how very little information was kept at this point.</p>

In conclusion, when deciding on how large the mask should be to retain decent image quality while still compressing the file size, I think around 50% shows a very good compromise. It is difficult to really compare the compression ratio since Matlab further compresses the images that it creates.

SVD

The second method of image compression that I decided to research is using singular value decomposition.

To calculate the singular value decomposition of a matrix M , we use the equation

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

To compress an image, we use the factorization of M , and calculate
 $\text{image} = U \Sigma \text{ones} V^T$


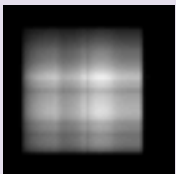
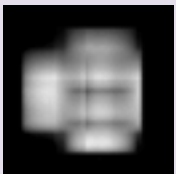
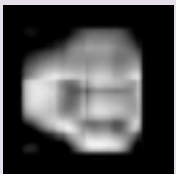
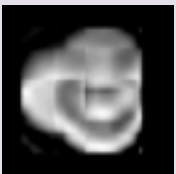
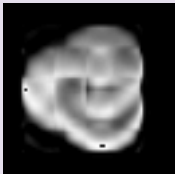
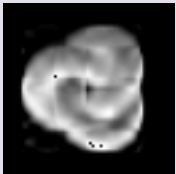
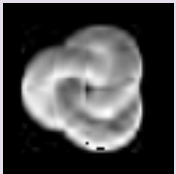
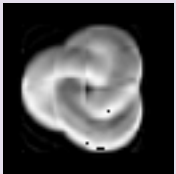
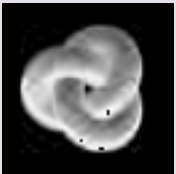
where "ones" is a diagonal matrix of 1's, filled only for the amount of singular values that you want to calculate.

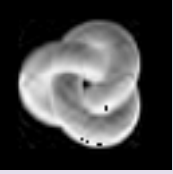






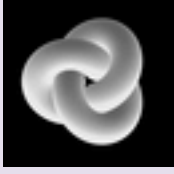


I used Matlab code that I found by searching for SVD image compression. Here is the link <http://math.la.asu.edu/~kawski/MATLAB/image/svdpix.m>

The code calculates the number of singular values to keep, starting at 1. The more singular values, the better the image quality. We are trying to find what is a good singular value to compress an image and still retain an acceptable image quality. Smaller singular values will result in more data lost than higher singular values, where more data is kept.

When I ran the Matlab code, it calculated 63 possible singular values for the 'knot.bmp' original picture.

Here are the results:

Singular Values per Image				
 original 1,531 bytes	 sing. values: 1 1,461 bytes	 sing. values: 2 1,587 bytes	 sing. values: 3 1,632 bytes	 sing. values: 4 1,750 bytes
 sing. values: 5 1,786 bytes	 sing. values: 6 1,827 bytes	 sing. values: 7 1,811 byte	 sing. values: 8 1,827 bytes	 sing. values: 9 1,832 bytes

 sing. values: 10	 sing. values: 15	 sing. values: 20	 sing. values: 25	 sing. values: 30
 sing. values: 35 1,571 bytes	 sing. values: 40 1,555 bytes	 sing. values: 50 1,531 bytes	 sing. values: 60 1,531 bytes	 sing. values: 63 1,531 bytes

The compressed image using 63 singular values is the same as the original picture, and if we look at all of the other images it looks like if you take a singular value between 20 and 25 you will get a good image quality.

It is once again difficult to judge the true image compression from looking at only the file size alone. The file size is not a good indicator of the compression ratio because displaying and subsequently storing an image through Matlab will distort the image size. For instance, if we look at knot.bmp, the original picture, the file size is 5,174 bytes, but according to Matlab when we run the SVD Compression code, the "original" file size is now only 1531 bytes. We can also see when the image is clearly distorted using few singular values (such as 5-9), the image quality is very poor yet the file size is larger than the claimed original.

The only conclusion I can form as far as the compression ratio is to take the number of singular values as the percentage of the amount of data retained. When the number of singular values used is 63, we are using 100% of the data since $63/63 = 1$. When we are only using 1 singular value, we are keeping $1/63$, which is about 1.6% this is not an obvious number judging by the compressed picture, but what is obvious is that the image is distorted beyond recognition. When we use 25 singular values, we are keeping $25/63$ values and this is about 40%. This is probably the ideal number to use.

I had a lot of fun seeing the images compress using both methods, and it would be a good experiment to truly see what the real compression ratios are in each case. It seems that in both DCT and SVD methods, getting around 50% compression will deliver a good image quality.