3SK3 Project

Applying Singular Value Decompositions for Image Compression

Surendra Sapkota / SAPKOS/1317270

Aquib Mohammad/ azharam/1319616

/Ibrahim Tannir / tanniri / 1204850

Moshiur Howlader /Howlam /1316948

Introduction:

In the digital world, there is a constant need to make processes more efficient. That being said, a major example of said efficiency and that have been taken place lately is, image processing algorithms. Digital images can be of very large size and can occupy a lots of storage space. Due to their larger size, they take larger bandwidth and more time for upload or download through the Internet. This also makes file sharing inconvenient. The counterpart to this problem, is to combat a special family of techniques called, “Image Compression”. It not only solves the problems stated above, but it also has many applications in the medical world and in CAD tools. The image compression applications reduce the size of an image file without causing major degradation to the quality of the image. There are two main techniques to this application; lossless and lossy. The latter is a perfect example of having a good trade-off between compression and quality. This project will break down all the elements that compose “Image Compression” and its different algorithms.

Objective:

The objective of this project is to understand and experiment image compression method using singular value decomposition (SVD) and SSVD.

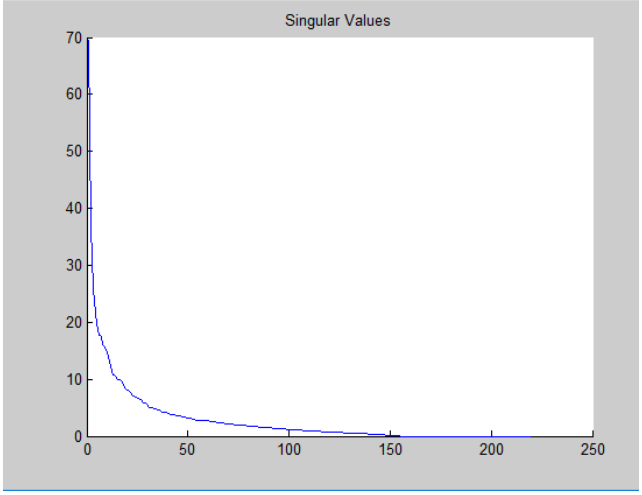
Part 1: Use SVD for Image Compression

Step 1.

ThresholdLogo.bmp

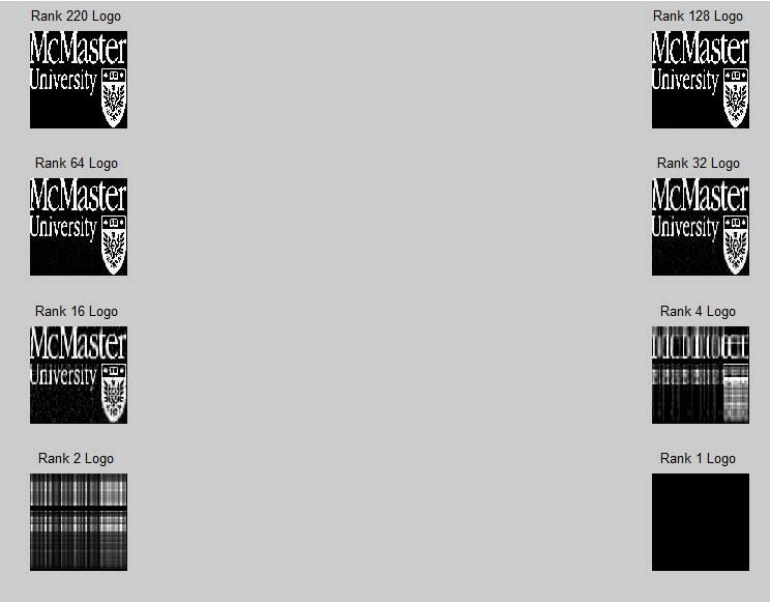
1_2

- **Plot for Singular Values**



1_3

- **Recompose input image with SVD rank approximate:**



1_4 & 1_5

Residual percentage energy error with Frobenius norm and compression rate of approximation from [220,128, 64, 32, 16, 4, 2, 1]

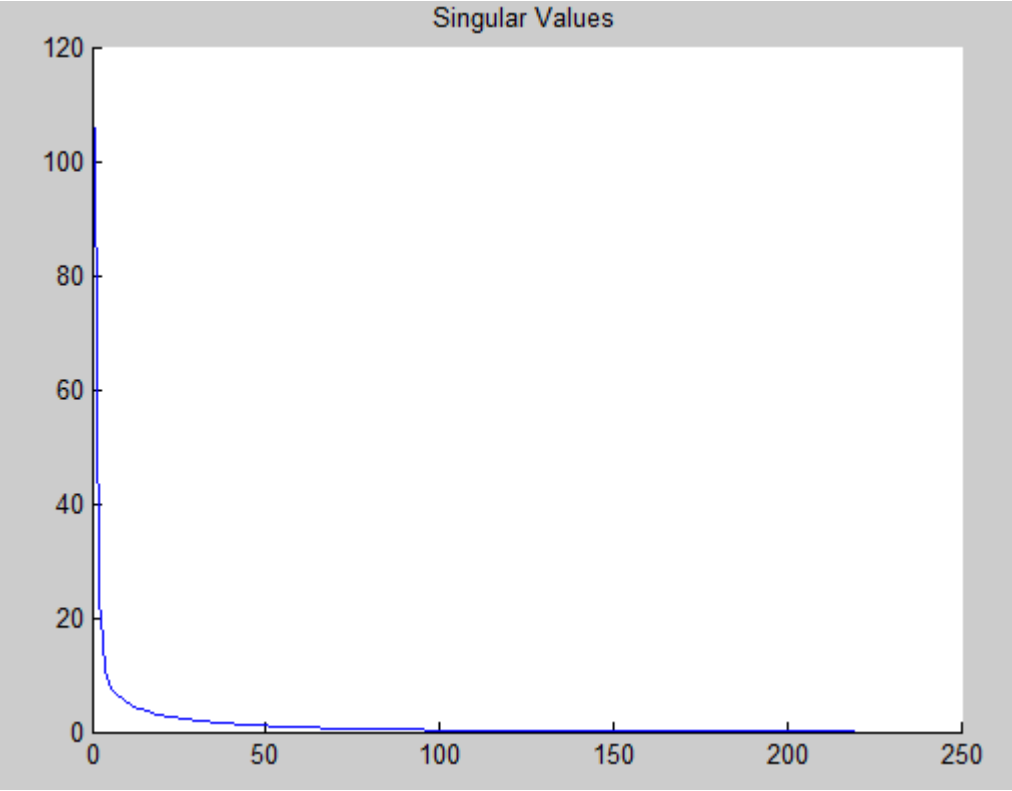
```
froNormDif =

  105.8867


residE =

     1


residEArray =

    0.0000    0.0185    0.1123    0.2210    0.3565    0.6238    0.7546    1.0000


cRate =

    2.0045    1.1663    0.5831    0.2916    0.1458    0.0364    0.0182    0.0091
```

Step 2.

Input image “Weave.bmp”

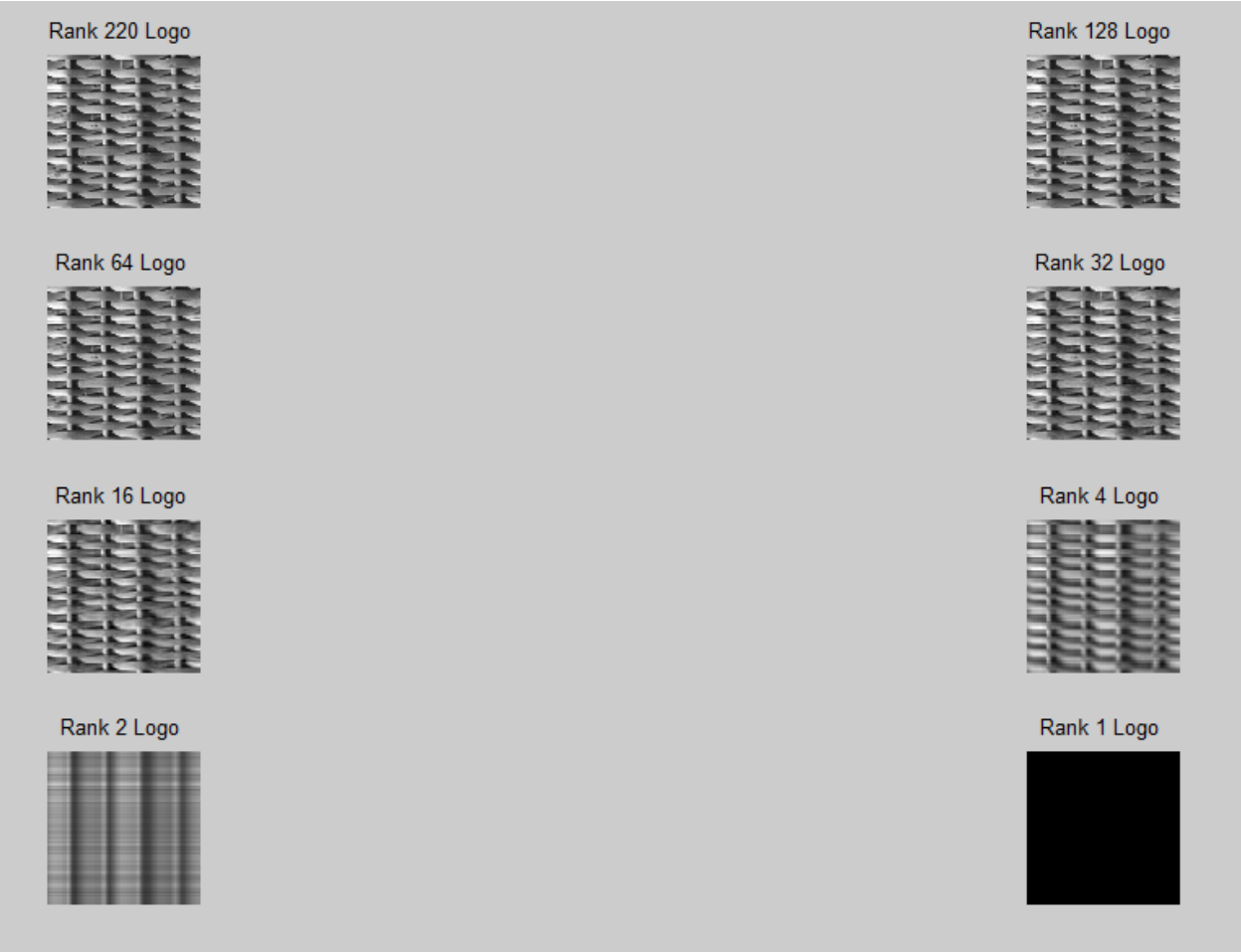
2_2

- **Singular Value Graph**



2_3

- **Recompose the input picture with the SVD rank approximation [220, 128, 64, 32, 16, 4, 2, 1]**



2_4 & 2_5

Residual percentage energy error with Frobenius norm and compression rate of approximation from [220,128, 64, 32, 16, 4, 2, 1]

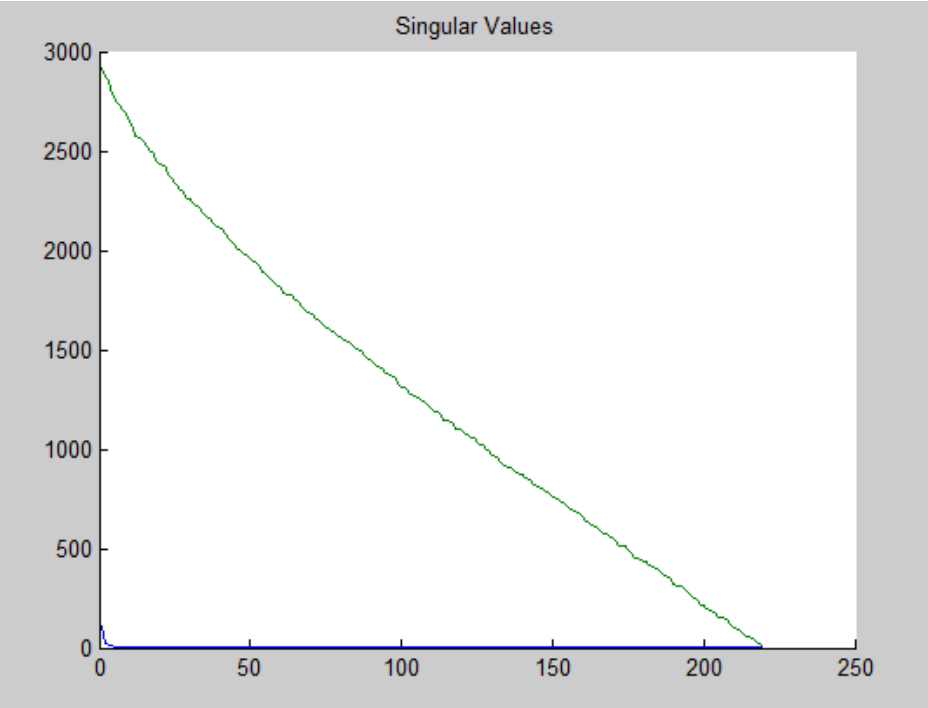
```
froNormDif =

  112.3977


residEArray =

    0.0000    0.0073    0.0305    0.0696    0.1169    0.2247    0.3373    1.0000


cRate =

    2.0045    1.1663    0.5831    0.2916    0.1458    0.0364    0.0182    0.0091
```

Step 3.

- **Step 1 with the input being a 220x220 matrix whose elements are picked from a Gaussian random variable with zero mean and 100 standard deviation**

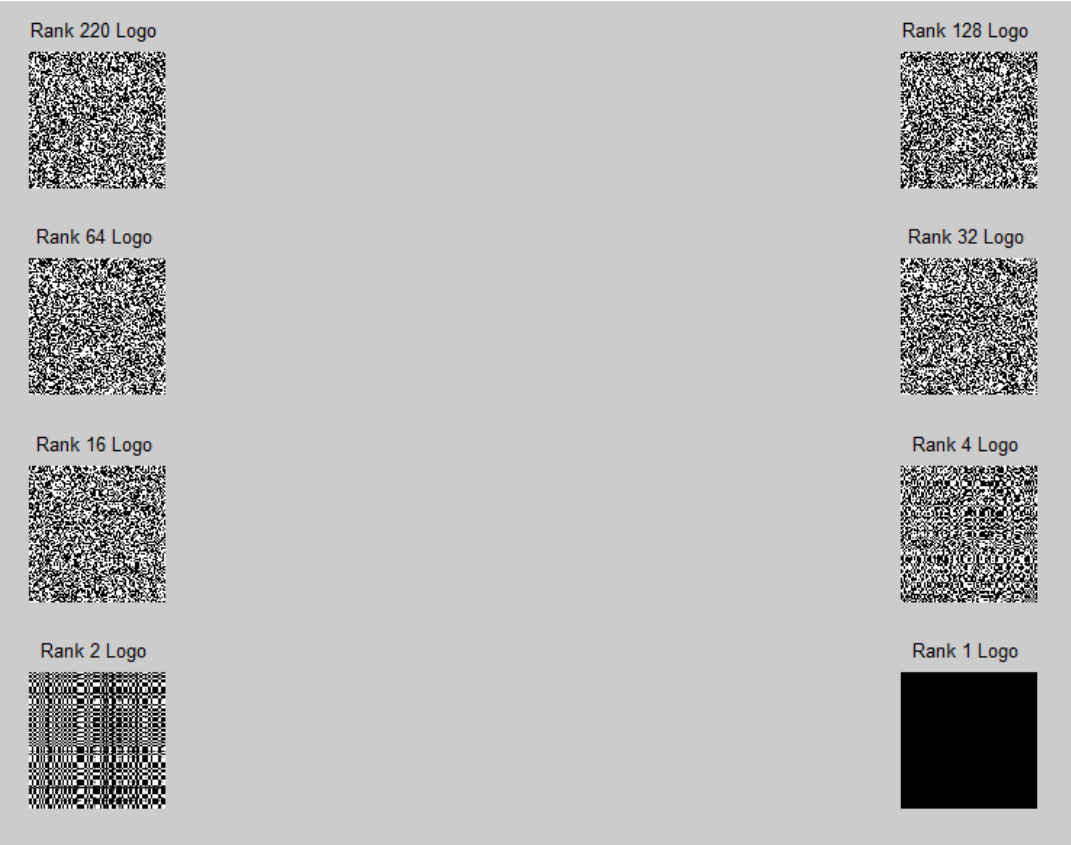
3_2

-
- **Singular Value Graph**



3_3

- **Recompose the input picture with the SVD rank approximation [220, 128, 64, 32, 16, 4, 2, 1]**



3_4 Residual percentage energy error with Frobenius norm and compression rate of approximation from [220,128, 64, 32, 16, 4, 2, 1]

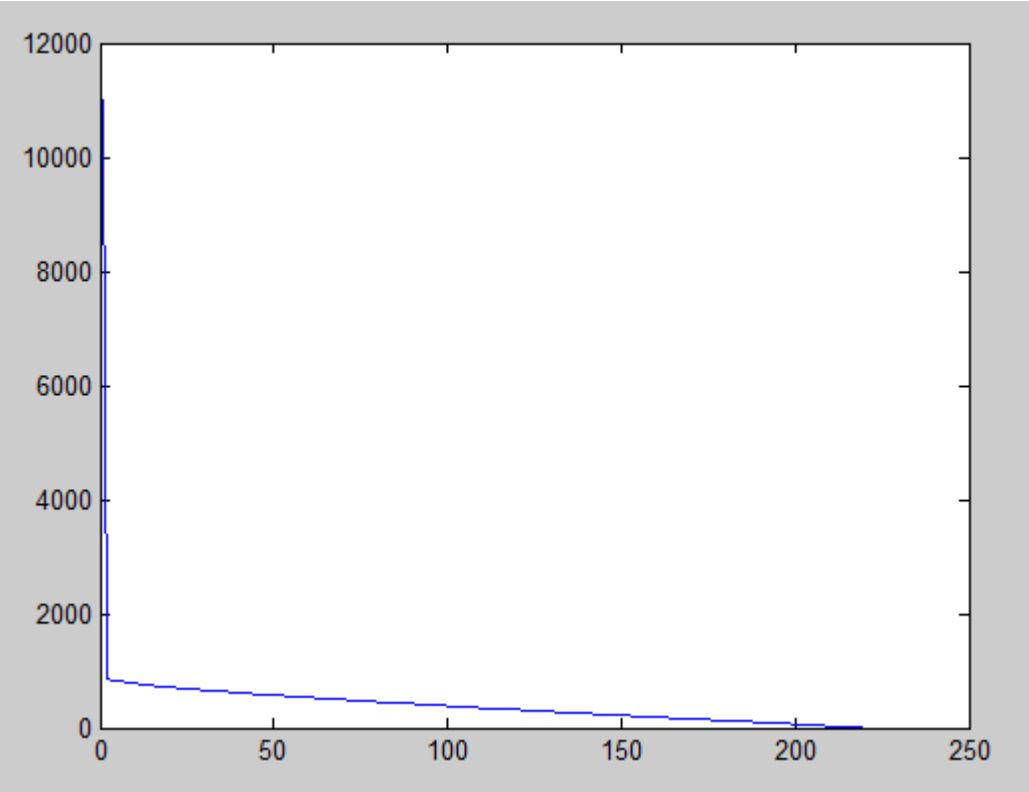
```
froNormDif =

   2.1994e+04


residE =

     1


residEArray =

    0.0001    0.2536    0.5687    0.7674    0.8796    0.9740    0.9912    1.0000


cRate =

    2.0045    1.1663    0.5831    0.2916    0.1458    0.0364    0.0182    0.0091
```

Step 4.

- **Step 1 with the input being a 220x220 matrix whose elements are picked from the uniform distribution between 0 and 100.**

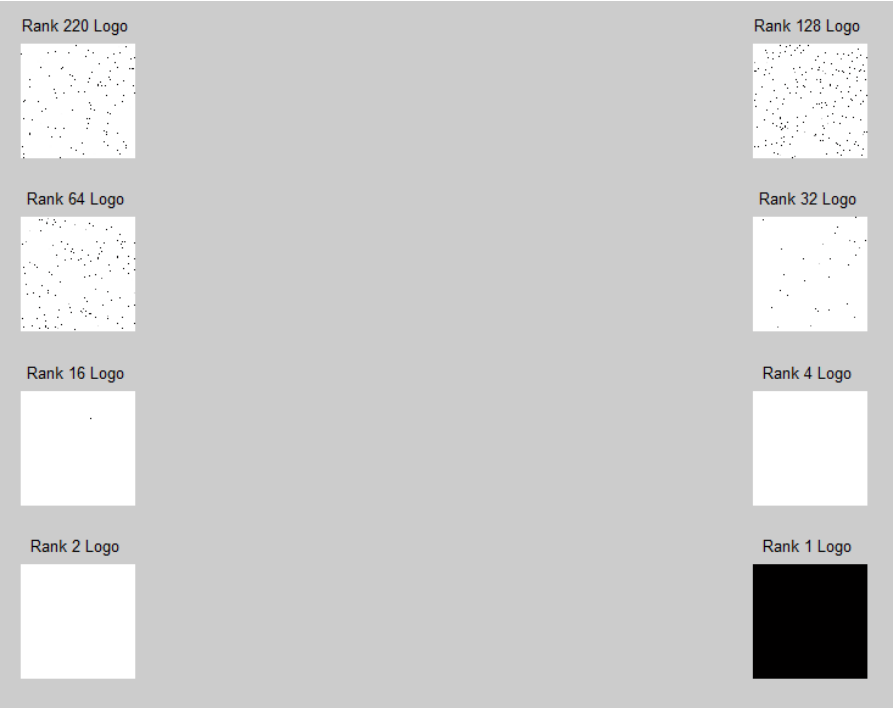
4_2

- **Singular Value Graph**



4_3

- **Recompose the input picture with the SVD rank approximation [220, 128, 64, 32, 16, 4, 2, 1]**



4_4 & 4_5

- **Residual percentage energy error with Frobenius norm and compression rate of approximation from [220,128, 64, 32, 16, 4, 2, 1]**

```
froNormDif =

   1.2710e+04


residE =

     1


residEArray =

    0.0001    0.1287    0.2879    0.3879    0.4450    0.4925    0.5013    1.0000


cRate =

    2.0045    1.1663    0.5831    0.2916    0.1458    0.0364    0.0182    0.0091
```

Discussion:

At which rank approximation can you make out what University this is in step 1?

- We can make out word "University in step 1's image at rank approximation of K=16.

At which ran k approximation can you make out that you are seeing a weave (step 2)?

- We can make out that we are seeing a weave in step 2's image at rank approximation of K=16 but weave looks bit clear at 32. Image look blurry before 16 but after 16 image looks sharper and sharper.

How quickly does the wording from step 1 come out as opposed to the crest?

- By comparison between wording and the crest in step1, the "McMaster University" logo image comes out clearly and faster compare to the crest. From output image, rank approximation for crest is higher than the wording or at K=32.

Does this seem to make sense with the percentage of energy left over in the approximation?

- The residual percentage energy error using Frobenius norm of the "McMaster University" logo image at k=16 there is 0.3565 energy error left compare to crest at k=32 there is 0.2210 energy error left. Therefore, as energy error decreases, image gets sharper.

Give an uneducated guess at which rank approximation a standard OCR program can decipher the wording from step 1?

- The standard OCR program can decipher the wording from step 1's image at rank approximation of K=16 but if we want to see crest clear we have to increase k.

At what point do you think this is an adequate approximation to jpeg say?

- At rank k = 32, picture becomes adequate approximation to .jpeg but at k=64 or higher image looks more clear.

At what point is the picture crystal clear?

- From the output image we can conclude that the image looks clear at rank k=64 and higher but it looks crystal clear at rank approximation of k=220. This is because rank approximation is equal to the number of columns and rows in the actual image's matrix.

In terms of Singular Values, below what value do you not care about?

- In terms of Singular Value (SDV), we don't care about the singular values that are equal or less than zeros.

Does this change for the natural scenes versus the statistical pictures?

- No this doesn't not change for the natural scenes versus the statistical pictures.

How do the Singular Value compare across the 4 different inputs?

- The output graphs of singular values are very similar for the step 1 and step 2 and graphs for step 3 and step 4 are similar as well. This is because graph in step 1 and step 2 we are dealing with bitmapped images but, in step 3 and step 4 we are dealing with random variables not to images.

What can you see visually in the pictures, that matches up to the drop off in Singular Values?

- Just by looking at output images and singular values graph, we can conclude that images are looks clearer and not distorted.

What is the first singular value telling you (To understand this you might want to play around with the mean of the input distribution in step 3?

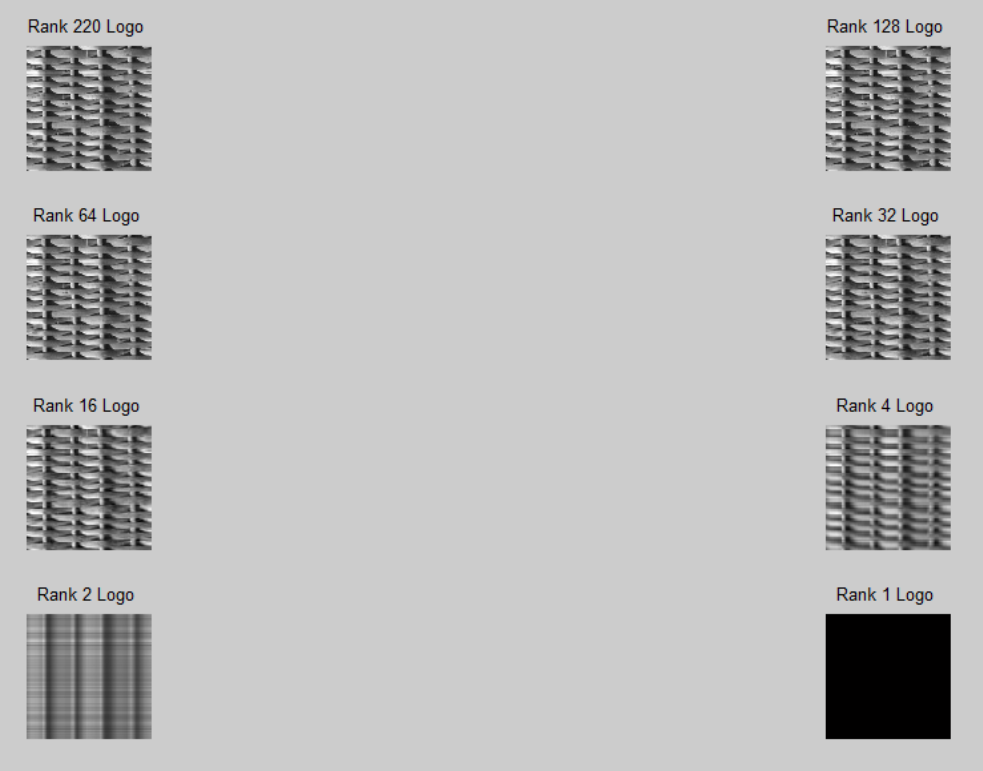
- The first singular value in singular matrix gives us the maximum singular value.

Part II: Use SSVD for Image Compression

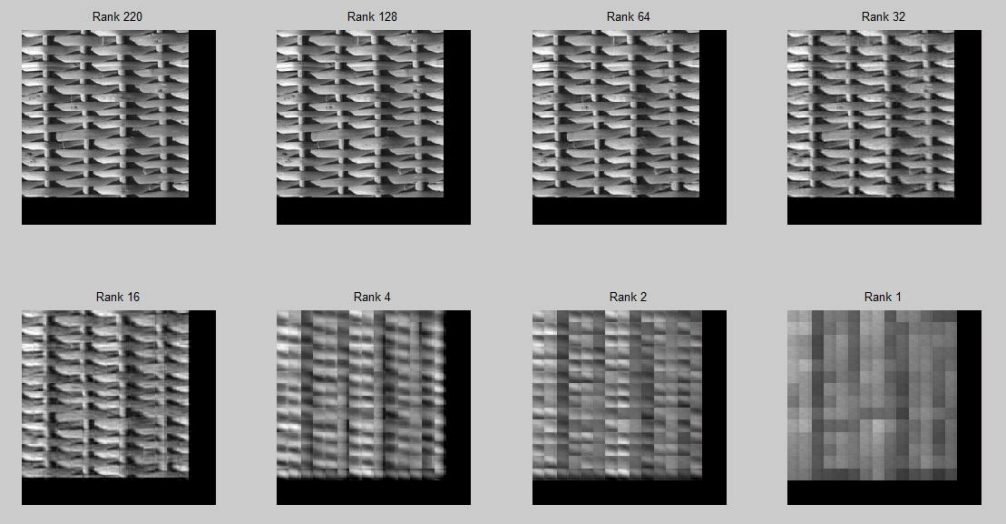
Step 1 & Step 2

Image Compression using SVD/SSVD

SVD



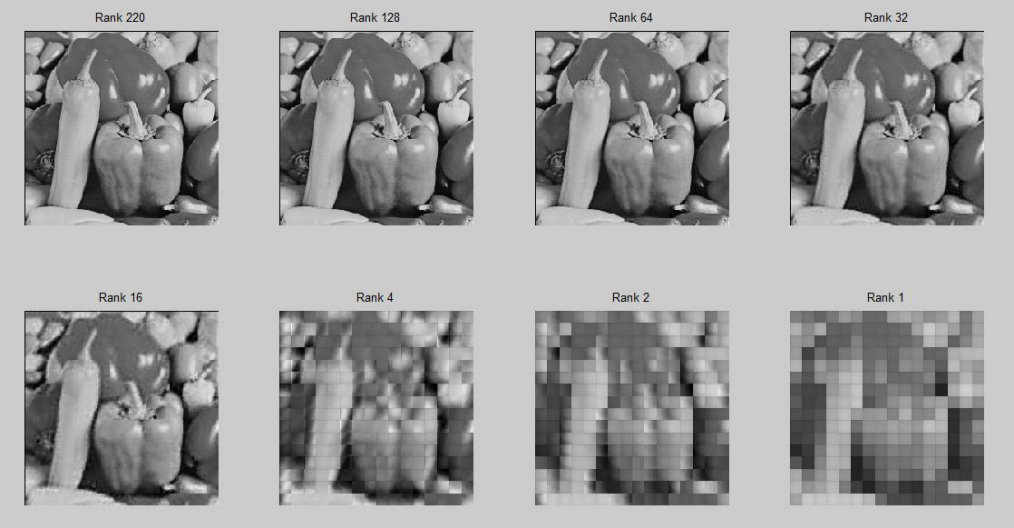
SSVD


Rank 220
Rank 128
Rank 64
Rank 32
Rank 16
Rank 4
Rank 2
Rank 1

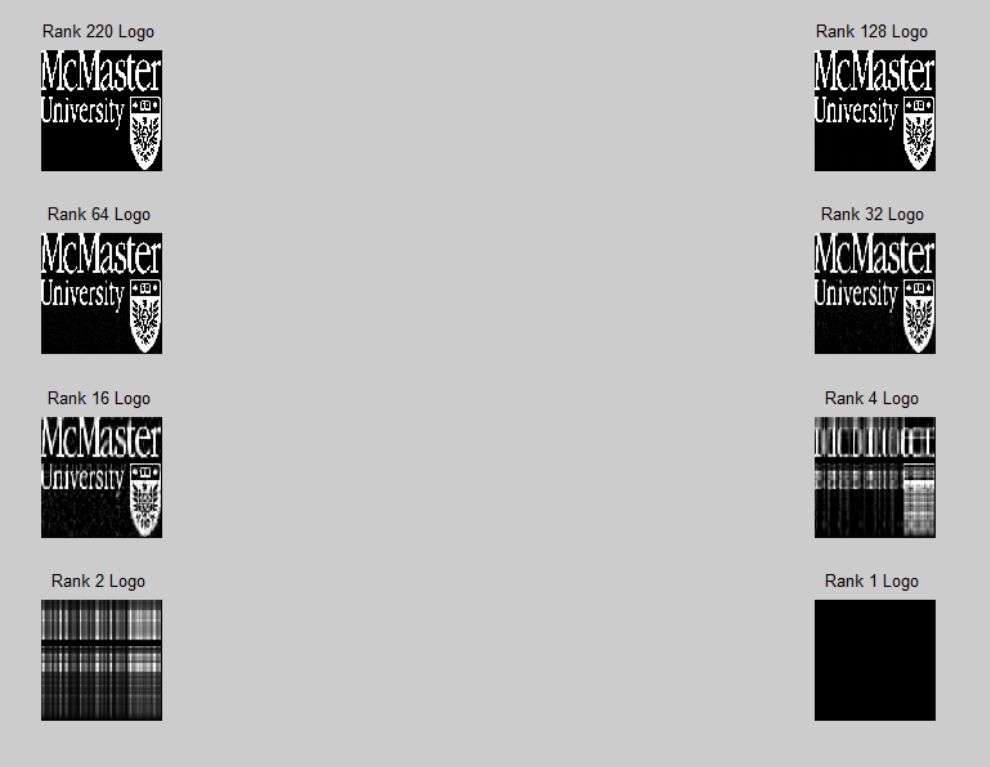
SVD


Rank 220 Logo
Rank 128 Logo
Rank 64 Logo
Rank 32 Logo
Rank 16 Logo
Rank 4 Logo
Rank 2 Logo
Rank 1 Logo

SSVD


Rank 220
Rank 128
Rank 64
Rank 32
Rank 16
Rank 4
Rank 2
Rank 1

SVD


Rank 220 Logo
Rank 128 Logo
Rank 64 Logo
Rank 32 Logo
Rank 16 Logo
Rank 4 Logo
Rank 2 Logo
Rank 1 Logo

SSVD


Rank 220
Rank 128
Rank 64
Rank 32
Rank 16
Rank 4
Rank 2
Rank 1

SVD


Rank 220 Logo
Rank 128 Logo
Rank 64 Logo
Rank 32 Logo
Rank 16 Logo
Rank 4 Logo
Rank 2 Logo
Rank 1 Logo

SSVD

Observation for Part II:

When we compare output images using SVD and SSVD, the SSVD image is clearer. This is due to the fact that the SSVD is refined version of the original method SVD. This is clear with all the other images as well.

Matlab Code was used to implement image compression

Matlab Code for Step 1:

```
%Project: Applying Singular Value Decompositions for Image Compression
close all;
hold all;
logoArray = imread('ThresholdLogo.bmp');
%image(logoArray);
logoArray = im2double(logoArray);
%logoArray = logoArray(:,:,1);
[U,S,V] = svd(logoArray);
sigmas = diag(S);
title('Singular Values')
plot(svd(logoArray))
figure;
title('Cumulative Percent of Total Sigmas');
ranks = [220, 128, 64, 32, 16, 4, 2, 1];
residEArray = [0, 0, 0, 0, 0, 0, 0, 0];
cRate = [0, 0, 0, 0, 0, 0, 0, 0];
for i = 1:length(ranks)
    % Keep largest singular values, and nullify others.
    approx_sigmas = sigmas;
    approx_sigmas(ranks(i):end) = 0;
    % Form the singular value matrix, padded as necessary
    ns = length(sigmas);
    approx_S = S;
    approx_S(1:ns, 1:ns) = diag(approx_sigmas);
    % Compute low-rank approximation by multiplying out component matrices.
    approx_logo = U * approx_S * V';
    froNormA = norm(logoArray,'fro');
    difArrayAB = logoArray - approx_logo;
    froNormDif = norm(difArrayAB, 'fro')
    residE = abs(froNormDif)/abs(froNormA)
    residEArray(i) = residE
    cRate(i) = (ranks(i)+ranks(i)*220+ranks(i)*220)/(220*220)
    % Plot approximation
    subplot(4, 2, i), imshow(approx_logo), title(sprintf('Rank %d Logo',
ranks(i)));
end
pause(3)
%close all;
```

Matlab Code for Step 2:

```
%Project: Applying Singular Value Decompositions for Image Compression
close all;

hold all;
logoArray = imread('Weave.bmp');
%image(logoArray);
logoArray = im2double(logoArray);
%logoArray = logoArray(:,:,1);
[U,S,V] = svd(logoArray);

sigmas = diag(S);
```

```
title('Singular Values')
plot(svd(logoArray))
figure;
ranks = [220, 128, 64, 32, 16, 4, 2, 1];
residEArray = [0, 0, 0, 0, 0, 0, 0, 0];
cRate = [0, 0, 0, 0, 0, 0, 0. 0];
for i = 1:length(ranks)
    % Keep largest singular values, and nullify others.
    approx_sigmas = sigmas;
    approx_sigmas(ranks(i):end) = 0;
    % Form the singular value matrix, padded as necessary
    ns = length(sigmas);
    approx_S = S;
    approx_S(1:ns, 1:ns) = diag(approx_sigmas);
    % Compute low-rank approximation by multiplying out component matrices.
    approx_logo = U * approx_S * V';
    froNormA = norm(logoArray,'fro');
    difArrayAB = logoArray - approx_logo;
    froNormDif = norm(difArrayAB, 'fro')
    residE = abs(froNormDif)/abs(froNormA);
    residEArray(i) = residE
    cRate(i) = (ranks(i)+ranks(i)*220+ranks(i)*220)/(220*220)
    % Plot approximation
    subplot(4, 2, i), imshow(approx_logo), title(sprintf('Rank %d Logo',
ranks(i)));
end
```

Matlab Code for Step 3

```
mu = 0;
sigma = 100;
myImage(1:220, 1:220) = 0;
for i=1:220
    for j = 1:220
        myRandom = normrnd(mu, sigma);
        myImage(i,j) = myRandom;
    end
end
%logoArray = imread('ThresholdLogo.bmp');
%image(logoArray);
%logoArray = im2double(logoArray);
%logoArray = logoArray(:,:,1);
logoArray= myImage;
[U,S,V] = svd(logoArray);
sigmas = diag(S);
title('Singular Values')
plot(svd(logoArray))
figure;
ranks = [220, 128, 64, 32, 16, 4, 2, 1];
residEArray = [0, 0, 0, 0, 0, 0, 0, 0];
cRate = [0, 0, 0, 0, 0, 0, 0, 0];
for i = 1:length(ranks)
```

```
    % Keep largest singular values, and nullify others.
    approx_sigmas = sigmas;
    approx_sigmas(ranks(i):end) = 0;
    % Form the singular value matrix, padded as necessary
    ns = length(sigmas);
    approx_S = S;
    approx_S(1:ns, 1:ns) = diag(approx_sigmas);
    % Compute low-rank approximation by multiplying out component matrices.
    approx_logo = U * approx_S * V';
    froNormA = norm(logoArray,'fro');
    difArrayAB = logoArray - approx_logo;
    froNormDif = norm(difArrayAB, 'fro')
    residE = abs(froNormDif)/abs(froNormA)
    residEArray(i) = residE
    cRate(i) = (ranks(i)+ranks(i)*220+ranks(i)*220)/(220*220)
    % Plot approximation
    subplot(4, 2, i), imshow(approx_logo), title(sprintf('Rank %d Logo',
ranks(i)));
end
```

Matlab Code for Step 4:

```
close all;
myImage(1:220, 1:220) = 0;
for i=1:220
    for j = 1:220
        myRandom = randi(101) - 1;
        myImage(i,j) = myRandom;
    end
end
logoArray= myImage;
[U,S,V] = svd(logoArray);
sigmas = diag(S);
figure();
hold on;
plot(log10(sigmas));
title('Singular Values (Log10 Scale)');
figure();
plot(cumsum(sigmas) / sum(sigmas));
title('Cumulative Percent of Total Sigmas');
ranks = [220, 128, 64, 32, 16, 4, 2, 1];
residEArray = [0, 0, 0, 0, 0, 0, 0, 0];
cRate = [0, 0, 0, 0, 0, 0, 0, 0];
for i = 1:length(ranks)
    % Keep largest singular values, and nullify others.
    approx_sigmas = sigmas;
    approx_sigmas(ranks(i):end) = 0;
    % Form the singular value matrix, padded as necessary
    ns = length(sigmas);
    approx_S = S;
    approx_S(1:ns, 1:ns) = diag(approx_sigmas);
    % Compute low-rank approximation by multiplying out component matrices.
    approx_logo = U * approx_S * V';
    froNormA = norm(logoArray,'fro');
    difArrayAB = logoArray - approx_logo;
    froNormDif = norm(difArrayAB, 'fro')
    residE = abs(froNormDif)/abs(froNormA)
```

```
    residEArray(i) = residE
    cRate(i) = (ranks(i)+ranks(i)*220+ranks(i)*220)/(220*220)
    % Plot approximation
    subplot(4, 2, i), imshow(approx_logo), title(sprintf('Rank %d Logo',
ranks(i)));
end
```

Matlab Code for Part II

```
close all;
clear all;
A = imread('Weave.bmp');
A = im2double(A);
n=16;
X(1:256, 1:256) = 0;
Y(1:256, 1:256) = 0;
%copy the thresholdlogo into the matrix X with the extra padded zeros
for i=1:220
    for j=1:220
        X(i,j)=A(i,j);
        Y(i,j)=A(i,j);
    end
end
if size(A) == 256
    X = A;
    Y = A;
end
%shuffle the matrix and save it into X
for i=1:256
    for j=1:256
        alpha=n*floor((i-1)/16)+floor((j-1)/16)+1;
        beta=n*mod((i-1),n)+mod((j-1),n)+1;
        X(alpha,beta)=Y(i,j);
    end
end
%Take the SVD of the shuffled image
[U,S,V] = svd(X);
approx  = [1,2,4,16,32,64,128,220];
%compute the approximations
rank1 = U(:,1:approx(1))*S(1:approx(1),1:approx(1))*V(:,1:approx(1))';
rank2 = U(:,1:approx(2))*S(1:approx(2),1:approx(2))*V(:,1:approx(2))';
rank4 = U(:,1:approx(3))*S(1:approx(3),1:approx(3))*V(:,1:approx(3))';
rank16 = U(:,1:approx(4))*S(1:approx(4),1:approx(4))*V(:,1:approx(4))';
rank32 = U(:,1:approx(5))*S(1:approx(5),1:approx(5))*V(:,1:approx(5))';
rank64 = U(:,1:approx(6))*S(1:approx(6),1:approx(6))*V(:,1:approx(6))';
rank128 = U(:,1:approx(7))*S(1:approx(7),1:approx(7))*V(:,1:approx(7))';
rank220 = U(:,1:approx(8))*S(1:approx(8),1:approx(8))*V(:,1:approx(8))';
disp = Y;
for i=1:256
    for j=1:256
        alpha=n*floor((i-1)/16)+floor((j-1)/16)+1;
        beta=n*mod((i-1),n)+mod((j-1),n)+1;
        disp(alpha,beta)=rank1(i,j);
    end
```

```
end
subplot(2,4,8);
imshow(disp);
title('Rank 1');

disp = Y;
for i=1:256
    for j=1:256
        alpha=n*floor((i-1)/16)+floor((j-1)/16)+1;
        beta=n*mod((i-1),n)+mod((j-1),n)+1;
        disp(alpha,beta)=rank2(i,j);
    end
end
subplot(2,4,7);
imshow(disp);
title('Rank 2');

disp = Y;
for i=1:256
    for j=1:256
        alpha=n*floor((i-1)/16)+floor((j-1)/16)+1;
        beta=n*mod((i-1),n)+mod((j-1),n)+1;
        disp(alpha,beta)=rank4(i,j);
    end
end
subplot(2,4,6);
imshow(disp);
title('Rank 4');

disp = Y;
for i=1:256
    for j=1:256
        alpha=n*floor((i-1)/16)+floor((j-1)/16)+1;
        beta=n*mod((i-1),n)+mod((j-1),n)+1;
        disp(alpha,beta)=rank16(i,j);
    end
end
subplot(2,4,5);
imshow(disp);
title('Rank 16');

disp = Y;
for i=1:256
    for j=1:256
        alpha=n*floor((i-1)/16)+floor((j-1)/16)+1;
        beta=n*mod((i-1),n)+mod((j-1),n)+1;
        disp(alpha,beta)=rank32(i,j);
    end
end
subplot(2,4,4);
imshow(disp);
title('Rank 32');

disp = Y;
for i=1:256
    for j=1:256
        alpha=n*floor((i-1)/16)+floor((j-1)/16)+1;
```

```
        beta=n*mod((i-1),n)+mod((j-1),n)+1;
        disp(alpha,beta)=rank64(i,j);
    end
end
subplot(2,4,3);
imshow(disp);
title('Rank 64');

disp = Y;
for i=1:256
    for j=1:256
        alpha=n*floor((i-1)/16)+floor((j-1)/16)+1;
        beta=n*mod((i-1),n)+mod((j-1),n)+1;
        disp(alpha,beta)=rank128(i,j);
    end
end
subplot(2,4,2);
imshow(disp);
title('Rank 128');

disp = Y;
for i=1:256
    for j=1:256
        alpha=n*floor((i-1)/16)+floor((j-1)/16)+1;
        beta=n*mod((i-1),n)+mod((j-1),n)+1;
        disp(alpha,beta)=rank220(i,j);
    end
end
subplot(2,4,1);
imshow(disp);
title('Rank 220');
```