Indian Institute of Technology, Madras

Pattern Recognition

CS6690

Assignment 1

Submitted by:
Athul Vijayan
ED11B004

August 29, 2014

The given images are 24bit color depth, i.e. each of R, G, B is stored as 8 bits. The values for each color in each pixel can be put in a matrix form.

1 Singular Value decomposition

Any m × n matrix can be factored as $A = U\Sigma V^T$ where U is an m × m orthogonal matrix, V is an n × n orthogonal matrix, and Σ is an m × n matrix with entries $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r$ down the main diagonal and zeros elsewhere.
If the columns of U are called $u_1, u_2, ..., u_m$ and the columns of V are called $v_1, v_2, ..., v_n$, the equivalent statement that every matrix A can be written

$$A = U\Sigma V^T \tag{1}$$
$$A = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \cdots + \sigma_r u_r v_r^T \tag{2}$$

This is the form of the SVD which is used in image compression. Image compression might be useful if you have an image to transfer electronically, say over the internet or from a satellite transmission. Perhaps it is possible to not transmit all of the data, but still transmit enough of it so that the essential information is there and the image can be reconstructed sufficently on the receiving end. The SVD accomplishes this. The idea is that since the σ_i are ordered from large to small, we can get a good approximation to A by throwing out the terms toward the end of the expansion.
But in the assignment, we experiment with the singular values that we throw out. and make inferences by checking the RMS Error in the image recreated using reduced dimensions.

$$A = U\Sigma V^T \tag{3}$$
$$\hat{A} = U\hat{\Sigma}V^T$$
$$RMSE = rms(A - \hat{A}) \tag{4}$$

1. First we Do it by applying svd to each channel seperately and merge them after processing.

   (a) Seperate channels and apply svd to them

```
% In this part we do the svd analysis of the input image and plot the results.
%% svdAnanlysis: function to get image name as argument
clear all
inputImage = '29.jpg';
oriImg = imread(inputImage);
figure;
image(oriImg);
imageInfo = imfinfo(inputImage);
imWidth = imageInfo.Width;
imHeight = imageInfo.Height;

% Matlab scales the values from 0-1
imgRChannel = im2double(oriImg(:,:,1));          % extract R channel
imgGChannel = im2double(oriImg(:,:,2));          % extract G channel
imgBChannel = im2double(oriImg(:,:,3));          % extract B channel

% first do svd on each channels seperately, choose different singular values
    and recreate each channel.
% then we remake the image using new channels
[U_R, S_R, V_R] = svd(imgRChannel);
[U_G, S_G, V_G] = svd(imgGChannel);
[U_B, S_B, V_B] = svd(imgBChannel);
```

For the following parts, the sequence of three images in a row will show original image, reconstructed image and error image respectively.
The following code will be used for reconstruction and plotting of images.

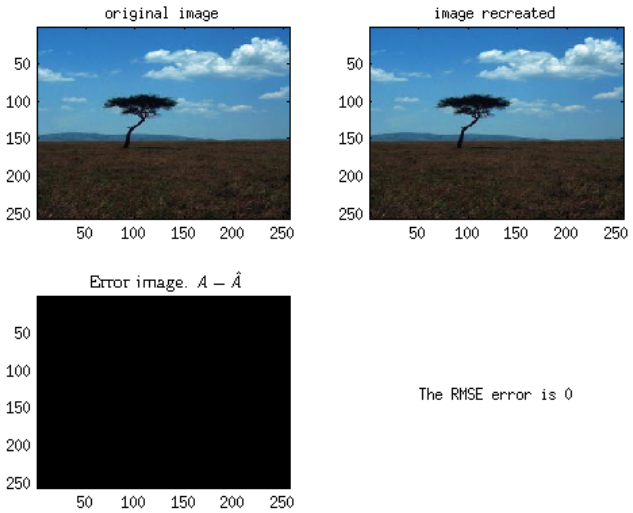
```
S_Rhat = S_R(choosenSVs, choosenSVs); %selected singular vectors
imgRhat = 255*U_R(:, choosenSVs)*S_Rhat*transpose(V_R(:, choosenSVs));
    %recreate R channel

S_Ghat = S_G(choosenSVs, choosenSVs); %selected singular vectors
imgGhat = 255*U_G(:, choosenSVs)*S_Ghat*transpose(V_G(:, choosenSVs));
    %recreate G channel

S_Bhat = S_B(choosenSVs, choosenSVs); %selected singular vectors
imgBhat = 255*U_B(:, choosenSVs)*S_Bhat*transpose(V_B(:, choosenSVs));
    %recreate B channel

imgHat = uint8(imgRhat);
imgHat(:,:,2) = uint8(imgGhat); imgHat(:,:,3) = uint8(imgBhat);
RMSE = rms(rms(rms(oriImg - imgHat)));

subplot(2, 2,2);
image(imgHat);
title('image recreated')
subplot(2, 2, 3);
image(oriImg-imgHat);
title('Error image. $$A - \hat{A}$$', 'Interpreter','Latex')
tsp = subplot(2,2,4);
text(0.2,0.5,['The RMSE error is ', num2str(RMSE)]);
set ( tsp, 'visible', 'off')
```

Now we can play with chosenSVs to for different cases.

(b) Chosen Singular values are 'Top N values'.

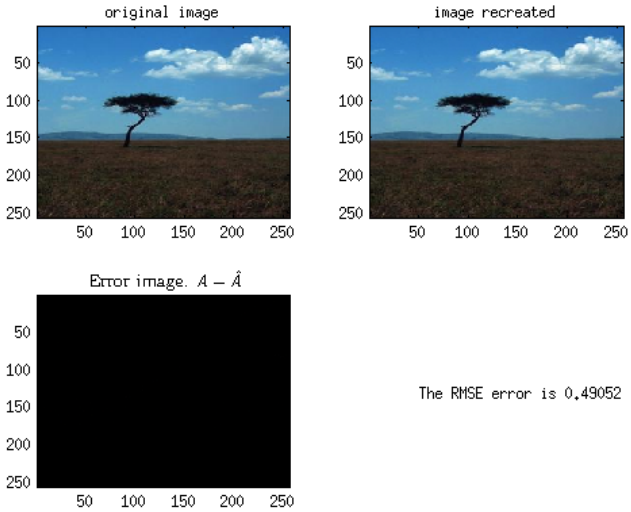
```
choosenSVs = [1: N];
```

i. choosenSVs = all SVs



No data is lost and no dimension is reduced.

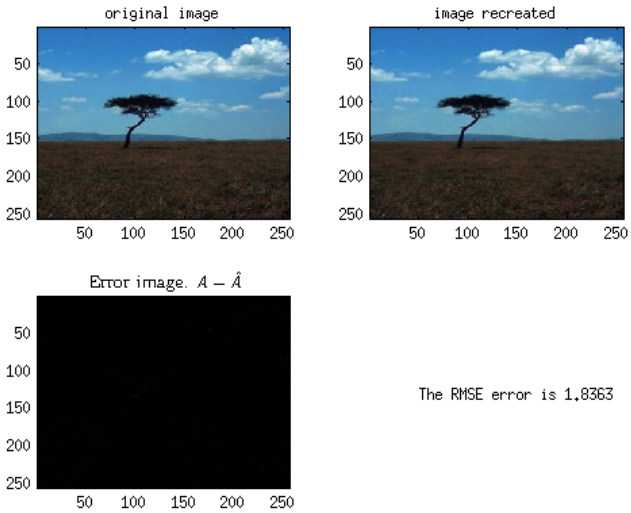
ii. choosenSVs = half SVs



data is lost by an average of RMSE at every pixel for every channel.

iii. choosenSVs are so that 90% of 'Power' is captured.

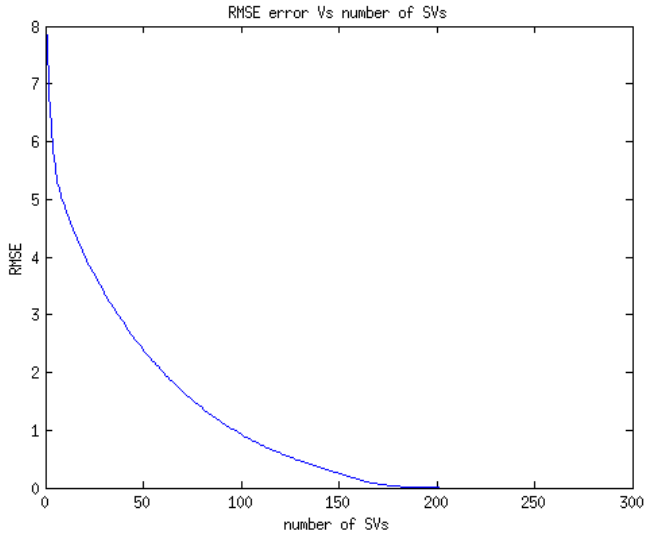
```
% choosen singular values
i = 1;
Rpower = sum(diag(S_R(1,1)))/sum(diag(S_R));
Gpower = sum(diag(S_G(1,1)))/sum(diag(S_G));
Bpower = sum(diag(S_B(1,1)))/sum(diag(S_B));
while ((Rpower+Gpower+Bpower)/3 < 0.9)
    i = i+1;
    Rpower = sum(diag(S_R(1:i, 1:i)))/sum(diag(S_R));
    Gpower = sum(diag(S_G(1:i, 1:i)))/sum(diag(S_G));
    Bpower = sum(diag(S_B(1:i, 1:i)))/sum(diag(S_B));
end
choosenSVs = [1: i];
clear('i', 'Rpower', 'Gpower', 'Bpower');
```



First 65 SVs got selected with this criteria.

iv. Now we plot RMSE Vs number of SVs (this operation will only be done if image size is small because of computational requirements)

```
for i=1:size(S_R)
choosenSVs = [1: i];
.
%====> same code as above <=======%
.
end

% plot RMSE Vs number of SVs
figure;
plot(choosenSVs, RMSE);
title('RMSE error Vs number of SVs')
set(get(gca,'XLabel'),'String','number of SVs');
set(get(gca,'YLabel'),'String','RMSE');
```

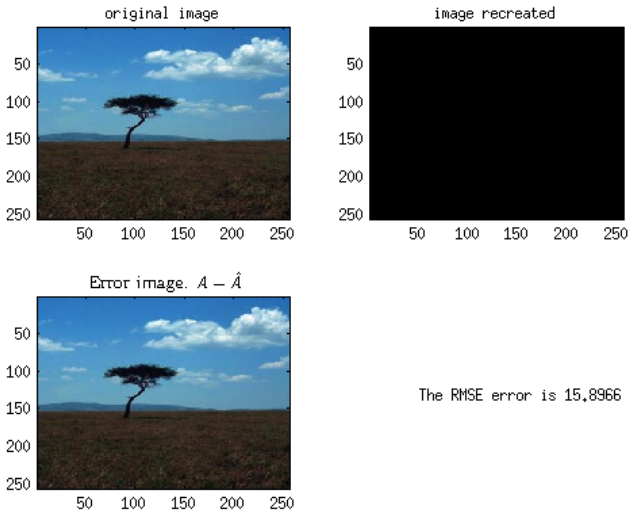


It is clear that error produced for chosen SVs greater than 180 is very minimal. so these dimensions can be omitted without much data loss. This reduces the dimensions / compress image without loss of much data.

(c) Chosen Singular values are 'Bottom N values'.

```
choosenSVs = [size(S_R,1) - N: size(S_R,1)];
```

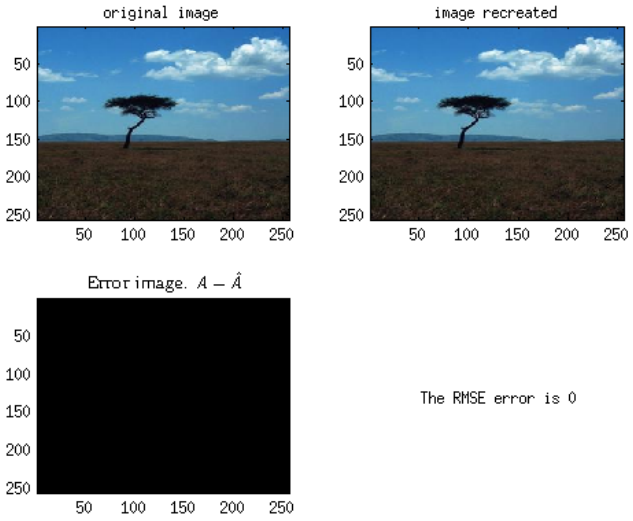
i. choosenSVs = half SVs



data is lost by an average of RMSE at every pixel for every channel. As you can see taking top half and bottom half is causes big error. Now the error image looks like the actual image; since most of the 'power' lies in first half SVs.

ii. choosenSVs are so that 90% of 'Power' is captured.

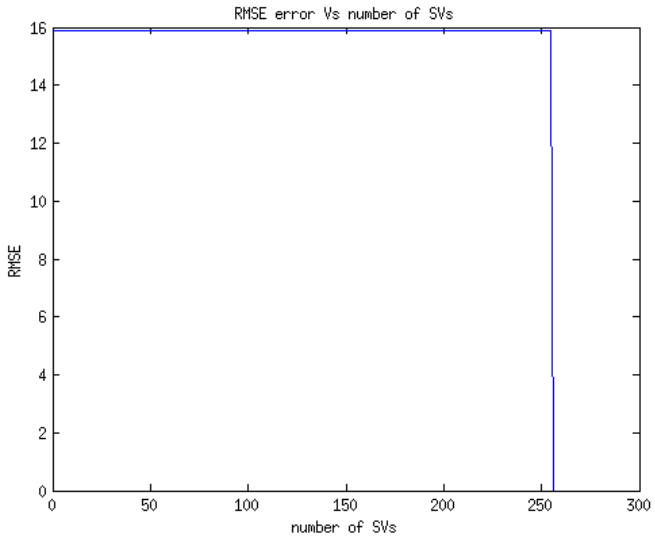
```
% choosen singular values
i = size(S_R,1);
Rpower = sum(diag(S_R(i,i)))/sum(diag(S_R));
Gpower = sum(diag(S_G(i,i)))/sum(diag(S_G));
Bpower = sum(diag(S_B(i,i)))/sum(diag(S_B));
while ((Rpower+Gpower+Bpower)/3 < 0.9)
    i = i-1;
    Rpower = sum(diag(S_R(i:end, i:end)))/sum(diag(S_R));
    Gpower = sum(diag(S_G(i:end, i:end)))/sum(diag(S_G));
    Bpower = sum(diag(S_B(i:end, i:end)))/sum(diag(S_B));
end
choosenSVs = [i:size(S_R,1)];
clear('i', 'Rpower', 'Gpower', 'Bpower');
```



To capture 90% of 'power', it had to use all 256 SVs; that means first SV itself is covering more than 10% 'power'.

iii. Now we plot RMSE Vs number of SVs chosen from bottom.(this operation will only be done if image size is small because of computational requirements)

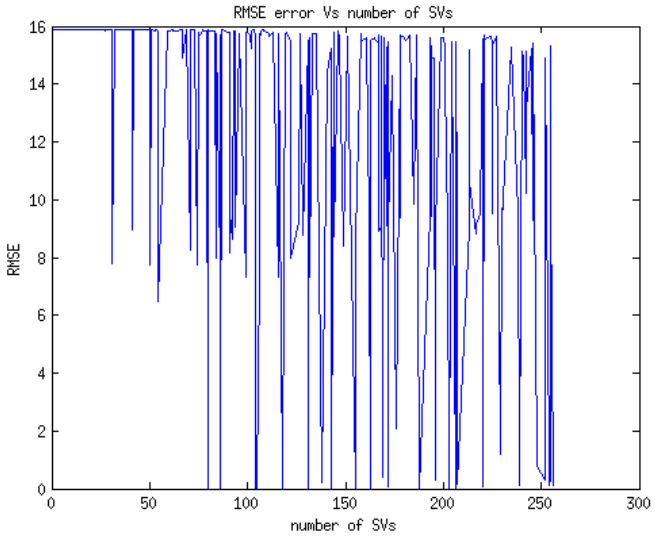
```
for i=1:size(S_R,1)
choosenSVs = size(S_R,1) + 1 -i: size(S_R,1);
.
%====> same code as above <=======%
.
end

% plot RMSE Vs number of SVs
figure;
plot(choosenSVs, RMSE);
title('RMSE error Vs number of SVs')
set(get(gca,'XLabel'),'String','number of SVs');
set(get(gca,'YLabel'),'String','RMSE');
```



It is clear that 'power' carried by bottom SVs are very less compared to top.

(d) Chosen Singular values are 'Random N values'. Loop N from 1 to 256 and plot RMSE., but the plot could be totally different on the next execution since its pseudo random selection.

```
choosenSVs = sort(randi(size(S_R, 1), 1, N));
```



2. Transforming into single matrix instead of operating on seperate channels.
   In order to have a unique value for each pixel color combination, you'll need 256^3 (16.8million) values. These values will have to be stored in floating point or uint32. You'll then have to make a colormap of [16.8million] floating point values to represent the colors you want.
   Then this indexed matrix can be used for svd analysis/ eigen decomposition. After the recreation of this matrix after choosing particular SVs/ eigenvalues, we can use the color map to decode the matrix into its R, G and B elements. this can be done with matlab command rgb2ind().
   code for creating this matrix is given:

```
% singleMat is the matrix where image is stored in 2D, map in the color map. and it
    can map into maximum of 65536 colors
[singleMat, map] = rgb2ind(oriImg, 65536);
```

   Now we can recreate this matrix and and come back to RGB matrix using matlab function ind2rgb() which will do the inverse color mapping.
   The rest of svd on this matrix is similar to any other matrix. so code for that is not included in the report.

2 Eigen Value Decomposition

Let A be a square $(N \times N)$ matrix with N linearly independent eigenvectors, q_i $(i = 1, \ldots, N)$. Then A can be factorized as

$$\mathbf{A} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{-1} \tag{5}$$

where Q is the square $(N \times N)$ matrix whose ith column is the eigenvector q_i of A and Λ is the diagonal matrix whose diagonal elements are the corresponding eigenvalues, i.e., $\Lambda_{ii} = \lambda_i$.
Also, we can reduce the dimensions like in svd here also. By reducing the number of eigenvalues selected, we can reduce the dimensions.

$$\hat{\mathbf{A}} = \mathbf{Q}\hat{\mathbf{\Lambda}}\mathbf{Q}^{-1} \tag{6}$$

And the error image can be found by $A - \hat{A}$.
Since A has to be square, we will include two case; One if image is square and another for rectangular images.

```
clear all
inputImage = '29.jpg';
oriImg = imread(inputImage);

figure;
subplot(2,2,1);
image(oriImg);
title('original image');
imageInfo = imfinfo(inputImage);
imgRChannel = im2double(oriImg(:,:,1));          % extract R channel
imgGChannel = im2double(oriImg(:,:,2));          % extract G channel
imgBChannel = im2double(oriImg(:,:,3));          % extract B channel
if (imageInfo.Width == imageInfo.Height)   % check if the image is square
    % ========> Process Square images <===============
else                                       % Rectangular
    % ========> Process Rectangular images <===============
end
```

1. Square Images
   Eigen Decomposition can be done by:

   ```
   [V_R, D_R] = eig(imgRChannel);
   [V_G, D_G] = eig(imgGChannel);
   [V_B, D_B] = eig(imgBChannel);
   ```

   So that $A = VDV^{-1}$.

2. Rectangular images
   For a rectangular matrix $A_{m\times n}$, Since eigen value decomposition cannot be done directly, we can do on A^TA or AA^T. If, $A = USV^T$, eigenvectors of AA^T make up the columns of U. Similarly A^TA makes up the columns of V.

   $$A^TA = (V\Sigma U^T)(U\Sigma V^T) = V\Sigma^2V^T \tag{7}$$
   $$AA^T = (U\Sigma V^T)(V\Sigma U^T) = U\Sigma^2U^T \tag{8}$$

   so for rectangular image, we can use svd on A^TA and AA^T to get eigen values and eigen vectors:

   ```
   else                                          % Rectangular
     [V_R, S_R, K] = svd(imgRChannel'*imgRChannel); clear('K');
     [U_R, S_R, K] = svd(imgRChannel*imgRChannel'); clear('K');

     [V_G, S_G, K] = svd(imgGChannel'*imgGChannel); clear('K');
     [U_G, S_G, K] = svd(imgGChannel*imgGChannel'); clear('K');

     [V_B, S_B, K] = svd(imgBChannel'*imgBChannel); clear('K');
     [U_B, S_B, K] = svd(imgBChannel*imgBChannel'); clear('K');
   end
   ```

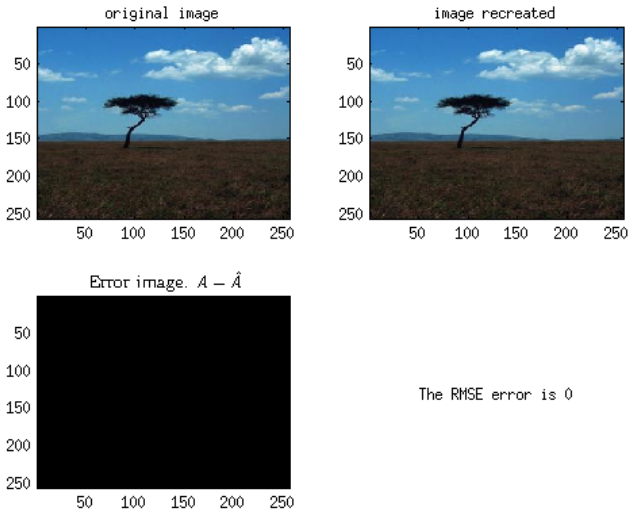
   From here on, the process of selecting eigenvalues and recreating matrix is done in same way as a square matrix.

And after selecting the required eigenvalues, we can recreate and plot the images as before with:

```
% Now similar to svd part, iterate the eigenvalues that you need to choose.
choosenSVs = 1:size(D_R, 1);

D_Rhat = D_R(choosenSVs, choosenSVs); %selected singular vectors
V_Rinv = inv(V_R);
imgRhat = 255*V_R(:, choosenSVs)*D_Rhat*V_Rinv(choosenSVs, :); %recreate R channel

D_Ghat = D_G(choosenSVs, choosenSVs); %selected singular vectors
V_Ginv = inv(V_G);
imgGhat = 255*V_G(:, choosenSVs)*D_Ghat*V_Ginv(choosenSVs, :); %recreate R channel

D_Bhat = D_B(choosenSVs, choosenSVs); %selected singular vectors
V_Binv = inv(V_B);
imgBhat = 255*V_B(:, choosenSVs)*D_Bhat*V_Binv(choosenSVs, :); %recreate R channel

imgHat = uint8(real(imgRhat));
imgHat(:,:,2) = uint8(real(imgGhat)); imgHat(:,:,3) = uint8(real(imgBhat));
RMSE = rms(rms(rms(oriImg - imgHat)));

subplot(2, 2,2);
image(imgHat);
title('image recreated')
subplot(2, 2, 3);
image(oriImg-imgHat);
title('Error image. $$A - \hat{A}$$', 'Interpreter','Latex')
tsp = subplot(2,2,4);
text(0.2,0.5,['The RMSE error is ', num2str(RMSE)]);
set ( tsp, 'visible', 'off')
```

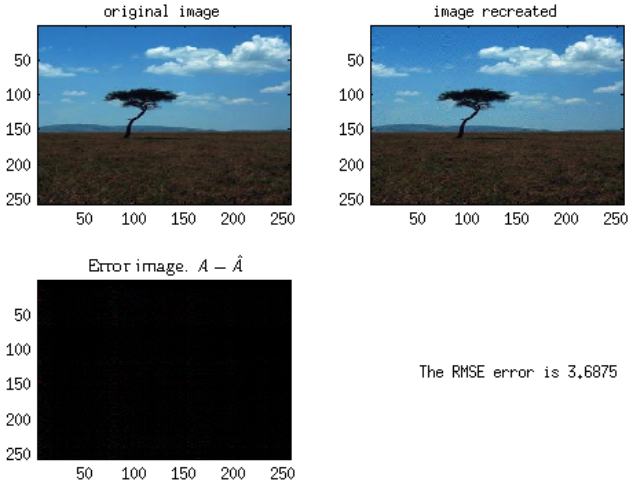
Now, we can play around with choosenSVs and make observations.

1. choosenSVs = all SVs



   No data is lost and no dimension is reduced.

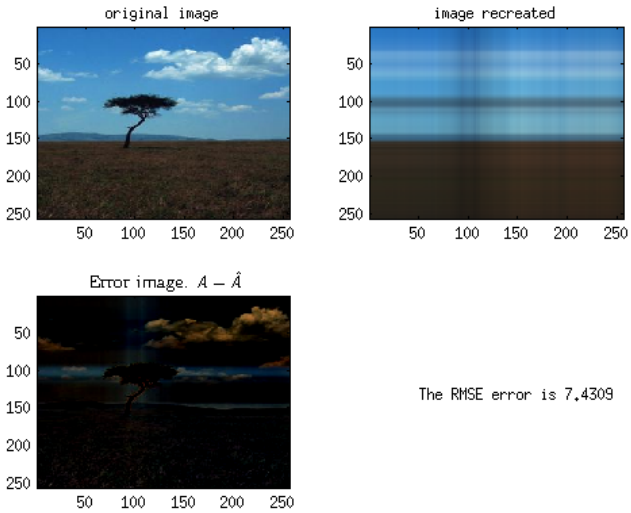
2. choosenSVs = half SVs



data is lost by an average of RMSE at every pixel for every channel.

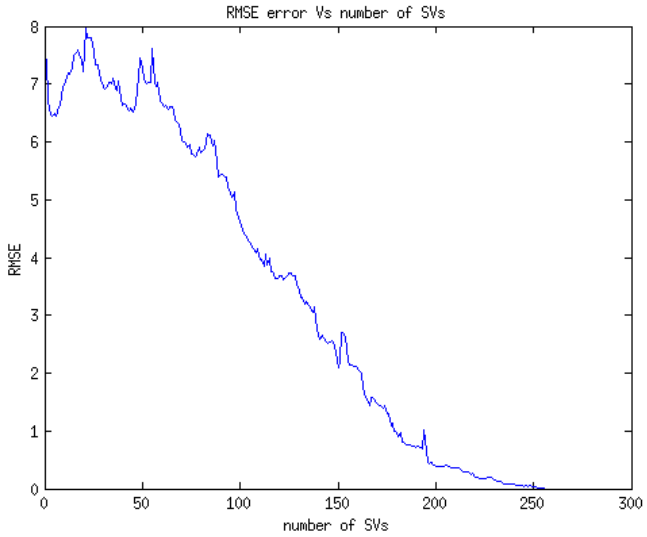
3. choosenSVs are so that 90% of 'Power' is captured.

```
i = 1;
Rpower = sum(diag(D_R(1,1)))/sum(diag(D_R));
Gpower = sum(diag(D_G(1,1)))/sum(diag(D_G));
Bpower = sum(diag(D_B(1,1)))/sum(diag(D_B));
while ((Rpower+Gpower+Bpower)/3 < 0.9)
    i = i+1;
    Rpower = sum(diag(D_R(1:i, 1:i)))/sum(diag(D_R));
    Gpower = sum(diag(D_G(1:i, 1:i)))/sum(diag(D_G));
    Bpower = sum(diag(D_B(1:i, 1:i)))/sum(diag(D_B));
end
choosenSVs = [1: i];
clear('i', 'Rpower', 'Gpower', 'Bpower');
```



4. Now we plot RMSE Vs number of eigenvalues (this operation will only be done if image size is small because of computational requirements)

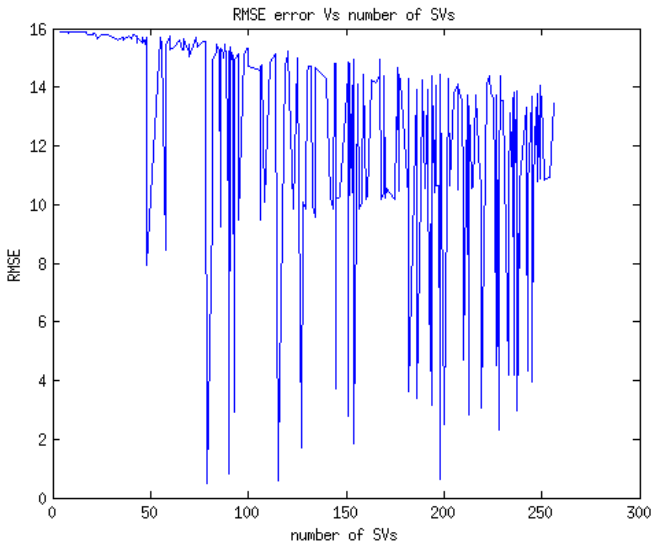
```
for i=1:size(D_R)
choosenSVs = [1: i];
.
%====> same code as above <=======%
.
end

% plot RMSE Vs number of SVs
figure;
plot(choosenSVs, RMSE);
title('RMSE error Vs number of SVs')
set(get(gca,'XLabel'),'String','number of SVs');
set(get(gca,'YLabel'),'String','RMSE');
```



The abnormal increase and decrease are caused by complex eigenvalues. Though the general trend is visible. As the number of eigenvalues increase, the error decrease.

5. Now we choose Random N eigenvalues and plot the error as we increase N:

```
choosenSVs = sort(randi(size(D_R, 1), 1, N));
```



3 Linear Regression

we decide to approximate y as a linear function of x:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 \tag{9}$$

Here, the θ_i 's are the parameters (also called weights) parameterizing the space of linear functions. According to least squares, we can show that Cost function reaches minimum at:

$$\theta = (X^T X)^{-1} X^T y \tag{10}$$

For training and validation we implement two famous methods:

1. Leave-p-out cross-validation
   Here we use $\frac{2}{3}$ of data for training and rest for validating model. best model is choosen for least RMS Error.

```
% choose data input
clear all
inputFile = 'Data_2_r0_s29.txt';

inData = dlmread(inputFile, ' '); % delimiter is space.
X = [ones(size(inData, 1), 1) inData(:, 1:size(inData, 2)-1)];
y = inData(:, size(inData, 2));

% Now we have two evaluation scheme:

% 1. Leave-p-out cross-validation: take 2/3 rd data for training, 1/3rd for validation
% 2. Leave one sample out evaluation
for i=1:round(size(X,1)/3)
    trainIds = [i: (i-1) + round(2*size(X,1)/3)];
    valIds = setxor([1:size(X,1)], intersect(trainIds, [1:size(X,1)]));
    Xtrain = X(trainIds, :);
    yTrain = y(trainIds, :);
    Xval = X(valIds, :);
    yVal = y(valIds, :);

    theta(i, :) = transpose(inv(Xtrain'*Xtrain)*Xtrain'*yTrain);
    % Now evaluate for validation set
```

```
    yhat = Xval*theta(i,:)';
    rmse(i) = rms(yVal - yhat);
end
[c, index] = min(rmse); clear('c');
disp(['best model is for theta ',num2str(theta(index,:))]);
```

2. Leave one out cross-validation
We use one data point for validating and rest for training. this is used in case of dataset with less data points; like we have. Model is selected based on least error.

```
clear all
inputFile = 'Data_2_r0_s29.txt';

inData = dlmread(inputFile, ' '); % delimiter is space.
X = [ones(size(inData, 1), 1) inData(:, 1:size(inData, 2)-1)];
y = inData(:, size(inData, 2));

% Leave one sample out evaluation

for i=1:size(X, 1)
    Xtrain = removerows(X, i);
    yTrain = removerows(y, i);
    Xval = X(i, :);
    yVal = y(i, :);

    theta2(i,:) = transpose(inv(Xtrain'*Xtrain)*Xtrain'*yTrain);
    yhat = Xval*theta2(i, :)';
    rmse(i) = rms(yVal - yhat);
end

% Minimum error; i.e best model is obtained for least rmse.
[c, index] = min(rmse); clear('c');
disp(['best model is for theta ',num2str(theta2(index,:))]);
```