**IMAGE COMPRESSION**

A PROJECT REPORT

*Submitted by*

ABHISHEK AGRAWAL

(09BCE327)

VIBHOR MISHRA

(09BCE194)

ANSUL GUPTA

(09BCE208)

*in partial fulfillment for the award of the degree*

*of*

**Bachelor of Technology (B.Tech.)**

IN

**Computer Science and Engineering (CSE)**

**VIT**

UN I V E R S I T Y

(Estd. u/s 3 of UGC Act 1956)

Vellore - 632 014, Tamil Nadu, India

VIT UNIVERSITY

November, 2012

**VIT**

U N I V E R S I T Y

(Estd. u/s 3 of UGC Act 1956)

Vellore - 632 014, Tamil Nadu, India

**School of Computing Science & Engineering**

**BONAFIDE CERTIFICATE**

This is to certify that the Mini project report entitled “**IMAGE COMPRESSION”** submitted by **ABHISHEK AGRAWAL**, **VIBHOR MISHRA** and **ANSUL GUPTA** to Vellore Institute of Technology, Vellore in partial fulfillment of the requirement for the award of the degree of B.TECH in Computer Science and Engineering (CSE) is a record of bonafide Mini project undertaken by us under my supervision. The project fulfills the requirements as per the regulations of this Institute and in my opinion meets the necessary standards for submission. The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Prof. Gayathri P.

**SUPERVISOR**

ASSISSTANT PROFESSOR (SENIOR)

COMPUTER SCIENCE AND ENGINEERING

VIT UNIVERSITY, Vellore – 632 014

Date: 2 November, 2012

**Internal Examiner (s) External Examiner (s)**

**ABSTRACT**

Our project ‘Image Compression’ performs image compression. For compressing the image we are using k-means clustering, bit-map and run-length encoding algorithm. As the clustering algorithm creates the clusters and causes repetition of the intensity values which run length encoding fully utilizes and compresses the image efficiently. The result of all above processes will lead to a compressed image. As k means is a lossy algorithm, after decompressing the image to recover the gradient as of original image we use interpolation.

**Table of contents**

|  |  |  |
| --- | --- | --- |
| **CHAPTER NO.** | **TITLE** | **PAGE NO.** |
|  | Abstract | 1 |
| 1. | List of figures | 3 |
| 2. | Introduction | 4 |
| 3. | Literature review | 5 |
| 4. | Design | 6-9 |
| 5. | Methodology | 10 |
| 6. | Implementation | 11-18 |
| 7. | Results | 19-20 |
| 8. | Conclusion | 21 |
| 9. | References | 22 |

**LIST OF FIGURES**

Figure 1(a)-(d) : Clustering Algorithm Explanation

Figure 2 : Bitmap Concept

Figure 3 : Interpolation Concept

Figure 4 : Left Arrow image

Figure 5 : LCD Display Image

Figure 6 : Game Ranger Icon

Figure 7 : Snake image

Figure 8 : Torrent image

**INTRODUCTION**

In the digital imaging environment, one of the central means of addressing this particular challenge is the use of image data compression. The purpose of compression is to reduce the size of the image with the goal of decreasing transmission time and reducing storage space. To compress the image we are using k-means clustering, bit-map and run-length encoding algorithm. Initially we have a taken an image of at most 256 colors (for lossless compression). Using k-means clustering algorithm, k clusters are formed by selecting k centroids and finding mean of the distances of individual points (here distance is measured using difference in the intensity values of pixels) from the centroids and grouping them together into a cluster. After repeating the same process again and again until same centroids are chosen in next iterations, we get the final cluster with their corresponding centroids value.

Using Bit-map algorithm, we create a file which act like a hash table for the k colors we get after applying clustering algorithm. After this, we use run length encoding method in which we will need to store only one intensity value with its frequency instead of storing all repeated intensity values. The result of all above processes will lead to a compressed image.

**LITERATURE REVIEW**

Images are very important documents nowadays; to work with them in some applications they need to be compressed, more or less depending on the purpose of the application. Many applications provide Graphical User Interface (GUI), which makes it easier to interpret e.g. internet download manager shows, the status of downloading a file graphically. Moreover, it is easier to understand things with the help of images. There are different types of images based on the pixel intensity and value. Firstly, a monochromatic image – the pixel can have only one of the two values black and white and represented by one bit only. Secondly, a grayscale image which can have 2N no of gray shades. The pixel can have 0 to N-1 values. Each pixel is represented by bytes and lastly, a continuous image where an image is represented by RGB pattern. There are some algorithms that perform this compression in different ways; some are lossless and keep the same information as the original image, some others lose information when compressing the image. Some of these compression methods are designed for specific kinds of images, so they will not be so good for other kinds of images. Some algorithms even let you change parameters they use to adjust the compression better to the image. Different algorithms for compression of gray scale images by maintaining the compression ratio and quality of an image. Fermat number transform (FNT) makes the compression to about 87.93% of JPEG image. FNT combines features of gray scale method and provides high performance.

**DESIGN**

# K-means Clustering

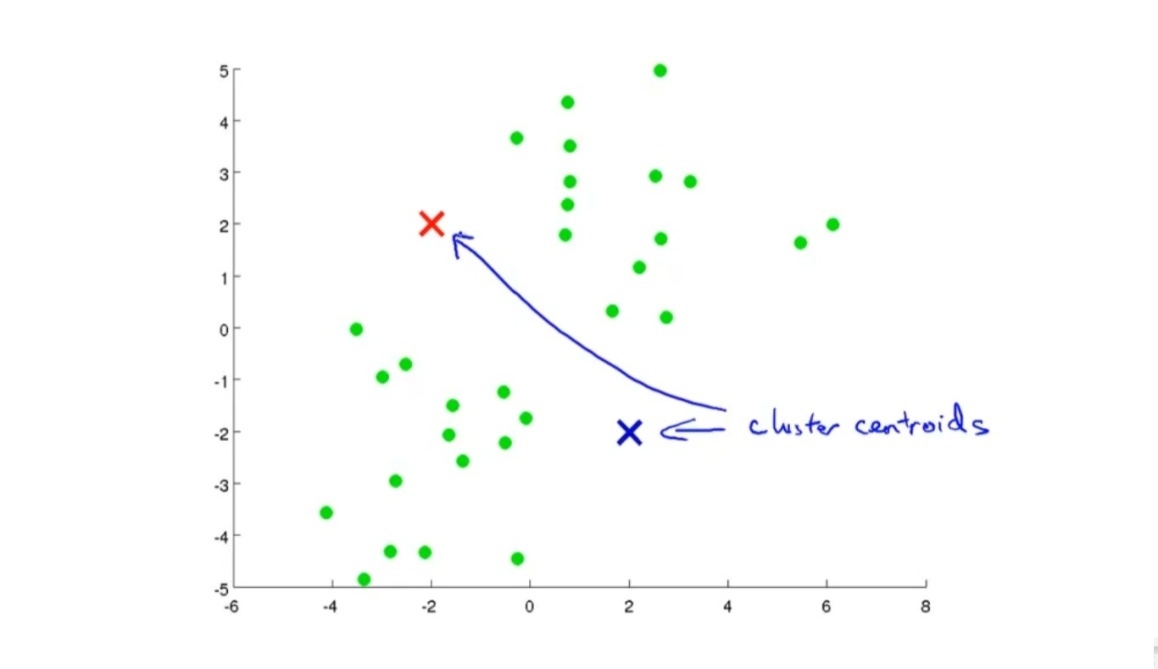
Initially, we select **2** centroids at random from among the given data points.

Fig: 1(a) – choose randomly 2 cluster centroids

Applying k – means for the 1st time to find the Euclidian distance of all the points from the 2 randomly chosen centroids

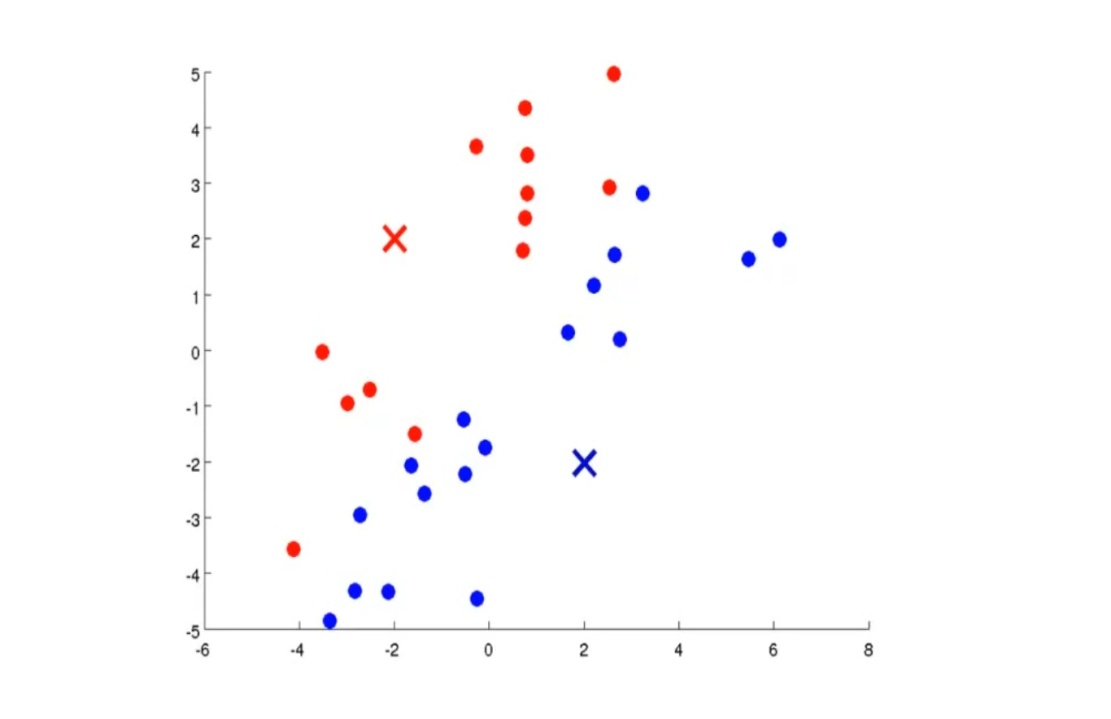
****

Fig: 1(b) – coloring each point based on Euclidean distance

Repeating the above process again and again (say 20 times)

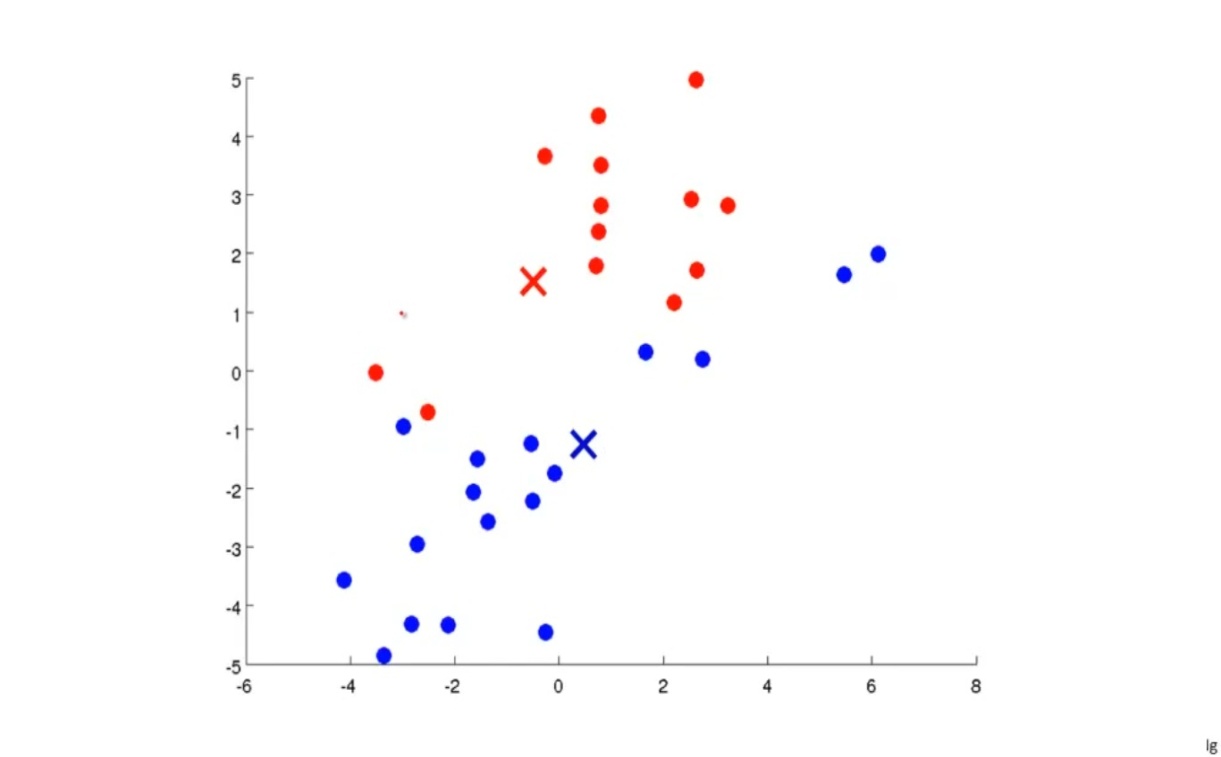
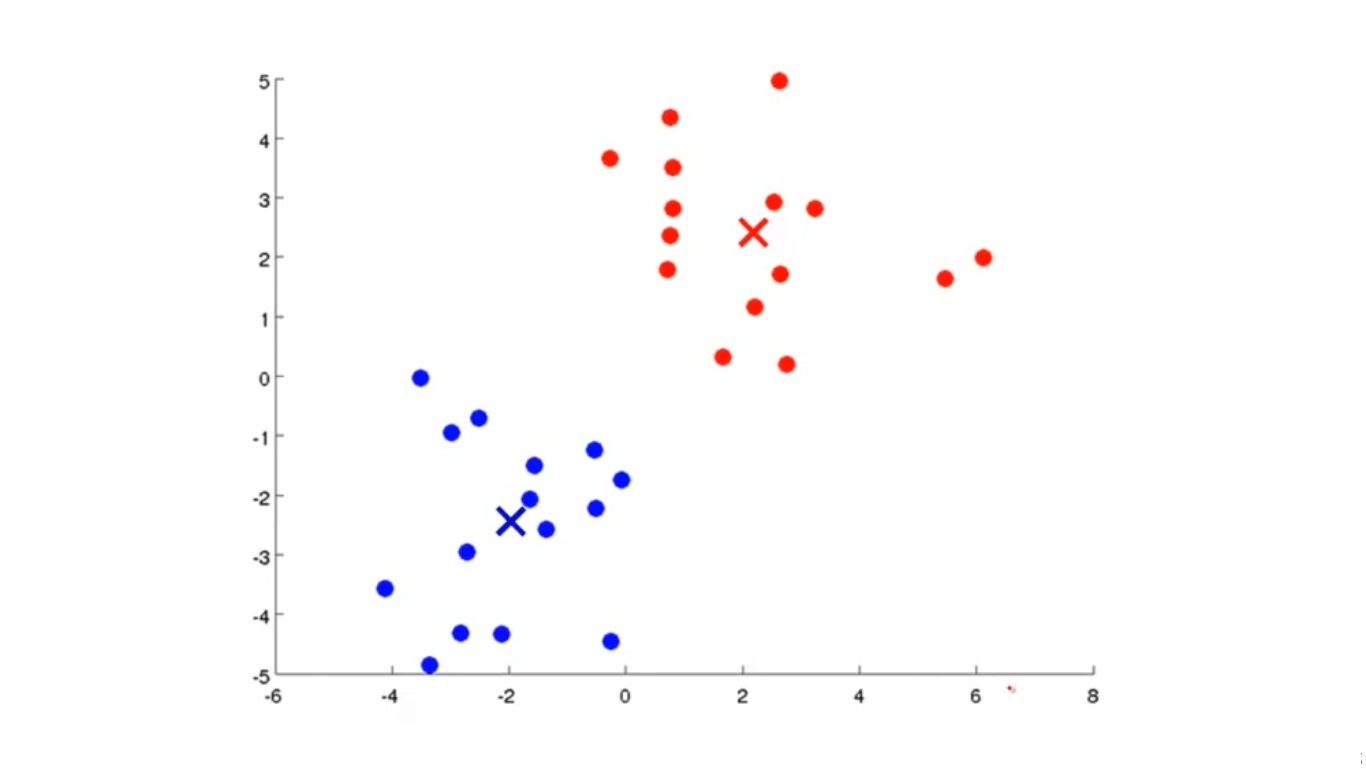
****

Fig: 1(c) – new centroids

We finally get 2 clusters:

****

## Fig: 1(d) – final clusters

## Bit-Map Generation

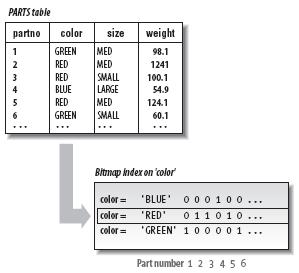
Once K-means algorithm yields **k** clusters, we store the information about which centroid corresponds to which color. Besides, we associate each centroid with a binary code. For **k** clusters, we require **log k** or (**log k**) **+ 1** bits to represent the code.

Fig 2:Applying bit map encoding

## Run Length Encoding

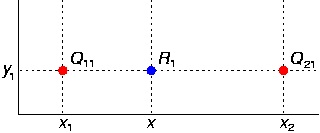
If we decide to use ,say, **m** bits per pixel in the compressed file then, we store the centroid code in leftmost **log k** bits and the remaining bits are used to store the number of times that color is repeated in the image. Thus, instead of dedicating **m** bits for each duplicate pixel, we can simply count the number of repetitions of that pixel color and store that value.

**Interpolation**

The interpolation method is used to smoothen the gradient in the image which is obtained by decompression of image. This is purely subjective task.

 f(R_1) \approx \frac{x_2-x}{x_2-x_1} f(Q_{11}) + \frac{x-x_1}{x_2-x_1} f(Q_{21})

whereR_1 = (x,y_1),



**Fig.3** applying interpolation

F(Q11), F(Q21) and F(R1) are the RGB intensity values.

**METHODOLOGY**

To compress the image we are using k-means clustering, bit-map and run-length encoding algorithm. Initially we have a taken an image of at most 256 colors (for lossless compression). Using k-means clustering algorithm, k clusters are formed by selecting k centroids and finding mean of the distances of individual points (here distance is measured using difference in the intensity values of pixels) from the centroids and grouping them together into a cluster. After repeating the same process again and again until same centroids are chosen in next iterations, we get the final cluster with their corresponding centroids value.

Using Bit-map algorithm, we create a file which act like a hash table for the k colors we get after applying clustering algorithm. After this, we use run length encoding method in which we will need to store only one intensity value with its frequency instead of storing all repeated intensity values. The result of all above processes will lead to a compressed image. As the clustering algorithm creates the clusters and causes repetition of the intensity values which run length encoding fully utilizes and compresses the image efficiently. The result of all above processes will lead to a compressed image. As k means is a lossy algorithm, after decompressing the image to recover the gradient as of original image we use interpolation.

**IMPLEMENTATION**

# **Main module : image\_cmp**

function image\_cmp(filename, type) %filename must not contain extension

clc;

fullname = strcat(filename,'.',type);

A = double(imread(fullname));

img\_size = size(A);

A = A ./ 255;

X = reshape(A, img\_size(1) \* img\_size(2), 3);

%X represents n-pixel image as a n\*3 matrix, the 3 columns representing

% Red, Green, Blue values respectively.

[K, candidates] = countCentroids(X);%K = number of centroids

max\_iters = 10;

if (K < 64)

centroids(1:K,:) = candidates(1:K,:);

else

shuffle = randperm(size(X,1));

centroids = X(shuffle(1:K), : );

end

% Run K-Means

[centroids, idx] = runkMeans(X, centroids, max\_iters);

X\_recovered = centroids(idx,:);

bitmapEncode2(filename, centroids, idx, img\_size);

% Reshape the recovered image into proper dimensions

X\_recovered = reshape(X\_recovered, img\_size(1), img\_size(2), 3);

% Display the original image

subplot(1, 2, 1);

imagesc(A);

title('Original');

% Display compressed image side by side

subplot(1, 2, 2);

imagesc(X\_recovered)

title(sprintf('Compressed, with %d colors.', K));

newname = strcat(filename,'\_cmp.',type);

imwrite(X\_recovered, newname,type);

end

## sub – module 1-1: countCentroids

function [K, temp] = countCentroids(X)

K = 0;

temp = zeros(64,3);

s = size(X,1);

for i=1:s

flag = 0;

for k = 1:K

if(sum( power( temp(k,:) - X(i,:), 2 ) ) == 0)

flag = 1;

break;

end

end

if (flag == 0)

K = K + 1; %count number of centroids required upto maximum of 64

temp(K,:) = X(i,:);

if (K == 64)

break;

end

end

end

fprintf('Total centroids : %d \n',K);

end

## **Sub – module 1-2: runkmeans**

function [centroids, idx] = runkMeans(X, initial\_centroids, max\_iters)

% Initialize values

[m n] = size(X);

K = size(initial\_centroids, 1);

centroids = initial\_centroids;

idx = zeros(m, 1);

% Run K-Means

for i=1:max\_iters

% Output progress

fprintf('K-Means iteration %d/%d...\n', i, max\_iters);

% For each example in X, assign it to the closest centroid

idx = findClosestCentroids(X, centroids);

% Given the memberships, compute new centroids

centroids = computeCentroids(X, idx, K);

end

% Now the centroids are finalized,so do final assignment of each example in X to closest centroid

idx = findClosestCentroids(X, centroids);

end

## **Sub – module 1-3: findClosestCentroids**

function idx = findClosestCentroids(X, centroids)

K = size(centroids, 1);

idx = zeros(size(X,1), 1);

m = size(X,1);

for i=1:m

dist = 10^6; %assign some random large value

for j=1:K

temp = sum( (X(i,:)-centroids(j,:)) .^ 2);

if(temp < dist)

dist = temp;

idx(i) = j;

end

end

end

end

## **Sub – module 1-4: computeCentroids**

function centroids = computeCentroids(X, idx, K)

[m n] = size(X);

centroids = zeros(K, n);

for j=1:K

[r,c]=find(idx==j);

centroids(j,:) = mean(X(r,:));

end

end

# **Module 2: bitmapEncode2**

function bitmapEncode2(filename, centroids, idx, img\_size)

[row, col] = size(idx);

newcentroids = uint8(centroids \* 255);

num\_cluster = size(centroids,1);

file2 = strcat(filename, '.txt');

fid = fopen(file2,'w+','l','UTF-8');

fprintf(fid,'%c%c%c',img\_size(1),img\_size(2),num\_cluster);

fprintf(fid,'%c',newcentroids);

i=1;

num\_pixel=img\_size(1)\*img\_size(2);

c=1;

centroids

while (i<=num\_pixel)

count=1;

for j=(i+1):row

if(idx(i)==idx(j))

count = count + 1;

else

break;

end

end

%fprintf('i = %2d, ( idx(i), count) = %2d %2d\n',i,idx(i),count);

asc(1,c) = char(idx(i));

asc(1,c+1) = char(count);

c = c+2;

i = j;

end

fprintf(fid,'%c',asc);

fclose(fid);

end

# **Module 3: decode**

function decode(filename)

close all;

%filename = 'C:\Users\Phalanx\Documents\Academics\MINI Project\v2.txt';

file2 = strcat(filename,'.txt');

fid = fopen(file2,'r+','l','UTF-8');

data = fscanf(fid,'%c');

data = double(data);

[m, n] = size(data);

encoded = zeros(1,2);

img\_size = zeros(1,2);

img\_size(1,1) = data(1,1);

img\_size(1,2) = data(1,2);

c = data(1,3);

cluster = zeros(c,3);

for j=1:3

for i=1:c

cluster(i,j) = data(1,(j-1)\*c+i+3);

end

end

i = (j-1)\*c + i + 4;

j=1;

while(i<n)

encoded(j,1) = data(1,i);

encoded(j,2) = data(1,i+1);

j = j+1;

i = i+2;

end

fclose(fid);

num\_pixels = img\_size(1)\*img\_size(2);

new\_matrix = zeros(num\_pixels,3);

i=1;

k=1;

while(k<=num\_pixels)

new\_matrix(k,1) = cluster(encoded(i,1),1);

new\_matrix(k,2) = cluster(encoded(i,1),2);

new\_matrix(k,3) = cluster(encoded(i,1),3);

for j=1:encoded(i,2)-1

k = k + 1;

new\_matrix(k,1) = cluster(encoded(i,1),1);

new\_matrix(k,2) = cluster(encoded(i,1),2);

new\_matrix(k,3) = cluster(encoded(i,1),3);

end

k = k+1;

i = i+1;

end

X\_recovered = reshape(new\_matrix, img\_size(1), img\_size(2), 3);

filter\_imag(X\_recovered);

end

## **Sub – module 3-1: filter\_imag**

function filter\_imag(image\_matrix)

h=3;

figure(h);

subplot(1,2,1);

imagesc(image\_matrix/255)

title('Compressed image');

img\_size = size(image\_matrix);

X = reshape(image\_matrix,img\_size(1)\*img\_size(2),3);

[r,c] = size(X);

s = 3;

s2 = 2;

hashed = zeros(1,60);

for i=1:r-s2

x1 = i;

x2 = i+s2;

d = sum((X(x1,:) - X(x2,:)).^ 2);

if( d >= 6000)

continue;

end

hashed(1+fix(d/100)) = hashed(1+fix(d/100)) + 1;

end

sumh = 0;

for i=5:60

sumh = sumh + hashed(i)\*i;

end

sumh = sumh / sum(hashed(5:60));

hashed

theta1 = fix(sumh \* 100)

if(hashed(1)>200000)

s = 6;

end

for i=1:r-s2\*img\_size(1)

x1 = i;

x2 = i+s2\*img\_size(1);

d = sum((X(x1,:) - X(x2,:)).^ 2);

if( d >= 6000)

continue;

end

hashed(1+fix(d/100)) = hashed(1+fix(d/100)) + 1;

end

sumh = 0;

for i=5:60

sumh = sumh + hashed(i)\*i;

end

sumh = sumh / sum(hashed(5:60));

hashed

theta2 = fix(sumh \* 100)

if(hashed(1)>200000)

s=6;

end

fprintf('s = %d\n',s)

if(s==6) % t = number of pixels to skip between two consecutive scans

t=s;

elseif(s==3)

t=1;

end

for outer = 1:2 % do repeated vertical and horizontal smoothing

for i=1:r-s\*img\_size(1) %horizontal smoothing

x1 = i;

x2 = i + s\*img\_size(1);

d = sum((X(x1,:) - X(x2,:)).^ 2);

d2 = sum((X(i,:) - X(i+fix(s\*img\_size(1)/3),:)).^ 2);

d3 = sum((X(i,:) - X(i+fix(2\*s\*img\_size(1)/3),:)).^ 2);

if((abs(d) < theta2) && (abs(d2) < theta2) && (abs(d3) < theta2))

for k = x1:img\_size(1):x2

%moving in steps of img\_size(1) is same as traversing pixel in a row

X(k,:) = ( X(x1,:)\*(x2-k) + X(x2,:)\*(k-x1) ) / (x2 - x1);

end

end

end

for i=1:t:r-s %vertical smoothing

x1 = i;

x2 = i+s;

d = sum((X(x1,:) - X(x2,:)).^ 2);

d2 = sum((X(x1,:) - X(i+fix(s/3),:)).^ 2);

d3 = sum((X(x1,:) - X(i+fix(2\*s/3),:)).^ 2);

if((abs(d) < theta1) && (abs(d2) < theta1) && (abs(d3) < theta1))

for k = x1:x2

%moving in steps of 1 is same as traversing pixel column wise

X(k,:) = ( X(x1,:)\*(x2-k) + X(x2,:)\*(k-x1) ) / (x2 - x1);

end

end

end

theta1 = 0.7 \* theta1;

theta2 = 0.7 \* theta2;

end

subplot(1,2,2);

image\_matrix = reshape(X, img\_size(1,1),img\_size(1,2),3);

imagesc(image\_matrix/255)

title('Recovered Image');

end

**RESULTS**

By applying image compression on various images we get a compression of around 30% for less detailed images and it performs a compression of around 75% for vibrant pictures.

Table: Comparing various results

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **File name** | **Original size** | **Dimensions** | **No. of clusters** | **Compressed image size** | **Percentage compressed** |
| Left-arrow.png | 25 kb | 256 X 256 | 64 | 16.4 kb | 34.4 |
| LCD.png | 59.8 kb | 256 X 256 | 64 | 13.3 kb | 77.8 |
| Batman.png | 9.37 kb | 256 X 256 | 64 | 5.34 kb | 43.0 |
| Snake.png | 48.6 kb | 394 X 399 | 64 | 25.1 kb | 48.3 |
| Torrent.png | 11.2 kb | 128 X 128 | 64 | 5.70 kb | 49.1 |

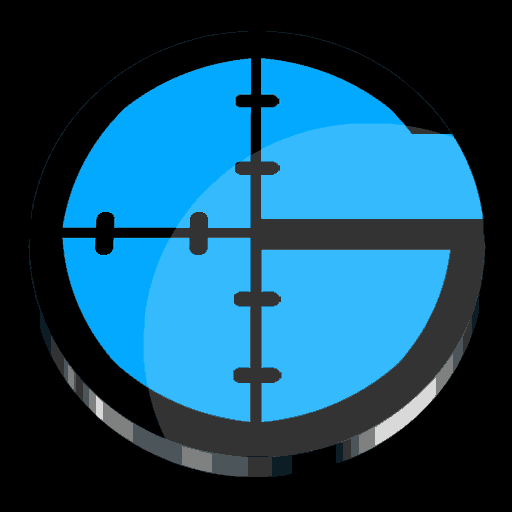
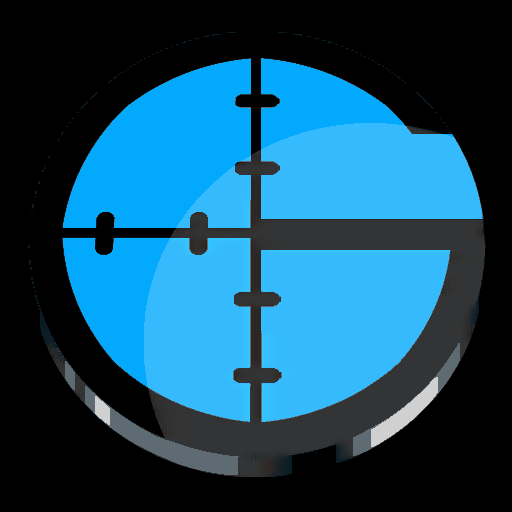
Original image After compression After Decompression

**Fig 4: Left Arrow.png**

**Fig 5: LCD Display.png**

**Fig 6: Game Ranger.png**

**  **

**Fig 7: Snake.png**

**Fig 8. Torrent.png**

**CONCLUSIONS AND FUTURE WORK**

The report presents a novel approach for image compression using k-means, bit map encoding and run length encoding. To conclude, the application of image compression is very vast. It is used in archiving, DVD etc. Moreover, an algorithm is chosen depending on the application.

We can further apply encoding in terms of rectangular blocks instead of line encoding. By applying block encoding, we will further compress the image as compared to line encoding.

**REFERENCES**

**Web References**

1. <http://www.mathworks.in/help/matlab/ref/fprintf.html>
2. <http://www.mathworks.in/help/matlab/ref/strcat.html>
3. <https://www.coursera.org/course/ml>

**Journal References**

1. Ayan Banerjee, Amiya Halder, “An efficient image compression algorithm for most dual color image based on k-means clustering, bit-map generation and RLE”

*International Conference on Computer and Communication Technology* [ICCT’10]

**Book References**

1. “Digital Image Processing” by Rafael C. Gonzalez and Richard E. Woods.
2. “Principles Of Soft Computing” by S. N. Sivanandam and S. N. Deepa.