Project 4 – Topics in Data Science ECE 592

Name: Alhiet Orbegoso
ID: 200322491

For the development of the project it has been used MATLAB. In the code there are 3 sections. The last section is used to solve sections 3, 4 and 5.

1. Load Image

For this project the image selected has a size of 2880x5120. To meet the requirements the image is cropped to the size of 2048x4096. Following is presented the original and the cropped image in gray scale color.

Original Image

Cropped Image

2. Clustering and Vector Quantization

For the solution of this section it has been created 2 functions:

- ***[V,C,err,p,it] = K_means(Elements, n_Clusters, delta, iter):***

  It is a custom implementation of the kmeans algorithm. The inputs are detailed below:

  Elements: It a matrix of elements to be classified. The columns represent the elements to be classified, and the rows the number of features (pixel) of each element. For example, if the patch size is 2, there are 4 pixels in total and 4 features to classify. The matrix will have the form:

	N=1	N=2	…	N=n
Feature 1 (Pixel 1):	123	55	…	102
Feature 2 (Pixel 2):	111	54	…	101
Feature 3 (Pixel 3):	123	65	…	123
Feature 4 (Pixel 4):	122	51	…	111

Matrix Elements

  N_Clusters: It is the number of clusters. Must be positive non-zero integer.

  delta: It is a stop condition of the algorithm. If the absolute difference of 2 consecutive clusters is below this threshold the algorithm will stop. If the clusters are multidimensional, the Euclidean norm is used for calculating the difference.

$$if\ (norm(C_k - C_{k-1}) < delta)\ \rightarrow STOP$$

  Iter: It is a stop condition of the algorithm. If the number of iterations is above this value, algorithm will stop. It works together with ***delta***, whichever reach the condition, algorithm stops.

The outputs are detailed below:

  V: It is a vector 1xN that stores the cluster index assignment for each element in the ***Elements*** vector. A position in vector V is the same as the position in vector Elements. For example, if ***V(93)*** = k, means that the element in position 93 in ***Elements*** vector belongs to cluster index k.

  C: It the clusters vector. It has 1xk size, where k must be equal to ***n_Clusters***. Each element represents the center of the cluster that the algorithm has calculated.

  Err: It is the sum of all Euclidean distances of elements from one cluster to their center.

  It: It is the number of iterations executed by algorithm.

This function has been wrapped into another function to improve code reutilization. The function encapsulating ***k_means*** is ***img_compression*** which is detailed below:

- ***[img_out,V] = img_compression(img_in, P, R, Type):***

  It is a function that calculates the compressed version of the ***img_in*** image for a given patch and rate, the inputs are detailed below:

  Img_in: It is the input image to be compressed. Must be in gray scale color.

  P: It is the size of the patch. The total number of pixels is the square of P.

  R: It is the value of rate.

  Type: It is a flag to select which kmeans algorithm is used. If 0 is used the custom algorithm, if 1 is used the MATLAB built-in algorithm. If other value is entered will throw error.

The outputs are detailed below:

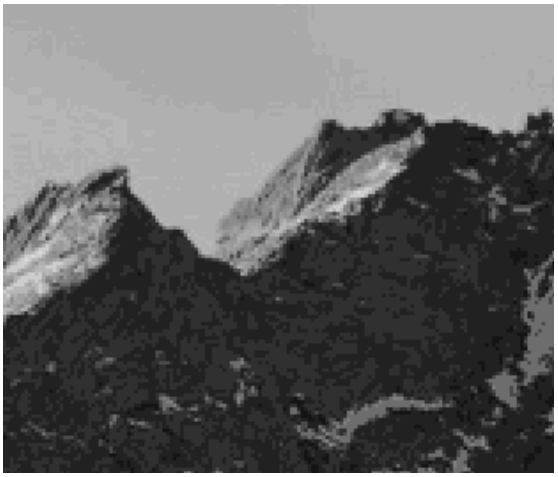
  Img_out: It is the compressed image in gray scale color.

  V: It ***V*** vector from k_means algorithm (custom or MATLAB built-in). It is useful for the entropy calculation.

For the solution of this section, the function ***img_compression*** is used with parameters R=1, P=2 and Type=0 (custom function). K_means parameters are delta=0.01, it=20. The resulting image is presented below:

Compressed Image P=2, R=1

Original



Compressed

Comments: The original image has a smooth transition between the clouds and mountain texture. The layers are almost invisibles, while in the compressed version can be observed the layers overlapping without any smooth transition. Given that there are only 16 vectors (limited alphabet) to reconstruct the image, the repeated vectors cannot recreate the smoothness of the original image.

3. Rate vs Distortion

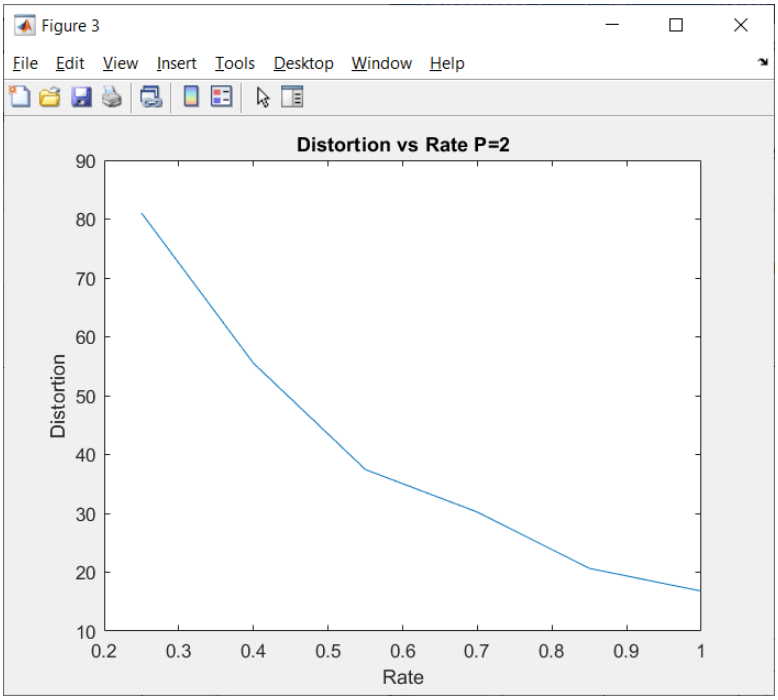
For this section is required to change the value of rate from a low value to 1 and patch size P=2. To generate to rate values to be tested is more convenient to generate a vector of N equally spaced values, the lowest values selected is R = 1/4, given that RxP^2=1 resulting in 2 clusters initially. The values of R tested for this section are the following:

$$R = [0.25,0.4,0.55,0.7,0.85,1]$$

Replacing in the equation 2^{RP^2} and rounding, number of clusters are:

$$clusters = [2,3,5,7,10,16]$$

Calculating distortion for all rates and P=2, it is obtained the following plot:



Distortion vs Rate

Comments: It is verified that the RD relationship is inversely proportional, if rate is low means less bits and thus better compression but distortion is high. By the other hand, bigger rate means more bits and less compression, but distortion is low.

4. Patch Size

For this section is required to change the value of rate from a low value to 1 but also the patch size. Here it is presented patch size {2,4,8,16}. Nevertheless, given that the num of cluster grows exponentially with the patch size, it is convenient to limit the rate for a given patch size:

First is calculated the number of elements for each given patch size. It is known that clusters must be less than the quantity of elements, it is not possible to classify a group of elements into a bigger group of clusters. To simplify this selection, it has been decided to take as maximum quantity of clusters the square root of the quantity of elements.

For a given P, the quantity of elements is given by image size divided by P^2 (quantity of pixels in patch). Then is taken the square root of the quantity of elements. This square root represents the maximum quantity of clusters per patch size.

$$\max clusters = \sqrt{\frac{MN}{P^2}} \dots (1)$$

The expression (1) is the maximum number of clusters for a given patch. This expression is also equal to the bits used to encode each patch, to get the maximum value of the rate for each patch, then is taken the base 2 logarithm of the maximum quantity of clusters, and then divide that result by the square of the patch size:

$$2^{R_{max}P^2} = \max clusters = \sqrt{\frac{MN}{P^2}}$$

$$R_{max} = \log_2\left(\sqrt{\frac{MN}{P^2}}\right).\frac{1}{P^2} \dots (2)$$

The expression (2) is the maximum rate for a given patch size P. Using this formula for the patches P = {2,4,8,16} then is obtained:

Patch	Max. Rate
2	1
4	0.5938
8	0.1328
16	0.0293

In the case of patch size 2, the maximum rate is 2.625 but it has been limited to 1. A rate bigger than 1 will indicate that patch can be represent with more than 16 bits. But given that each pixel can be represented with only 1 bit, 4bits for this patch size is the limit.

For the minimum rate calculation, it is convenient to start with 2 clusters. If is assumed 2 clusters, then for the minimum rate calculation the following expression is used:

$$2^{R_{min}P^2} = \min clusters = 2$$

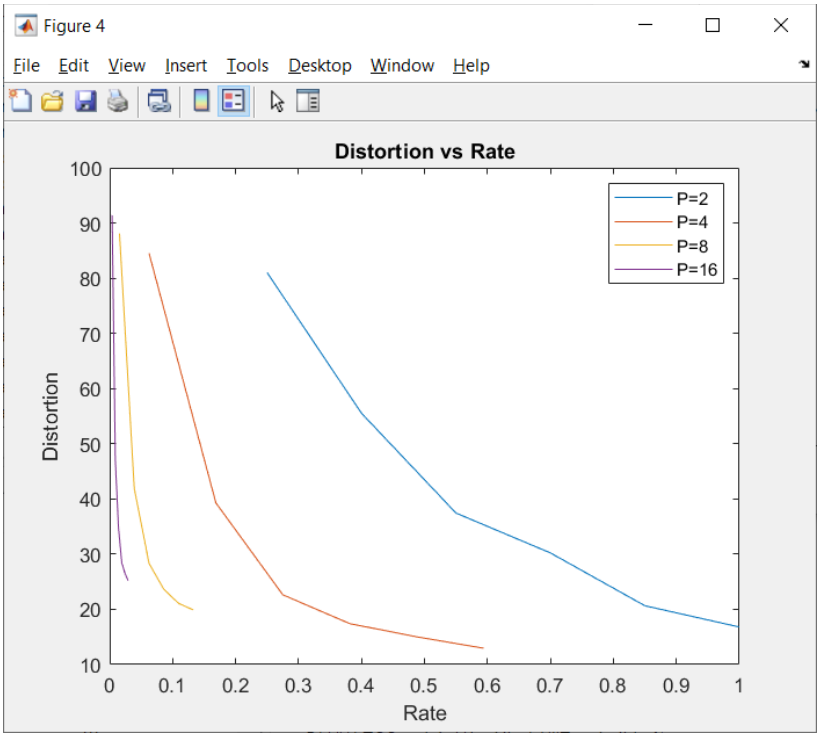
$$R_{min} = \frac{1}{P^2} \dots (3)$$

The expression (3) is the minimum rate for a given patch size P. Using this formula for the patches P = {2,4,8,16} then is obtained:

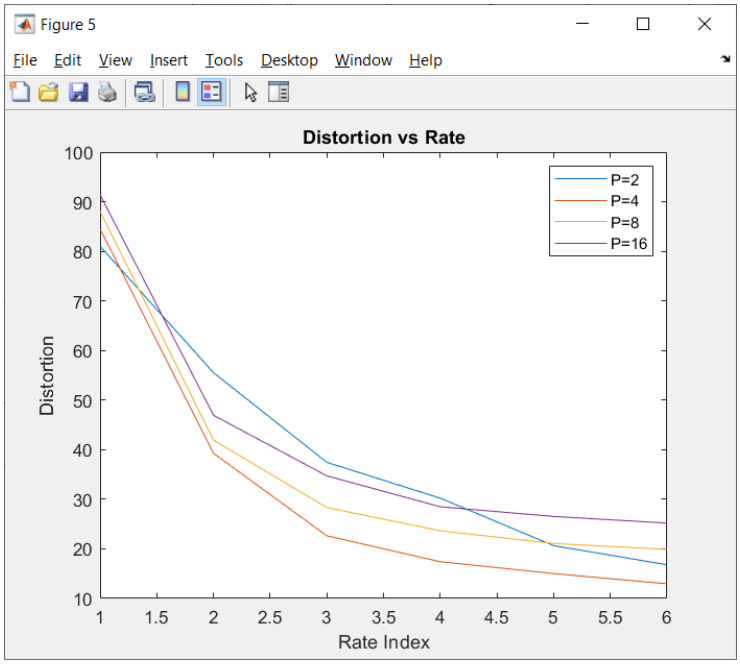
Patch	Max. Rate
2	0.25
4	0.0625
8	0.0156
16	0.0039

After calculating maximum and minimum rates for a given P, it is possible to form a vector containing equally spaced elements located inside the interval ***[min clusters, max clusters]***. Using the MATLAB function ***linspace()*** and N=6 elements the rate vectors are generated and consequently the number of clusters for each (P,R) pair.

Following is presented the plot obtained for each rate and patch size:



Distortion vs Rate for Patches (2,4,8,16)



Distortion vs Rate Index for Patches 2,4,8,16

In the plot the rate ranges are not the same for all patches due to the quantity of clusters restriction. This causes an “incomplete” plot for P = {4,8,16}. To provide a comparable result, below is presented a plot where the x axis is the rate index in vector (position of rate instead of value).

Comments: It observed that for all patches the distortion presents minimal values when the rate increases. Moreover the minimum distortion is obtained for the pair (P=4,R=0.5938).

For larger patch sizes the number of necessary bits to encode all possible vector increase exponentially. For small images to have a patch size larger than 4 is not recommendable since much lower rates would be necessary. For example, the image presented size is 2048x4096 (big image) and for a patch size 16 the maximum rate selected is 0.0293 (which is small compared with the patch size).

5. Better Compression

This section presents the calculation of normalized rates using the entropy formula presented in the project. It presented the normalized rate for all pairs R from the section 4.

First in presented the rates in section 4 for all pairs P = {2,4,8,16}. Then is presented a table with normalized rates.

	Min rate	n=2	n=3	n=4	n=5	Max rate
Pair =2	0.25	0.4	0.55	0.7	0.85	1
Pair = 4	0.0625	0.1688	0.275	0.3813	0.4875	0.5938
Pair = 8	0.0156	0.0391	0.0625	0.0859	0.1094	0.1328
Pair = 16	0.0039	0.009	0.0141	0.0191	0.0242	0.0293

Rates for all patches 2,4,8,16

	Min rate	Rate (2)	Rate (3)	Rate (4)	Rate (5)	Max rate
Pair =2	0.1899	0.3808	0.514	0.6205	0.7606	0.9178
Pair = 4	0.0478	0.1387	0.2225	0.3108	0.4118	0.5039
Pair = 8	0.0122	0.035	0.0504	0.0715	0.0917	0.1111
Pair = 16	0.0032	0.0076	0.0126	0.0158	0.0198	0.0241

Normalized rates for all patches 2,4,8,16

From the values in both tables is possible to obtain the reduction coding rate in percentage applying the following formula:

$$r(\%) = \left(1 - \frac{R_{norm}}{R}\right).100\% \ldots (4)$$

Applying formula 4 with the data from both tables:

	Min rate	Rate (2)	Rate (3)	Rate (4)	Rate (5)	Max rate
Pair =2	24.031%	4.789%	6.543%	11.352%	10.517%	8.223%
Pair = 4	23.445%	17.820%	19.103%	18.483%	15.520%	15.140%
Pair = 8	21.637%	10.411%	19.369%	16.831%	16.187%	16.313%
Pair = 16	19.276%	15.235%	10.470%	17.376%	18.372%	17.737%

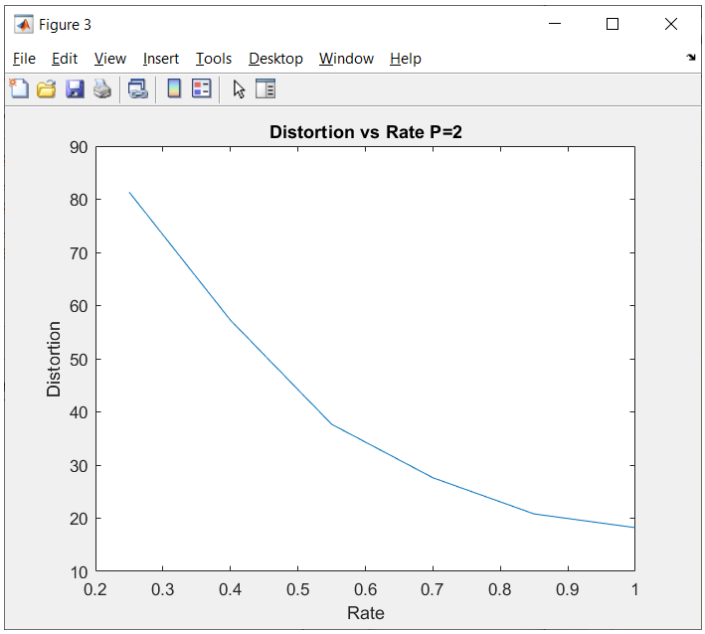
Reduction coding rate in %

Comments: Given that entropy is a disorder measurement, this value is more image dependent. For images with histograms near uniform distributions is expected high entropy and consequently the reduction is minimal. For the project case, this distribution is equivalent to how the final clusters are distributed. If all the clusters are equally spaced, it is expected an equally number of elements belonging to each of them. This will maximize the entropy value and the reduction would be near 0%. In this case entropy reduction is more than 10% for almost all cases.

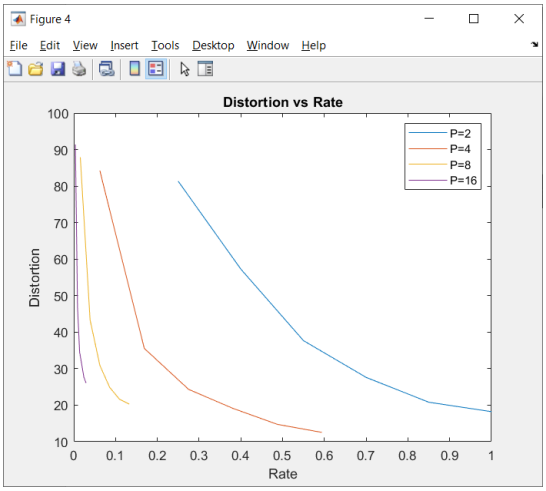
6. Kmeans Implementation

In section 2. Are explained the details of the ***k_means*** function implemented. Here are presented results with MATLAB ***kmeans*** function and are compared with implemented function. For this the argument ***Type*** of function ***img_compression()*** is set to ***1*** (MATLAB function):

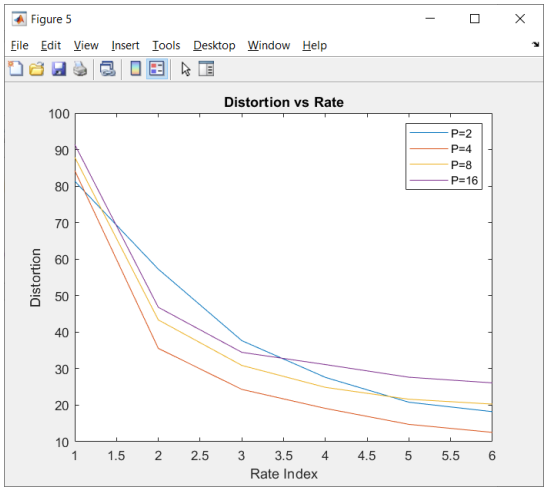
Quantized image P=2, R=1



Distortion vs Rate P=2



Distortion vs Rate for Patches (2,4,8,16)



Distortion vs Rate Index for Patches (2,4,8,16)

	Min rate	Rate (2)	Rate (3)	Rate (4)	Rate (5)	Max rate
Pair =2	0.1892	0.3779	0.5356	0.6745	0.7888	0.8566
Pair = 4	0.0478	0.1524	0.2157	0.2787	0.3714	0.4472
Pair = 8	0.0122	0.0351	0.0501	0.0652	0.0812	0.0937
Pair = 16	0.0032	0.0086	0.0121	0.0149	0.018	0.0202

Normalized Rates for all patches 2,4,8,16

	Min rate	Rate (2)	Rate (3)	Rate (4)	Rate (5)	Max rate
Pair =2	24.324%	5.513%	2.623%	3.637%	7.201%	14.344%
Pair = 4	23.495%	9.704%	21.546%	26.898%	23.825%	24.681%
Pair = 8	21.637%	10.112%	19.806%	24.082%	25.769%	29.447%
Pair = 16	19.330%	4.769%	13.786%	22.299%	25.785%	31.012%

Reduction coding rate in %

Finally, is provided the execution time for both implementations for 128 clusters and 25 iterations, the input matrix is in both cases the image processed in this project. This test is performed using the ***Kmeans_test*** script.

- Custom time: 329.78s

- MATLAB time: 59.258s

Comments: Both implementations present similar results. The same conclusions can be derived with kmeans MATLAB function calculations. The time execution is faster in MATLAB function than in custom function. MATLAB could take advantage of low-level implementation and optimization which are not accessible for users.

MATLAB Code:

Project4 Script

```
%% Project 4
%% --- 1) Load Image and resize
img_name = 'Landscape.png';
img = rgb2gray(imread(img_name)); % Convert to gray scale
M = 2048; % length M
N = 4096; % length N
img = img(1:M,1:N); % Image Resized
Type = 0; % Type kmeans used 0=custom, 1=MATLAB
%% --- 2) Clustering and vector quantization
R = 1; % Rate
P = 2; % Pixels per patch (must be power of 2)
img_Q = img_compression(img,P,R,Type);
figure(1)
imshow(img)
figure(2)
imshow(img_Q)
imwrite(img,'Landscape_Gray.png')
imwrite(img_Q,'Landscape_Q.png')
%% --- 3) Rate vs Distortion and 4) Patch Size and 5) Better Compression
P = [2,4,8,16]; % Patch {2,4,8,16}
Num = 6; % It is generated 5 rates per Patch
R = zeros(length(P),Num); % Rate matrix
D = zeros(length(P),Num); % Distortion matrix
R_norm = zeros(length(P),Num); % Normalized Rate matrix
% Generating rates for each patch P
maxR = log2(sqrt(M*N./P.^2))./P.^2; % Maximum rates
maxR(maxR>=1)=1; % No rate greater than 1
minR = ones(1,length(P))./P.^2; % Minimum rates
for i =1:length(P)
    R(i,:) = linspace(minR(i),maxR(i),Num);
end
% Distortion vs Rate and Normalized Rate calculation
for i = 1:length(P)
    for j=1:length(R)
        [img_Q ,V] = img_compression(img,P(i),R(i,j),Type);
        D(i,j) = sum((img_Q-img).^2,'all')/(M*N);
        %Entropy and Rate Calculation
        H = 0; % Entropy
        n_C = round(2^(R(i,j)*P(i)^2)); % Number of clusters
        for k=1:n_C
            p = sum(V==k)/length(V); % Probability
            if(p~=0) % Verifying if cluster exists
                H = H - p*log2(p); % Entropy
            end
        end
        R_norm(i,j) = H/P(i)^2; % Normalized Rate
        % Progress
        c = fix(clock);
        fprintf('Progress: %d of %d ',((i-1)*length(R)+j),length(R)*length(P));
        fprintf('Time: %d:%d:%d \n',c(4),c(5),c(6));
    end
    % Save img with highest Rate
    imwrite(img_Q,strcat('Landscape_Q_P_',num2str(P(i)),'.png'));
end
r_p = (1-R_norm./R)*100; % Reduction Coding Rate calculation (%)
%% PLOTS
% Plot Rate vs Distortion for P=2
figure(3)
plot(R(1,:),D(1,:))
title('Distortion vs Rate P=2')
xlabel('Rate')
ylabel('Distortion')
% Plot Rate vs Distortion for P=2,4,8,16
```

```
figure(4)
for i=1:length(P)
    plot(R(i,:),D(i,:))
    hold on
end
title('Distortion vs Rate')
xlabel('Rate')
ylabel('Distortion')
legend('P=2','P=4','P=8','P=16')
figure(5)
for i=1:length(P)
    plot((1:Num),D(i,:))
    hold on
end
title('Distortion vs Rate')
xlabel('Rate Index')
ylabel('Distortion')
legend('P=2','P=4','P=8','P=16')
```

Function img_compression()

```
function [img_out,V] = img_compression(img_in,P,R,Type)
    M = size(img_in,1);
    N = size(img_in,2);
    img_data = zeros(P^2,M*N/P^2); % Vector of elements
    img_out = zeros(M,N); % Image quantized
    % Partition Image into vector of elements for clustering algorithm
    for dy=1:N/P
        for dx=1:M/P
            sub_img = img_in(P*dx-P+1:P*dx,P*dy-P+1:P*dy);
            img_data(:,(dx+(dy-1)*M/P)) = reshape(sub_img,P^2,1);
        end
    end
    % Clustering Type=0 (Custom), Type=1 (MATLAB)
    if (Type==0)
        delta = 0.01; % Stop condition between to consecutive kernels
        iter = 20;
        [V,C]=k_means(img_data/255.0,round(2^(R*P^2)),delta,iter);
    elseif (Type==1)
        [V,C]=kmeans(img_data'/255.0,round(2^(R*P^2)));
        V = V';
        C = C';
    end
    % Image reconstruction
    img_vector = uint8(C(:,V)*255);
    % Image reconstruction from vector elements
    for dy=1:N/P
        for dx=1:M/P
            sub_vec = img_vector(:,(dx+(dy-1)*M/P));
            img_out(P*dx-P+1:P*dx,P*dy-P+1:P*dy)= reshape(sub_vec,P,P);
        end
    end
    img_out = uint8(img_out);
end
```

Function k_means()

```
function [V,C,err,it] = k_means(Elements,n_Clusters,delta,iter)
    %Variables
    N = size(Elements,2); % number of elements
    f = size(Elements,1); % number of features
    clusters = ones(f,1)*rand(1,n_Clusters); % Clusters initialization
    new_clusters = zeros(f,n_Clusters); % New Clusters initialization
    labels = (1:n_Clusters);
    label_E = zeros(1,N); % Vector of element labels
    diff_clusters = 10; % Initialization diff clusters
    it = 0; % Iterations counter
```

```
    % Number of iterations
    while ((diff_clusters>delta)&&(it<iter))
        % Label indexing for each element
        for element=1:N
            dist = Elements(:,element)-clusters;
            [~,min_index] = min(vecnorm(dist));
            label_E(element) = labels(min_index);
        end
        % Calculating new clusters
        err = zeros(1,n_Clusters); % Vector to save distances of elements to its
cluster
        p = zeros(1,n_Clusters); % Vector of probability of clusters
        for label=1:n_Clusters
            X = Elements(:,label_E==label);
            if (~isempty(X))
                new_clusters(:,label) = mean(X,2); % New clusters
                err(:,label) = sum(vecnorm(X - mean(X,2))); % Distance clusters and
labeled data
            end
        end
        % Error consecutive clusters
        diff = abs(clusters - new_clusters);
        diff = (diff(~isnan(diff))); % Verifying NaN clusters
        diff_clusters = norm(vecnorm(diff));
        % New clusters assigment
        clusters = new_clusters;
        it = it + 1; % Increase iteration counter
    end
    % Outputs
    V = label_E; % Vector of final elements labels
    C = clusters; % Vector of final clusters
end
```

Kmeans_test script

```
%% Kmeans Test
%% Variables
img_name = 'Landscape.png';
img = rgb2gray(imread(img_name)); % Convert to gray scale
M = 2048; % length M
N = 4096; % length N
P = 2; % Patch size
img = img(1:M,1:N); % Image Resized
img_data = zeros(P^2,M*N/P^2); % Vector of elements
%% Partition Image into vector of elements for clustering algorithm
for dy=1:N/P
    for dx=1:M/P
        sub_img = img(P*dx-P+1:P*dx,P*dy-P+1:P*dy);
        img_data(:,(dx+(dy-1)*M/P)) = reshape(sub_img,P^2,1);
    end
end
%% Custom Kmeans
delta = 0;
n_clusters = 128;
iter = 25;
tic
[idx1,C1,sum1] = k_means(img_data/255,n_clusters,delta,iter);
t_custom = toc;
%% MATLAB Kmeans
tic
[idx2,C2,sum2] = kmeans(img_data'/255,n_clusters,'Maxiter',iter);
t_matlab = toc;
```