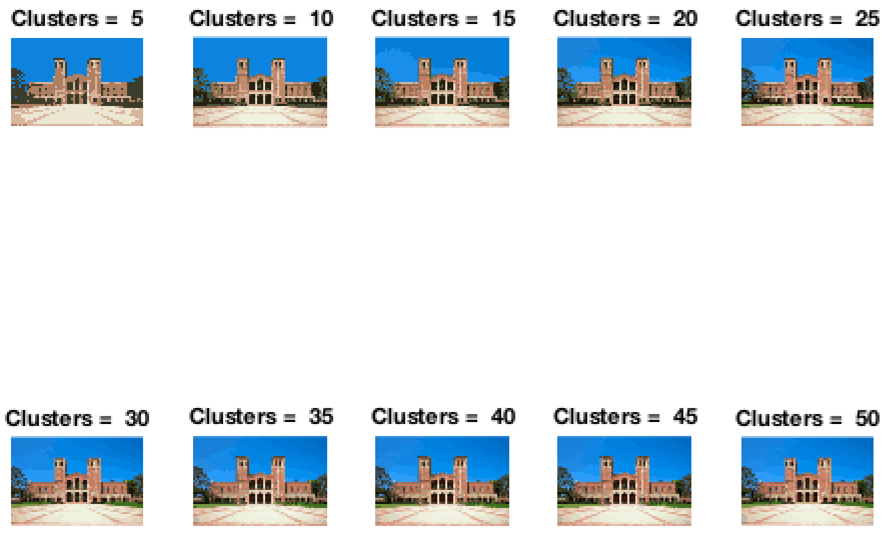
```
load royce_hall_small;
%Going to reshape our matrix to a 2-D matrix where the rows are pixels

img = zeros(96*144, 3);
for i = 1:3
    img(:, i) = reshape(I(:, :, i), 96*144,1);

end

figure
imshow(I);
title('royce hall');
```

```
%2.

segm = zeros(13824, 3);
segm2 = zeros(96, 144, 3);
figure
for j = 1:10
    [labels, means] = kmean(img, 5*j);
    for i = 1:13824
        for k = 1:5*j
            if labels(i) == k
                segm(i, :) = means(k,:);
            end
        end
    end
    for i = 1:3
        segm2(:, :, i) = reshape(segm(:, i), [96, 144, 1]);
    end

    subplot(2,5,j);
    imshow(segm2);
    title(['Clusters =  ' int2str(5*j)]);
end

%{
```

```
We see that at K = 5, we are able to discern the main components of
 our
image. As we increase the number of clusters, our image becomes more
 and
more clear. I would say a good range of values for K would be K = 5
 from K
= 10 since we are able to recover the most important features of our
 image
in that range.

%}
```

```
     5

    10

    15

    20

    25

    30

    35

    40

    45

    50
```



```
%4.
rng('default');
noise = zeros(13824, 3);
col_1 = sqrt(var(img(:,1))/100)*randn(13824, 1);

col_2 = sqrt(var(img(:, 2))/100)*randn(13824, 1);

col_3 = sqrt(var(img(:, 3))/100)*randn(13824, 1);
noise = horzcat(col_1, col_2, col_3);

pix_img = img + noise;

pix_segm = zeros(13824, 3);
pix_segm2 = zeros(96, 144, 3);
figure
for j = 1:5
    [labels, means] = kmean(pix_img, j);
    for i = 1:13824
        for k = 1:j
            if labels(i) == k
                pix_segm(i, :) = means(k,:);
            end
        end
    end
    for i = 1:3
        pix_segm2(:, :, i) = reshape(pix_segm(:, i), [96, 144, 1]);
    end

    subplot(1,5,j);
    imshow(pix_segm2);
    title(['Clusters =  ' int2str(j)]);
end

%{
We see that our K-means image reconstruction is more pixelated and
 less
accurate than before. This is because more points are misclassified
 due to
noise.


%}
```

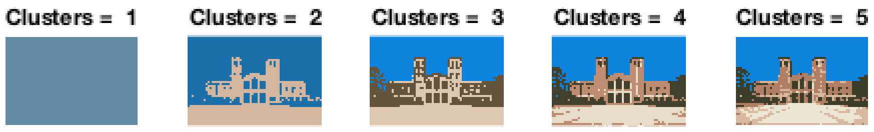
1

2

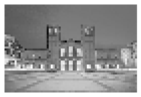
3

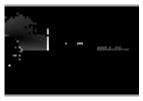
4

5

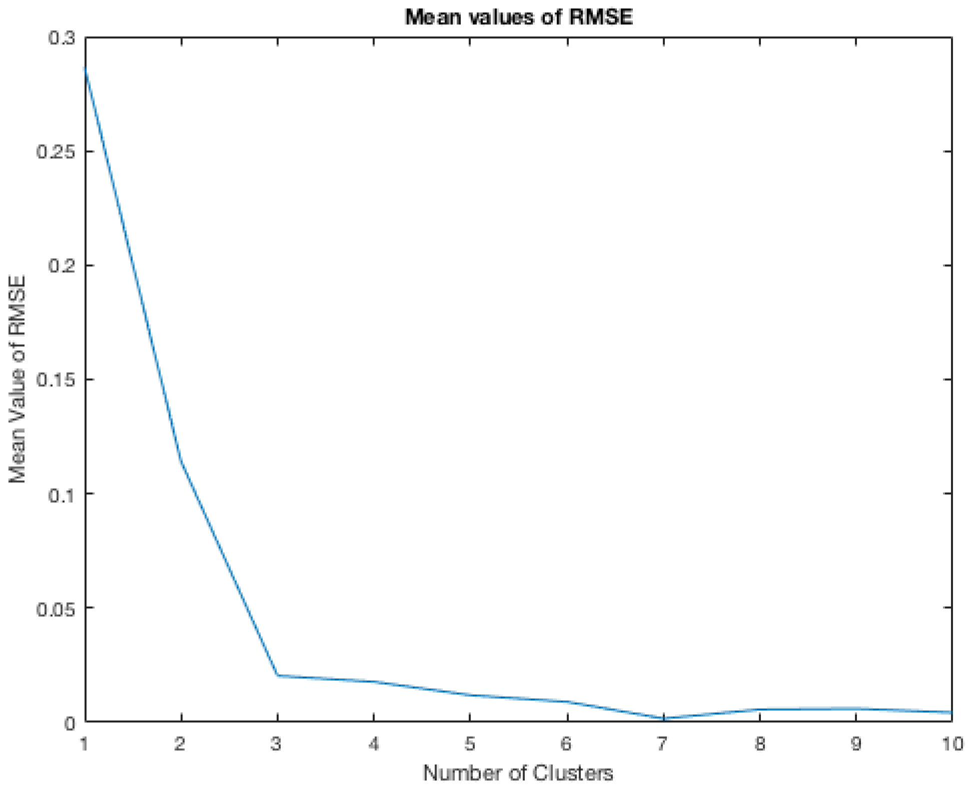


```
%6.
rmse = zeros(13824, 100);
num = 1;
figure
for k = 1:10
    [labels, means] = kmeans(img, k);
    for i = 1:13824
        for j = 1:k
            if labels(i) == k
                rmse(i, k) = 1/sqrt(3)*sqrt(sum((means(j, : ) -
 img(i, : )).^ 2));
            end
        end
    end
    if k == 1 || k == 10 || k == 40 || k == 70 || k == 100
        subplot(1, 5, num);
        num = num + 1;
        imshow(mat2gray(reshape(rmse(:, k), 96, 144)));
    end
end

%{
We see that our rmse images get more and more black as the
number of clusters increase because our approximation x_hat gets
 closer and
closer to our orig image x. Thus, we have lower rmse values as the
 number
of clusters increase. Lower values for RMSE mean that our image of the
 RMSE
will be black since black is associated with small pixel values.

%}
```



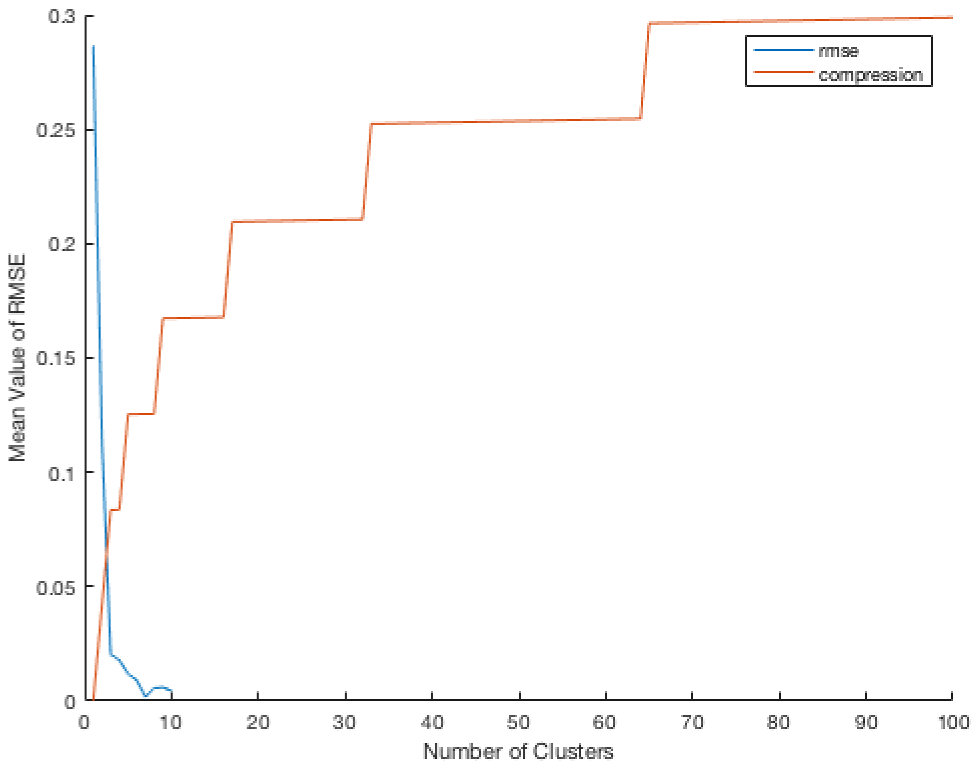


```
%7.
format long
mean_r = zeros(1, 10);
for i = 1:10
    mean_r(i) = mean(rmse(:, i));
end

figure
plot(mean_r);
title('Mean values of RMSE');
xlabel('Number of Clusters');
ylabel('Mean Value of RMSE');



%{
As K increases, RMSE decreases and when K approaches N, the RMSE
 approaches
0. Thus, K-means can be used for image compression, because it uses
 less
memory to store our image, yet also retains the most important
 features of
our image. The role of the parameter K is to determine how precise of
 a
representation we want of our original image. The higher K is, the
 better
our k-means approximated image will look.
%}
```

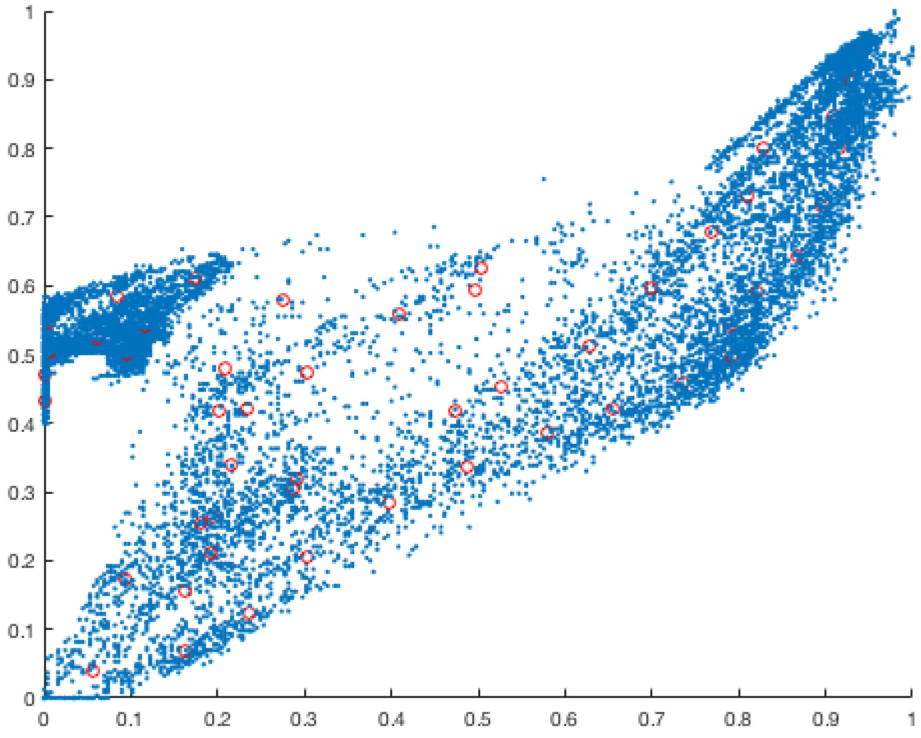


```
%8.
compress = zeros(1, 100);
total_bits = 24*13824;
for i = 1:100
    compress(i) = (24*i + 13824*ceil(log2(i)))/(total_bits);
end
figure
hold on
plot(mean_r);
plot(compress);
xlabel('Number of Clusters');
ylabel('Mean Value of RMSE');
legend('rmse', 'compression')

%{
```

```
The shape of our compression rate looks like a log function plus a
 linear
factor due to the ceil function. If we want a compression ratio from
 10% -
15%, we should use around K = 5 to K = 10 clusters as shown in our
 graph.
%}
```



```
%9.

[labels, means50] = kmeans(img, 50);
figure
hold on
scatter3(img(:, 1), img(:, 2), img(:, 3), '.');
for i = 1:50
    scatter3(means50(i, 1), means50(i,2), means50(i, 3),  'r');
end
hold off


%{
Each pixel is a vector valued continous function such that each
 element in
the vector represents the intensity of the r, b, or g channel. Thus,
 we discretize
each pixel (vector valued continues function) using the values of our
centroid vector (class). So essentially, we are assigning our pixels
 to a
few achievable values instead of a continuous range of r, g, or b
 values.
This is directly equivalent to discretization in signal processing
 where
instead we are working with vector valued functions.


%}
```



Published with MATLAB® R2017a