a. kmeans_single

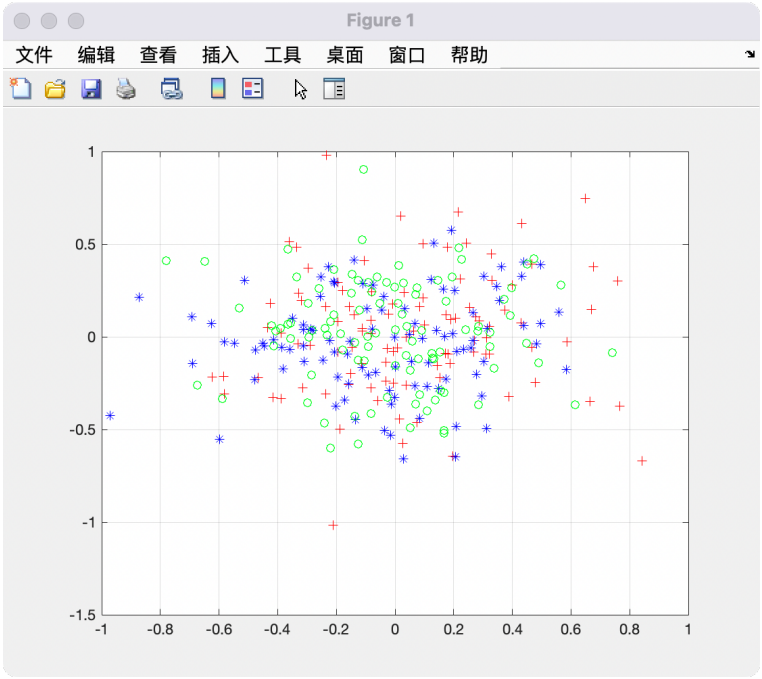
Code(with comments):

yunxing.m | random_data.m | segmentation.m | kmeans_single.m * | test.m

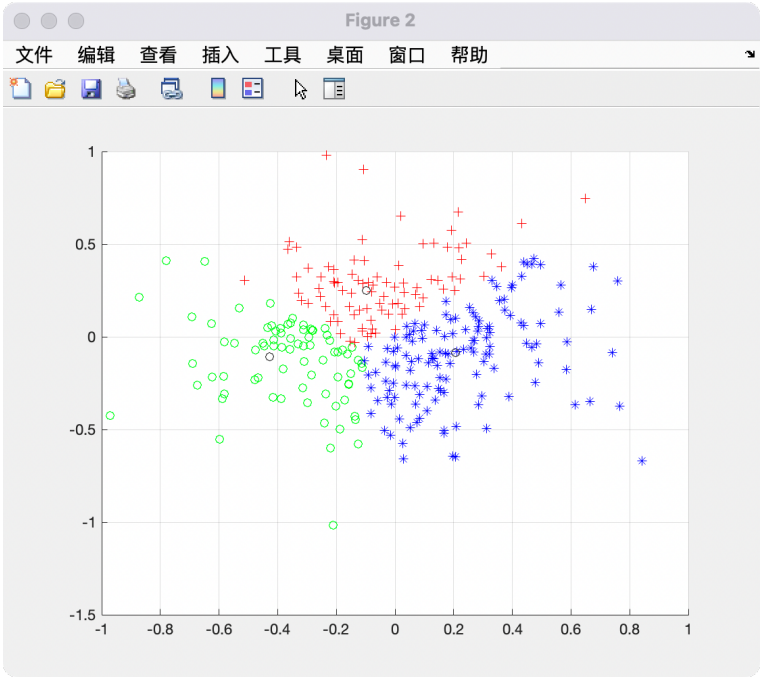
```
function [ids,means,ssd] = kmeans_single(X,K,iters)
[m,n] = size(X);
res = zeros(m,n+1); %The n+1 line used to show ids
means = zeros(K,n);
res(:,1:n)=X(:,:);
iter = 0;
for x=1:K
    means(x,:)=X(randi(m,1),:); %Randomly generate cluster center
end
while 1
while 1
distence = zeros(1,K);
num = zeros(1,K);
new_center = zeros(K,n);
ssd = 0;
for x = 1:m
    for y = 1:K
    distence(y) = norm(X(x,:) - means(y,:)); %Calculate the distance to each cluster
    end
    [~, temp] = min(distence); %Get the number of nearest cluster
    ssd = ssd + min(distence)^2; %Get ssd
    res(x,n+1) = temp;
end
k = 0;
for y = 1:K
    for x = 1:m
        if res(x,n+1)==y
           new_center(y,:) = new_center(y,:) + res(x,1:n); %Add up all the points that belong to the same group
           num(y) = num(y) + 1;
        end
    end
    new_center(y,:) = new_center(y,:) / num(y); %Get the new central point
    if norm(new_center(y,:) - means(y,:)) < 0.1
        k = k + 1;
    end
end
if k == K %When the offset of all the central points is less than the threshold, the central point remains the same.
     break;
else
      means = new_center; %Get the new_center
end
end
iter = iter + 1;
if iter == iters %run iters times
    break;
end
end
[m, n] = size(res);
ids = res(:,n);
hold on;
grid on;
end
```

I generated two-dimensional data and use kmeans_single() function so that through the output image we can get the effect of this function.

Input:



Output:



Through the output, we can find that after using the kmeans_single function, the pixels in the output have been classified according to a different cluster. Because the value of k is three this time, all the points are divided into three different colors of clusters according to the distance from each center.

Code:(test)

yunxing.m | random_data.m | segmentation.m | kmeans_single.m | test.m *

```
clear all;
close all;
clc;
a = [0 0 ];
S1 = [.1 0 ;0 .1];
data1 = mvnrnd(a,S1,100);
b = [0 0 ];
S2=  [.1 0 ;0 .1];
data2 = mvnrnd(b,S2,100);
c = [0 0 ];
S3 = [.1 0 ;0 .1];
data3 = mvnrnd(c,S3,100);
%Show randomly generated points
plot(data1(:,1),data1(:,2),'r+');
hold on;
plot(data2(:,1),data2(:,2),'b*');
plot(data3(:,1),data3(:,2),'go');
grid on;
%Three types of data are combined into one data class
data = [data1;data2;data3];
[ids,means,ssd] = kmeans_single(data,3,5);
[m,n] = size(data);
res = zeros(m,n+1);
res(:,1:n) = data(:,:);
res(:,n+1) = ids(:,:);
[m,n] = size(res);
figure;
hold on;
%Show the result of using kmeans_single/multiple
for i = 1:m
    if res(i,n) == 1
         plot(res(i,1),res(i,2),'r+');
         plot(means(1,1),means(1,2),'ko');
    elseif res(i,n) == 2
         plot(res(i,1),res(i,2),'b*');
         plot(means(2,1),means(2,2),'ko');
    elseif res(i,n) == 3
         plot(res(i,1),res(i,2),'go');
         plot(means(3,1),means(3,2),'ko');
    else
         plot(res(i,1),res(i,2),'m*');
         plot(means(4,1),means(4,2),'ko');
    end
end
grid on
```

ids:

300x1 double

	1	2	3	4	5	6	7	8	9	10	11
1	1										
2	2										
3	3										
4	1										
5	1										
6	1										
7	2										
8	1										
9	2										
10	1										
11	2										
12	2										
13	2										
14	3										
15	1										
16	3										
17	1										
18	1										
19	2										
20	1										
21	1										
22	2										
23	2										
24	1										
25	3										
26	3										
27	2										
28	1										
29	2										
30	3										

means:

3x2 double

	1	2
1	-0.0478	0.2030
2	0.2460	-0.2314
3	-0.1193	-0.1838

ssd:

1x1 double

	1
1	28.7017

b. kmeans_multiple

Code:(with comments)

yunxing.m | random_data.m | segmentation.m | kmeans_single.m | test.m | kmeans_multiple.m

```
function [ids,means,ssd] = kmeans_multiple(X,K,iters,R)
ssdmin = 2^1024 %Define a value that is huge enough
[m,n] = size(X);
best_center = zeros(K,n);
best_res = zeros(m,n+1);
for r = 1 : R %Do R times random start
res = zeros(m,n+1); %The n+1 line used to show ids
means = zeros(K,n);
res(:,1:n)=X(:,:);
iter = 0;
for x = 1:K
    means(x,:)=X(randi(m,1),:); %Randomly generate cluster center
end
while 1
while 1
distence = zeros(1,K);
num = zeros(1,K);
new_center = zeros(K,n);
ssd = 0;
for x = 1:m
    for y = 1:K
    distence(y) = norm(X(x,:) - means(y,:)); %Calculate the distance to each cluster
    end
    [~, temp] = min(distence); %Get the number of nearest cluster
    ssd = ssd + min(distence)^2; %Get ssd
    res(x,n+1) = temp;
end
k = 0;
for y = 1:K
    for x = 1:m
        if res(x,n+1) == y
           new_center(y,:) = new_center(y,:) + res(x,1:n); %Add up all the points that belong to the same group
           num(y) = num(y) + 1;
        end
    end
    new_center(y,:) = new_center(y,:) / num(y); %Get the new central point
    if norm(new_center(y,:) - means(y,:)) < 0.1
        k = k + 1;
    end
end
if k == K %When the offset of all the central points is less than the threshold, the central point remains the same.
    break;
else
     means = new_center; %Get the new_center
end
end
iter = iter + 1;
if iter == iters %run iters times
    break;
end
end
if ssd < ssdmin
    ssdmin = ssd %when get a smaller sdd, sddmin to store the value
    best_center = means; %best_center to store the means when the sdd is the smallest
    best_res = res; %best_res to store the ids when the sdd is the smallest
end
end
[m, n] = size(res);
ids = best_res(:,n); %get the ids with smallest ssd;
ssd = ssdmin; %get the smallest ssd;
means = best_center; %get the center with smallest ssd;
hold on;
grid on;
end
```

Based on kmeans_single, I add a loop, so it restarts several times, and then when I get the smallest ssd, I use ssdmin, best_center, best_res to store the corresponding value that needs to be output with the lowest ssd.

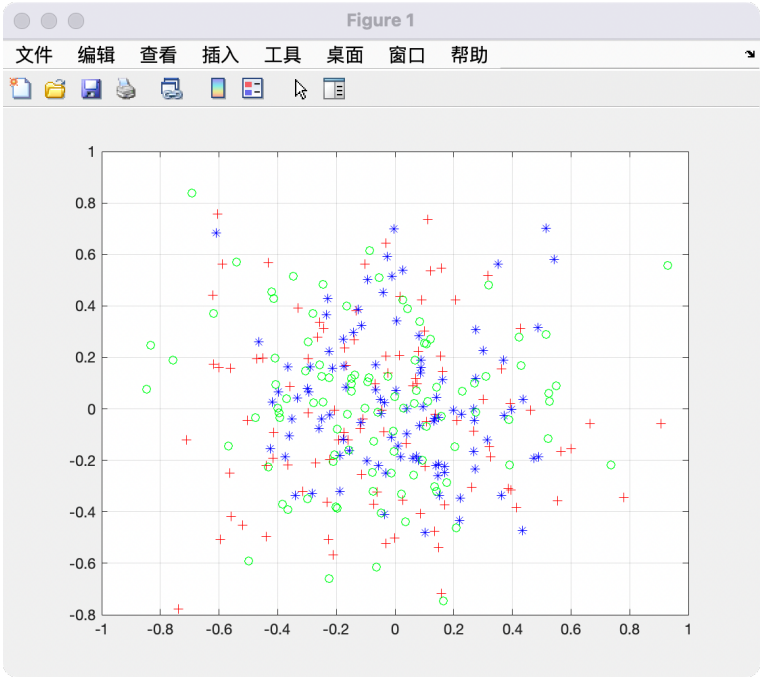
Code(test)

The test code is almost the same with the one of the kmeans_single. The only difference is to replace kmeans_single function with kmeans_multiple function.

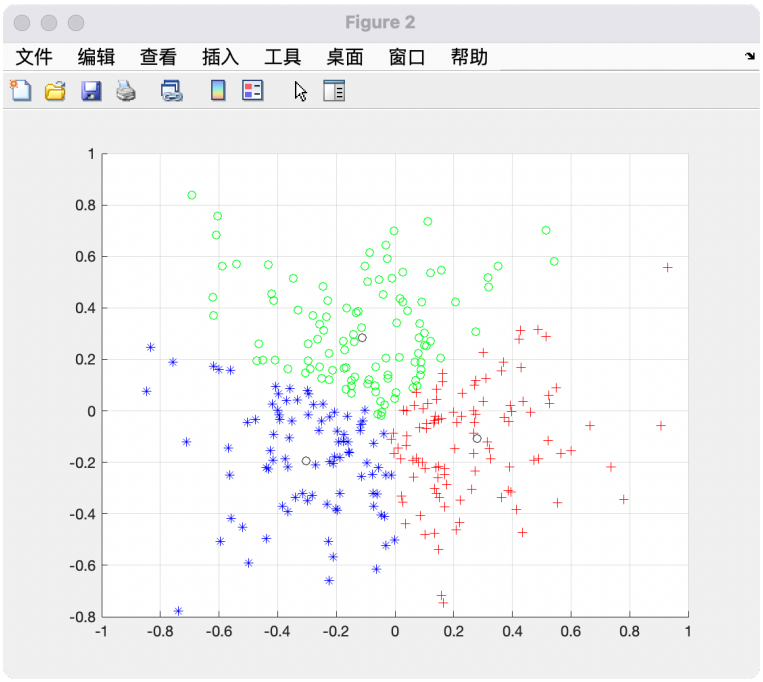
```
[ids,means,ssd] = kmeans_single(data,3,5);
```

```
[ids,means,ssd] = kmeans_multiple(data,3,5,5);
```

Input:



Output:



Comparing with the kmeans_single, I can find that each center point exists more accurately in the center of each cluster, because the ssd is the smallest at this time, which means that the image obtained is more accurate.

Ids:

distence | ids

300x1 double

	1	2	3	4	5	6	7	8	9	10	11
1	1										
2	1										
3	3										
4	2										
5	3										
6	2										
7	1										
8	3										
9	1										
10	2										
11	1										
12	3										
13	1										
14	2										
15	3										
16	2										
17	3										
18	3										
19	3										
20	3										
21	1										
22	3										
23	2										
24	1										

means:

3x2 double

	1	2
1	0.2795	-0.1080
2	-0.3024	-0.1945
3	-0.1111	0.2844

ssd:

1x1 double

	1
1	26.5086

C. Segment_kmeans

Code(with comments):

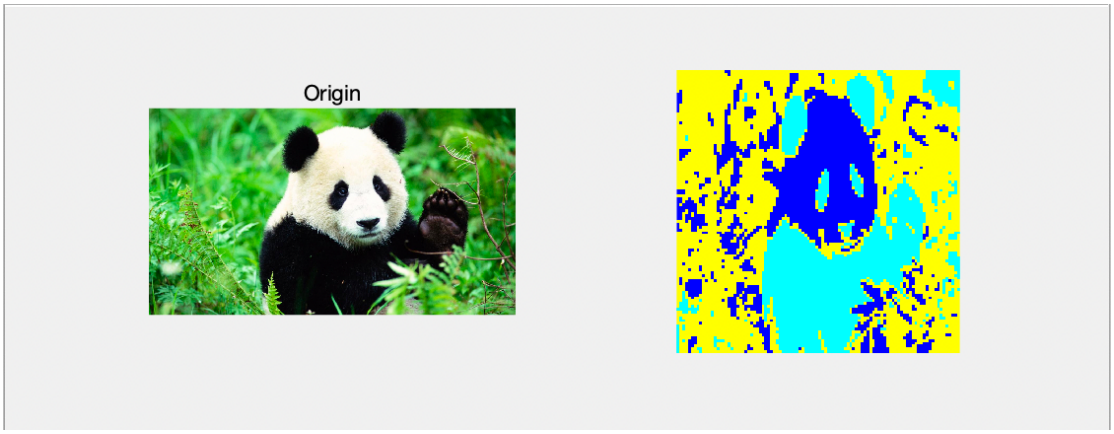
random_data.m | kmeans_single.m | test.m | kmeans_multiple.m | Segment_kmeans.m | test2.m

```
function [im_out] = Segment_kmeans(im_in,K,iters,R)
org = imread(im_in); %Input image
figure;
subplot(2,2,1);
imshow(org),title('Origin'); %To show the origin
org = imresize(org,[100,100]); %Downsample the images
[m,n,a] = size(org);
% decompose the image into RGB--3 channels
A = reshape(org(:, :, 1), m*n, 1);
B = reshape(org(:, :, 2), m*n, 1);
C = reshape(org(:, :, 3), m*n, 1);
data = [A B C]; %compose into one data
data = im2double(data);
[ids,means,ssd] = kmeans_multiple(data, K, iters, R); %Use kmeans_multiple
result = reshape(ids, m, n); % Reverse conversion to picture form
subplot(2,2,2);
%Convert tag matrix to RGB image and show the result
im_out = im2uint8(label2rgb(result));
end
```

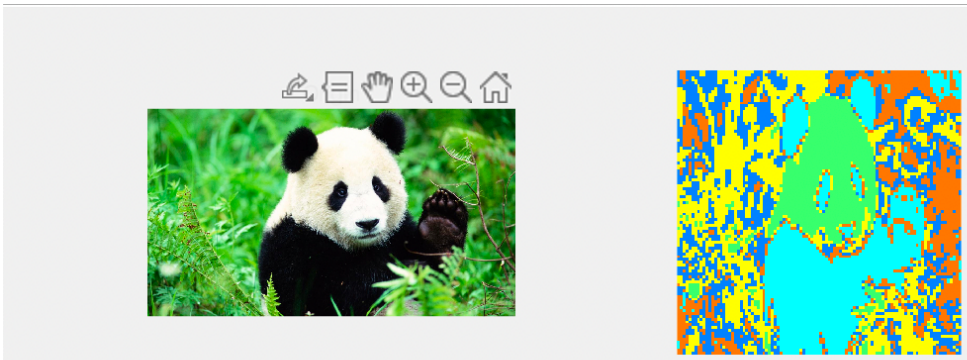
First input the image, then reduce the size to 100*100, in order to reduce the running time. Then convert the three-dimensional picture format into a two-dimensional matrix, after that, process it through kmeans_single, and finally convert the data into picture and then output it.

Picture 1.

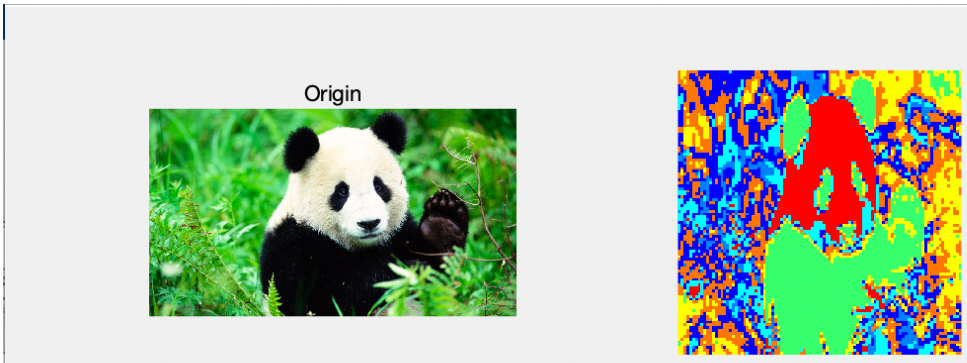
```
imshow(Segment_kmeans('im1.png',3,7,5));
```



```
imshow(Segment_kmeans('im1.png',5,15,15));
```

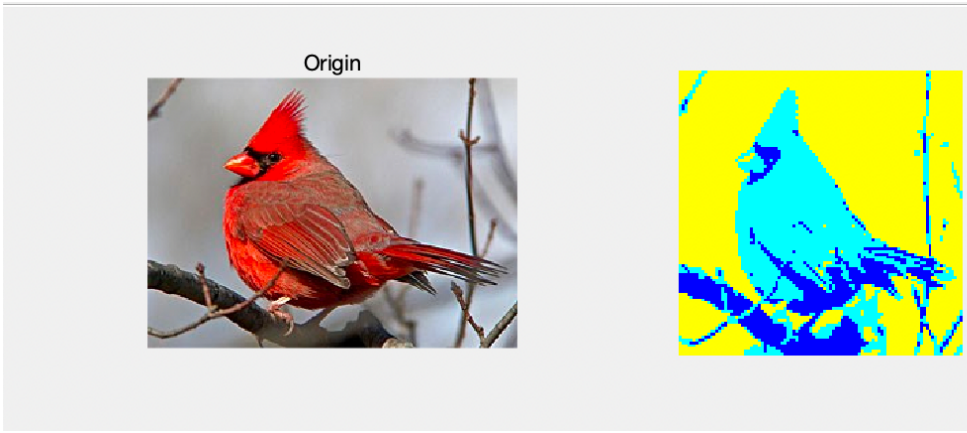


```
imshow(Segment_kmeans('im1.png',7,30,20));
```

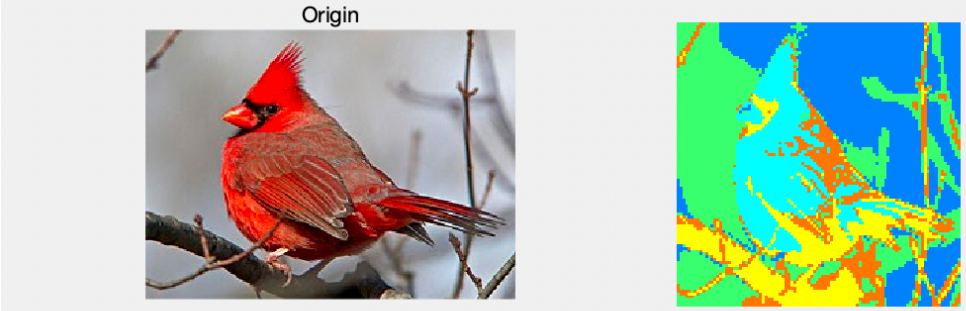


Picture 2.

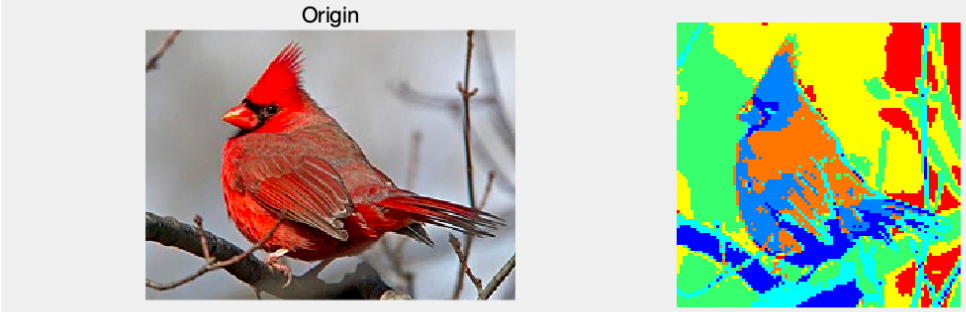
```
imshow(Segment_kmeans('im2.png',3,15,20));
```



```
imshow(Segment_kmeans('im2.png',5,30,5));
```

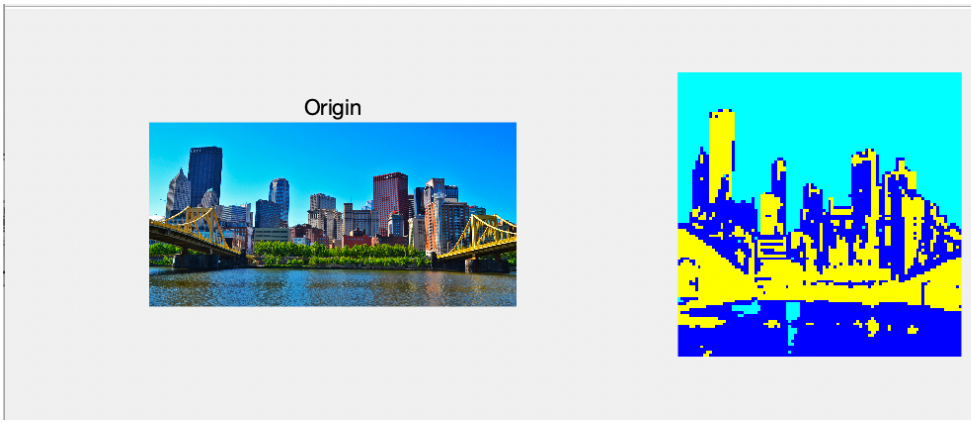



```
imshow(Segment_kmeans('im2.png',7,7,15));
```

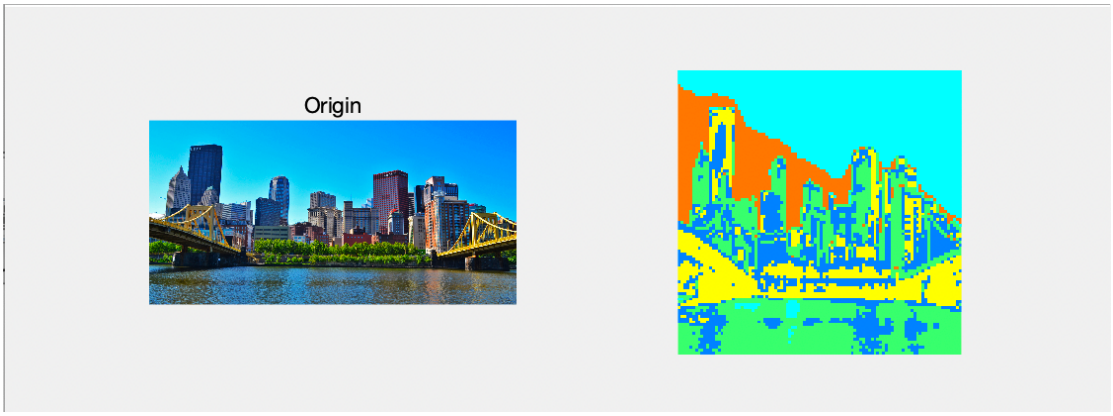


Picture 3.

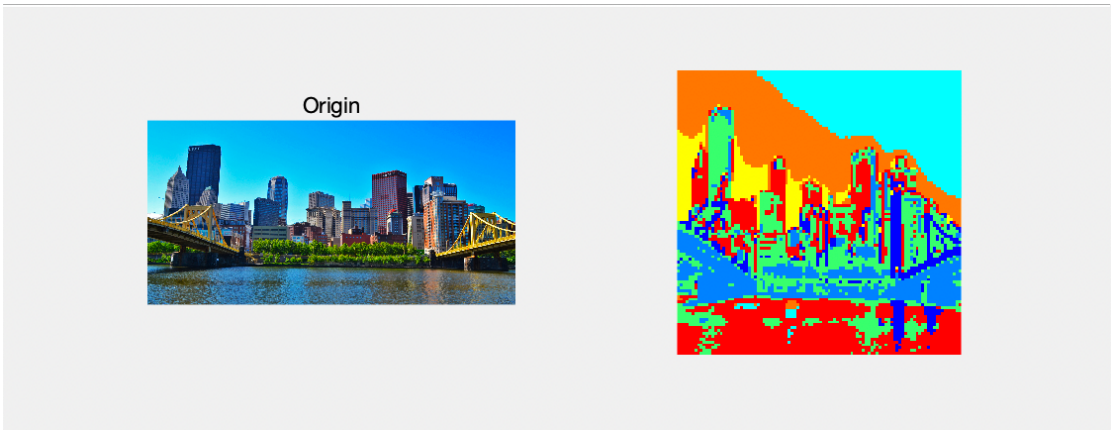
```
imshow(Segment_kmeans('im3.png',3,30,15));
```



```
imshow(Segment_kmeans('im3.png',5,7,20));
```

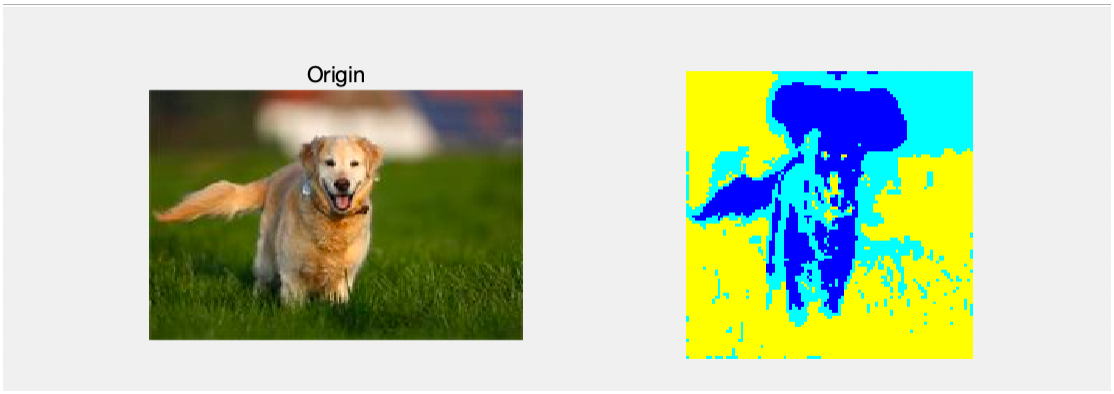



```
imshow(Segment_kmeans('im3.png',7,30,20));
```

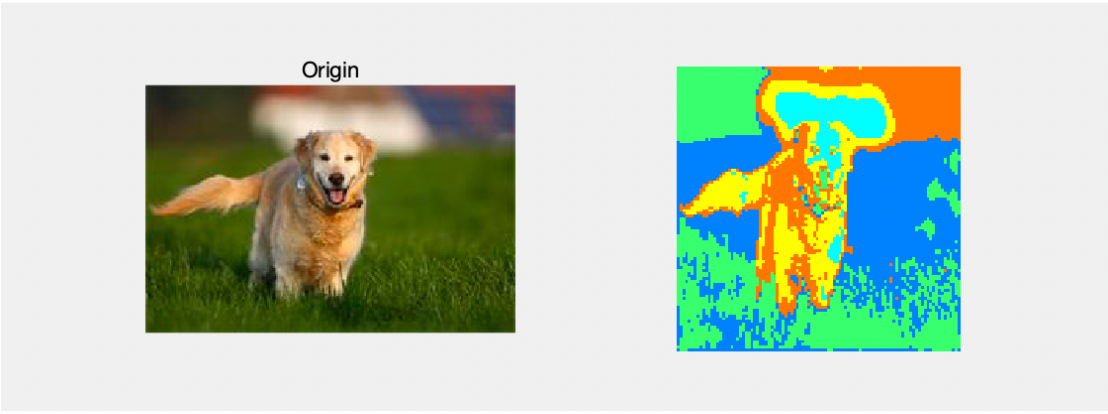


Picture 4.

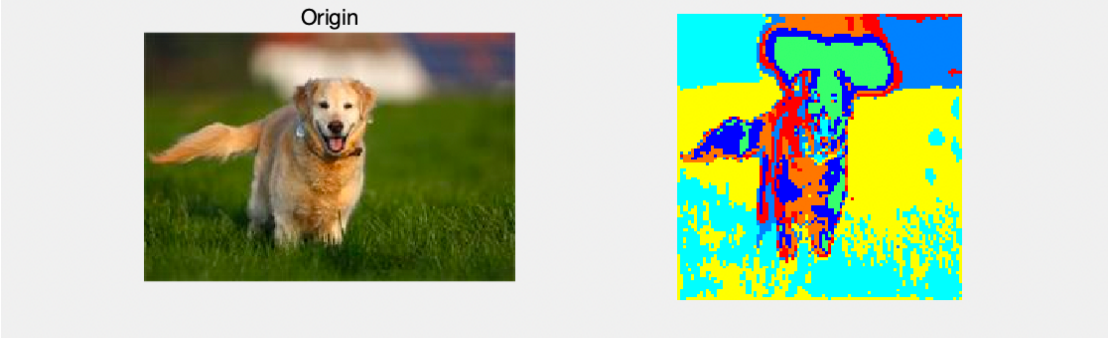
```
imshow(Segment_kmeans('im4.png',3,7,5));
```



```
imshow(Segment_kmeans('im4.png',5,15,15));
```




```
imshow(Segment_kmeans('im4.png',7,30,20));
```



I can find that with the increase of the values of iters and R, the impact of segmentation is better. But the functions I write still can't segment different objects some times because their colors are too close.

Haoyang Zeng
4597953