Project Report

Yin Gong #50290271

(1)

For this part, the code file is otsu.m.

When intending to run this code, only change the path name of testing pictures.

The code:

img = imread('/Users/apple/Desktop/Project1-images/t1.jpg');

img\_test = img(:,:,1); %Convert test images into R grayscale images

%img\_test = img(:,:,2);%Convert test images into G grayscale images

%img\_test = img(:,:,3);%Convert test images into B grayscale images

imshow(img\_test);

[m, n] = size(img\_test);

pixels = m\*n;% the sum of pixels

% for counting the pixels of every scale

count = zeros(1,256);

newcount = zeros(1,256);

%probability of each scale

pro = zeros(1,256);

%initialize the maximum value

maxvalue = 0;

maxthreshold = 0;

for i = 1:m

for j = 1:n

tmp = img\_test(i, j);

if tmp == 0

count(256) = count(256) + 1;

else count(tmp) = count(tmp) + 1;

end

end

end

%adjust the order of array

for i = 1:256

if i == 256

newcount(1) = count(i);

else newcount(i + 1) = count(i);

end

end

for i = 1:256

pro(i) = newcount(i)/pixels;

end

for i = 1 :256

w0 = 0 ; w1 = 0;u\_0 = 0; u0 = 0; u\_1 = 0; u1 = 0; tmp = 0;

for j = 1:256

if j <= i

w0 = w0 + pro(j);

u\_0 = u\_0 + (j-1)\*pro(j);

else

w1 = w1 + pro(j);

u\_1 = u\_1 + (j-1)\*pro(j);

end

end

u0 = u\_0/w0;

u1 = u\_1/w1;

tmp = w0\*w1\*((u0 - u1)^2);

if tmp > maxvalue

maxvalue = tmp;

maxthreshold = i - 1;

end

end

%compare the final results to the standard function;

standard = 255\*graythresh(img\_test);

error = maxthreshold - standard;

if error == 0

fprintf('%s\n','true');

else

fprintf('%s\n','false');

end

%binarize the images

for i=1:m

for j=1:n

if img\_test(i,j)> maxthreshold

img\_test(i,j) = 255;

else img\_test(i,j) = 0;

end

end

end

imshow(img\_test);

The image result:

For each image, there are three channels. According to each channel, I use otsu’s method to binarize the image.

Red channel:



Green channel:



Blue channel:



Red channel:



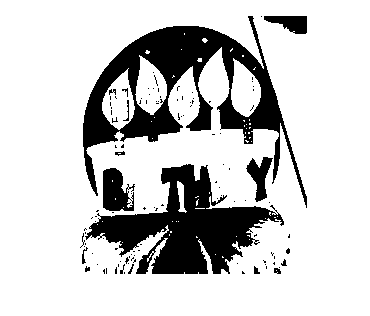
Green channel:



Blue channel:



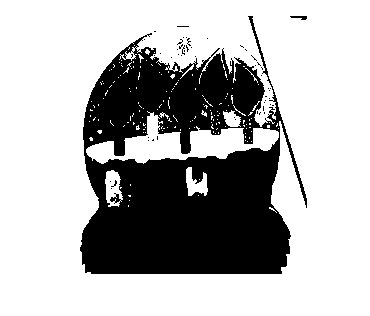
Red channel:



Green channel:



Blue channel:



Analysis: In my program, to verify my result, I use the standard library to generate another result to compare my result. The result is true.

(2)

For this part, the code files are quadtree.m and split4.m

My designed algorithm is use two points of pictures to use quadtree recursively. The two points are the left corner point and the right corner point.

But I didn’t achieve this part successively. My recursive algorithm seems to have problems.

My final results are like this:



(3)

For this part, the code files are mediancut\_2.m, mediancut\_4.m, mediancut\_16.m, mediansort.m

The mediansort.m is to sort based on the range.

When intending to run codes, only change the path name of testing pictures.

Just run mediancut\_2.m, mediancut\_4.m, mediancut\_16.m

The codes:

function res = mediansort(all)

A = all(:,1);

B = all(:,2);

C = all(:,3);

values = [range(A),range(B),range(C)];

maxvalue = max(values);

switch maxvalue

case range(A)

z= 1

all = sortrows(all,1);

case range(B)

z = 2

all = sortrows(all,2);

case range(C)

z = 3

all = sortrows(all,3);

end

res = all;

mediancut\_2.m

img = imread('/Users/apple/Desktop/Project1-images/t1.jpg');

maxvalue\_r = -1;

minvalue\_r = 256;

maxvalue\_g = -1;

minvalue\_g = 256;

maxvalue\_b = -1;

minvalue\_b = 256;

[m, n, z] = size(img);

A = (m\*n);

B = (m\*n);

C = (m\*n);

img\_red = img(:,:,1);

img\_green = img(:,:,2);

img\_blue = img(:,:,3);

for i = 1 : m

for j = 1 : n

A(i\*j) = img\_red(i,j);

minvalue\_r = min(img\_red(i,j),minvalue\_r);

maxvalue\_r = max(img\_red(i,j),maxvalue\_r);

end

end

range\_r = maxvalue\_r - minvalue\_r;

for i = 1 : m

for j = 1 : n

B(i\*j) = img\_green(i,j);

minvalue\_g = min(img\_green(i,j),minvalue\_g);

maxvalue\_g = max(img\_green(i,j),maxvalue\_g);

end

end

range\_g = maxvalue\_g - minvalue\_g;

for i = 1 : m

for j = 1 : n

C(i\*j) = img\_blue(i,j);

minvalue\_b = min(img\_blue(i,j),minvalue\_b);

maxvalue\_b = max(img\_blue(i,j),maxvalue\_b);

end

end

range\_b = maxvalue\_b - minvalue\_b;

ranges = [range\_r ,range\_g, range\_b];

maxrange = max(ranges);

all = [A',B',C'];

switch maxrange

case range\_r

base\_r = sortrows(all,1);

[x,y] = size(base\_r);

avg\_r1 = sum(base\_r(1:fix(x/2),1))/fix(x/2);

avg\_r2 = sum(base\_r(fix(x/2) + 1:x,1))/fix(x/2);

avg\_g1 = sum(base\_r(1:fix(x/2),2))/fix(x/2);

avg\_g2 = sum(base\_r(fix(x/2) + 1:x,2))/fix(x/2);

avg\_b1 = sum(base\_r(1:fix(x/2),3))/fix(x/2);

avg\_b2 = sum(base\_r(fix(x/2) + 1:x,3))/fix(x/2);

for i = 1 : m

for j = 1:n

if ((img(i,j,1) - avg\_r1)^2 +(img(i,j,2) - avg\_g1)^2 + (img(i,j,3) - avg\_b1)^2) <= ((img(i,j,1) - avg\_r2)^2 +(img(i,j,2) - avg\_g2)^2 + (img(i,j,3) - avg\_b2)^2)

img(i,j,1) = fix(avg\_r1);

img(i,j,2) = fix(avg\_g1);

img(i,j,3) = fix(avg\_b1);

else

img(i,j,1) = fix(avg\_r2);

img(i,j,2) = fix(avg\_g2);

img(i,j,3) = fix(avg\_b2);

end

end

end

case range\_g

base\_g = sortrows(all,2);

[x,y] = size(base\_g);

avg\_r1 = sum(base\_g(1:fix(x/2),1))/fix(x/2);

avg\_r2 = sum(base\_g(fix(x/2) + 1:x,1))/fix(x/2);

avg\_g1 = sum(base\_g(1:fix(x/2),2))/fix(x/2);

avg\_g2 = sum(base\_g(fix(x/2) + 1:x,2))/fix(x/2);

avg\_b1 = sum(base\_g(1:fix(x/2),3))/fix(x/2);

avg\_b2 = sum(base\_g(fix(x/2) + 1:x,3))/fix(x/2);

for i = 1 : m

for j = 1:n

if img(i,j,2) < avg\_g2

img(i,j,1) = fix(avg\_r1);

img(i,j,2) = fix(avg\_g1);

img(i,j,3) = fix(avg\_b1);

else

img(i,j,1) = fix(avg\_r2);

img(i,j,2) = fix(avg\_g2);

img(i,j,3) = fix(avg\_b2);

end

end

end

case range\_b

base\_b = sortrows(all,3);

[x,y] = size(base\_b);

avg\_r1 = sum(base\_b(1:fix(x/2),1))/fix(x/2);

avg\_r2 = sum(base\_b(fix(x/2) + 1:x,1))/fix(x/2);

avg\_g1 = sum(base\_b(1:fix(x/2),2))/fix(x/2);

avg\_g2 = sum(base\_b(fix(x/2) + 1:x,2))/fix(x/2);

avg\_b1 = sum(base\_b(1:fix(x/2),3))/fix(x/2);

avg\_b2 = sum(base\_b(fix(x/2) + 1:x,3))/fix(x/2);

for i = 1 : m

for j = 1:n

if img(i,j,1) < avg\_b2

img(i,j,1) = fix(avg\_r1);

img(i,j,2) = fix(avg\_g1);

img(i,j,3) = fix(avg\_b1);

else

img(i,j,1) = fix(avg\_r2);

img(i,j,2) = fix(avg\_g2);

img(i,j,3) = fix(avg\_b2);

end

end

end

end

imshow(img);

mediancut\_4.m

img = imread('/Users/apple/Desktop/Project1-images/t1.jpg');

img\_red = img(:,:,1);

img\_green = img(:,:,2);

img\_blue = img(:,:,3);

[m, n, z] = size(img);

A = (m\*n);

B = (m\*n);

C = (m\*n);

for i = 1 : m

for j = 1 : n

A(i\*j) = img\_red(i,j);

end

end

for i = 1 : m

for j = 1 : n

B(i\*j) = img\_green(i,j);

end

end

for i = 1 : m

for j = 1 : n

C(i\*j) = img\_blue(i,j);

end

end

all = [A',B',C'];

A = all(:,1);

B = all(:,2);

C = all(:,3);

values = [range(A),range(B),range(C)];

maxvalue = max(values);

switch maxvalue

case range(A)

z= 1

all = sortrows(all,1);

case range(B)

z = 2

all = sortrows(all,2);

case range(C)

z = 3

all = sortrows(all,3);

end

all1 = all(1:(m\*n)/2,:);

all2 = all((m\*n)/2 + 1:end,:);

A = all1(:,1);

B = all1(:,2);

C = all1(:,3);

values = [range(A),range(B),range(C)];

maxvalue = max(values);

switch maxvalue

case range(A)

z= 1

all1 = sortrows(all1,1);

case range(B)

z = 2

all1 = sortrows(all1,2);

case range(C)

z = 3

all1 = sortrows(all1,3);

end

A = all2(:,1);

B = all2(:,2);

C = all2(:,3);

values = [range(A),range(B),range(C)];

maxvalue = max(values);

switch maxvalue

case range(A)

z= 1

all2 = sortrows(all2,1);

case range(B)

z = 2

all2 = sortrows(all2,2);

case range(C)

z = 3

all2 = sortrows(all2,3);

end

[x,y] = size(all1);

avg\_r1 = sum(all1(1:fix(x/2),1))/fix(x/2);

avg\_r2 = sum(all1(fix(x/2) + 1:x,1))/fix(x/2);

avg\_g1 = sum(all1(1:fix(x/2),2))/fix(x/2);

avg\_g2 = sum(all1(fix(x/2) + 1:x,2))/fix(x/2);

avg\_b1 = sum(all1(1:fix(x/2),3))/fix(x/2);

avg\_b2 = sum(all1(fix(x/2) + 1:x,3))/fix(x/2);

[x,y] = size(all2);

avg\_r3 = sum(all2(1:fix(x/2),1))/fix(x/2);

avg\_r4 = sum(all2(fix(x/2) + 1:x,1))/fix(x/2);

avg\_g3 = sum(all2(1:fix(x/2),2))/fix(x/2);

avg\_g4 = sum(all2(fix(x/2) + 1:x,2))/fix(x/2);

avg\_b3 = sum(all2(1:fix(x/2),3))/fix(x/2);

avg\_b4 = sum(all2(fix(x/2) + 1:x,3))/fix(x/2);

for i = 1:m

for j =1:n

value1 = (img(i,j,1) - avg\_r1)^2 + (img(i,j,2) - avg\_g1)^2 + (img(i,j,3) - avg\_b1)^2;

value2 = (img(i,j,1) - avg\_r2)^2 + (img(i,j,2) - avg\_g2)^2 + (img(i,j,3) - avg\_b2)^2;

value3 = (img(i,j,1) - avg\_r3)^2 + (img(i,j,2) - avg\_g3)^2 + (img(i,j,3) - avg\_b3)^2;

value4 = (img(i,j,1) - avg\_r4)^2 + (img(i,j,2) - avg\_g4)^2 + (img(i,j,3) - avg\_b4)^2;

values = [value1, value2, value3, value4];

minvalue = min(values);

switch minvalue

case value1

img(i,j,1) = avg\_r1;

img(i,j,2) = avg\_g1;

img(i,j,3) = avg\_b1;

case value2

img(i,j,1) = avg\_r2;

img(i,j,2) = avg\_g2;

img(i,j,3) = avg\_b2;

case value3

img(i,j,1) = avg\_r3;

img(i,j,2) = avg\_g3;

img(i,j,3) = avg\_b3;

case value4

img(i,j,1) = avg\_r4;

img(i,j,2) = avg\_g4;

img(i,j,3) = avg\_b4;

end

end

end

imshow(img);

mediancut\_16.m

img = imread('/Users/apple/Desktop/Project1-images/t1.jpg');

img\_red = img(:,:,1);

img\_green = img(:,:,2);

img\_blue = img(:,:,3);

[m, n, z] = size(img);

A = (m\*n);

B = (m\*n);

C = (m\*n);

for i = 1 : m

for j = 1 : n

A(i\*j) = img\_red(i,j);

end

end

for i = 1 : m

for j = 1 : n

B(i\*j) = img\_green(i,j);

end

end

for i = 1 : m

for j = 1 : n

C(i\*j) = img\_blue(i,j);

end

end

all = [A',B',C'];

all = mediansort(all);

all1 = all(1:(m\*n)/2,:);

all2 = all((m\*n)/2 + 1:end,:);

all1 = mediansort(all1);

all2 = mediansort(all2);

all3 = all1(1:fix((m\*n)/4),:);

all4 = all1(fix((m\*n)/4) + 1:end,:);

all5 = all2(1:fix((m\*n)/4),:);

all6 = all2(fix((m\*n)/4) + 1:end,:);

all3 = mediansort(all3);

all4 = mediansort(all4);

all5 = mediansort(all5);

all6 = mediansort(all6);

[x, y] = size(all3);

all7 = all3(1:fix(x/2),:);

all8 = all3(fix(x/2) + 1:end,:);

all9 = all4(1:fix(x/2),:);

all10 = all4(fix(x/2) + 1:end,:);

all11 = all5(1:fix(x/2),:);

all12 = all5(fix(x/2) + 1:end,:);

all13 = all6(1:fix(x/2),:);

all14 = all6(fix(x/2) + 1:end,:);

all7 = mediansort(all7);

all8 = mediansort(all8);

all9 = mediansort(all9);

all10 = mediansort(all10);

all11 = mediansort(all11);

all12 = mediansort(all12);

all13 = mediansort(all13);

all14 = mediansort(all14);

[x, y] = size(all7);

avg\_r1 = sum(all7(1:fix(x/2),1))/fix(x/2);

avg\_r2 = sum(all7(fix(x/2) + 1:x,1))/fix(x/2);

avg\_g1 = sum(all7(1:fix(x/2),2))/fix(x/2);

avg\_g2 = sum(all7(fix(x/2) + 1:x,2))/fix(x/2);

avg\_b1 = sum(all7(1:fix(x/2),3))/fix(x/2);

avg\_b2 = sum(all7(fix(x/2) + 1:x,3))/fix(x/2);

[x, y] = size(all8);

avg\_r3 = sum(all8(1:fix(x/2),1))/fix(x/2);

avg\_r4 = sum(all8(fix(x/2) + 1:x,1))/fix(x/2);

avg\_g3 = sum(all8(1:fix(x/2),2))/fix(x/2);

avg\_g4 = sum(all8(fix(x/2) + 1:x,2))/fix(x/2);

avg\_b3 = sum(all8(1:fix(x/2),3))/fix(x/2);

avg\_b4 = sum(all8(1:fix(x/2),3))/fix(x/2);

[x, y] = size(all9);

avg\_r5 = sum(all9(1:fix(x/2),1))/fix(x/2);

avg\_r6 = sum(all9(fix(x/2) + 1:x,1))/fix(x/2);

avg\_g5 = sum(all9(1:fix(x/2),2))/fix(x/2);

avg\_g6 = sum(all9(fix(x/2) + 1:x,2))/fix(x/2);

avg\_b5 = sum(all9(1:fix(x/2),3))/fix(x/2);

avg\_b6 = sum(all9(1:fix(x/2),3))/fix(x/2);

[x, y] = size(all10);

avg\_r7 = sum(all10(1:fix(x/2),1))/fix(x/2);

avg\_r8 = sum(all10(fix(x/2) + 1:x,1))/fix(x/2);

avg\_g7 = sum(all10(1:fix(x/2),2))/fix(x/2);

avg\_g8 = sum(all10(fix(x/2) + 1:x,2))/fix(x/2);

avg\_b7 = sum(all10(1:fix(x/2),3))/fix(x/2);

avg\_b8 = sum(all10(1:fix(x/2),3))/fix(x/2);

[x, y] = size(all11);

avg\_r9 = sum(all11(1:fix(x/2),1))/fix(x/2);

avg\_r10 = sum(all11(fix(x/2) + 1:x,1))/fix(x/2);

avg\_g9 = sum(all11(1:fix(x/2),2))/fix(x/2);

avg\_g10 = sum(all11(fix(x/2) + 1:x,2))/fix(x/2);

avg\_b9 = sum(all11(1:fix(x/2),3))/fix(x/2);

avg\_b10 = sum(all11(1:fix(x/2),3))/fix(x/2);

[x, y] = size(all12);

avg\_r11 = sum(all12(1:fix(x/2),1))/fix(x/2);

avg\_r12 = sum(all12(fix(x/2) + 1:x,1))/fix(x/2);

avg\_g11 = sum(all12(1:fix(x/2),2))/fix(x/2);

avg\_g12 = sum(all12(fix(x/2) + 1:x,2))/fix(x/2);

avg\_b11 = sum(all12(1:fix(x/2),3))/fix(x/2);

avg\_b12 = sum(all12(1:fix(x/2),3))/fix(x/2);

[x, y] = size(all13);

avg\_r13 = sum(all13(1:fix(x/2),1))/fix(x/2);

avg\_r14 = sum(all13(fix(x/2) + 1:x,1))/fix(x/2);

avg\_g13 = sum(all13(1:fix(x/2),2))/fix(x/2);

avg\_g14 = sum(all13(fix(x/2) + 1:x,2))/fix(x/2);

avg\_b13 = sum(all13(1:fix(x/2),3))/fix(x/2);

avg\_b14 = sum(all13(1:fix(x/2),3))/fix(x/2);

[x, y] = size(all14);

avg\_r15 = sum(all14(1:fix(x/2),1))/fix(x/2);

avg\_r16 = sum(all14(fix(x/2) + 1:x,1))/fix(x/2);

avg\_g15 = sum(all14(1:fix(x/2),2))/fix(x/2);

avg\_g16 = sum(all14(fix(x/2) + 1:x,2))/fix(x/2);

avg\_b15 = sum(all14(1:fix(x/2),3))/fix(x/2);

avg\_b16 = sum(all14(1:fix(x/2),3))/fix(x/2);

for i = 1:m

for j =1:n

value1 = (img(i,j,1) - avg\_r1)^2 + (img(i,j,2) - avg\_g1)^2 + (img(i,j,3) - avg\_b1)^2;

value2 = (img(i,j,1) - avg\_r2)^2 + (img(i,j,2) - avg\_g2)^2 + (img(i,j,3) - avg\_b2)^2;

value3 = (img(i,j,1) - avg\_r3)^2 + (img(i,j,2) - avg\_g3)^2 + (img(i,j,3) - avg\_b3)^2;

value4 = (img(i,j,1) - avg\_r4)^2 + (img(i,j,2) - avg\_g4)^2 + (img(i,j,3) - avg\_b4)^2;

value5 = (img(i,j,1) - avg\_r5)^2 + (img(i,j,2) - avg\_g5)^2 + (img(i,j,3) - avg\_b5)^2;

value6 = (img(i,j,1) - avg\_r6)^2 + (img(i,j,2) - avg\_g6)^2 + (img(i,j,3) - avg\_b6)^2;

value7 = (img(i,j,1) - avg\_r7)^2 + (img(i,j,2) - avg\_g7)^2 + (img(i,j,3) - avg\_b7)^2;

value8 = (img(i,j,1) - avg\_r8)^2 + (img(i,j,2) - avg\_g8)^2 + (img(i,j,3) - avg\_b8)^2;

value9 = (img(i,j,1) - avg\_r9)^2 + (img(i,j,2) - avg\_g9)^2 + (img(i,j,3) - avg\_b9)^2;

value10 = (img(i,j,1) - avg\_r10)^2 + (img(i,j,2) - avg\_g10)^2 + (img(i,j,3) - avg\_b10)^2;

value11 = (img(i,j,1) - avg\_r11)^2 + (img(i,j,2) - avg\_g11)^2 + (img(i,j,3) - avg\_b11)^2;

value12 = (img(i,j,1) - avg\_r12)^2 + (img(i,j,2) - avg\_g12)^2 + (img(i,j,3) - avg\_b12)^2;

value13 = (img(i,j,1) - avg\_r13)^2 + (img(i,j,2) - avg\_g13)^2 + (img(i,j,3) - avg\_b13)^2;

value14 = (img(i,j,1) - avg\_r14)^2 + (img(i,j,2) - avg\_g14)^2 + (img(i,j,3) - avg\_b14)^2;

value15 = (img(i,j,1) - avg\_r15)^2 + (img(i,j,2) - avg\_g15)^2 + (img(i,j,3) - avg\_b15)^2;

value16 = (img(i,j,1) - avg\_r16)^2 + (img(i,j,2) - avg\_g16)^2 + (img(i,j,3) - avg\_b16)^2;

values = [value1, value2, value3, value4, value5, value6, value7, value8,value9, value10, value11, value12,value13, value14, value15, value16];

minvalue = min(values);

switch minvalue

case value1

img(i,j,1) = avg\_r1;

img(i,j,2) = avg\_g1;

img(i,j,3) = avg\_b1;

case value2

img(i,j,1) = avg\_r2;

img(i,j,2) = avg\_g2;

img(i,j,3) = avg\_b2;

case value3

img(i,j,1) = avg\_r3;

img(i,j,2) = avg\_g3;

img(i,j,3) = avg\_b3;

case value4

img(i,j,1) = avg\_r4;

img(i,j,2) = avg\_g4;

img(i,j,3) = avg\_b4;

case value5

img(i,j,1) = avg\_r5;

img(i,j,2) = avg\_g5;

img(i,j,3) = avg\_b5;

case value6

img(i,j,1) = avg\_r6;

img(i,j,2) = avg\_g6;

img(i,j,3) = avg\_b6;

case value7

img(i,j,1) = avg\_r7;

img(i,j,2) = avg\_g7;

img(i,j,3) = avg\_b7;

case value8

img(i,j,1) = avg\_r8;

img(i,j,2) = avg\_g8;

img(i,j,3) = avg\_b8;

case value9

img(i,j,1) = avg\_r9;

img(i,j,2) = avg\_g9;

img(i,j,3) = avg\_b9;

case value10

img(i,j,1) = avg\_r10;

img(i,j,2) = avg\_g10;

img(i,j,3) = avg\_b10;

case value11

img(i,j,1) = avg\_r11;

img(i,j,2) = avg\_g11;

img(i,j,3) = avg\_b11;

case value12

img(i,j,1) = avg\_r12;

img(i,j,2) = avg\_g12;

img(i,j,3) = avg\_b12;

case value13

img(i,j,1) = avg\_r13;

img(i,j,2) = avg\_g13;

img(i,j,3) = avg\_b13;

case value14

img(i,j,1) = avg\_r14;

img(i,j,2) = avg\_g14;

img(i,j,3) = avg\_b14;

case value15

img(i,j,1) = avg\_r15;

img(i,j,2) = avg\_g15;

img(i,j,3) = avg\_b15;

case value16

img(i,j,1) = avg\_r16;

img(i,j,2) = avg\_g16;

img(i,j,3) = avg\_b16;

end

end

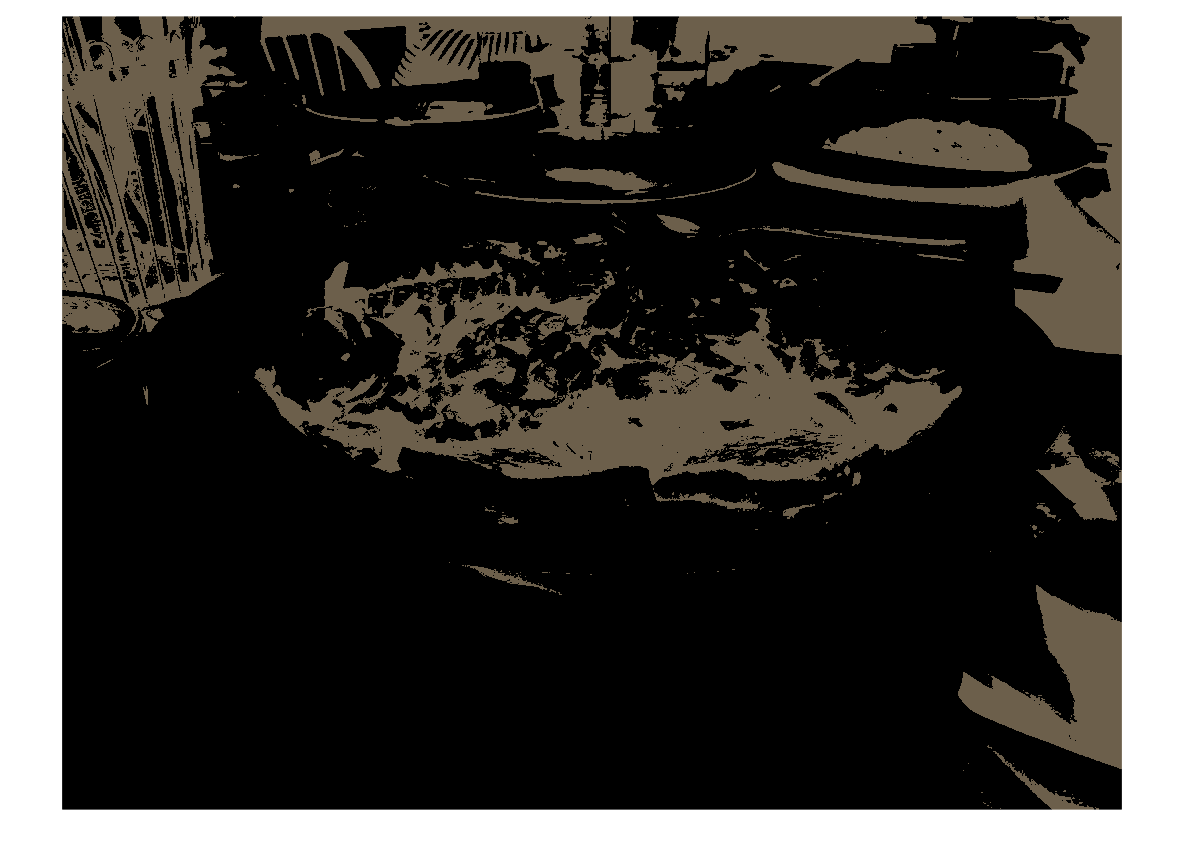
end

imshow(img);

The final results:

Bit-depth = 1







Bit-depth = 2

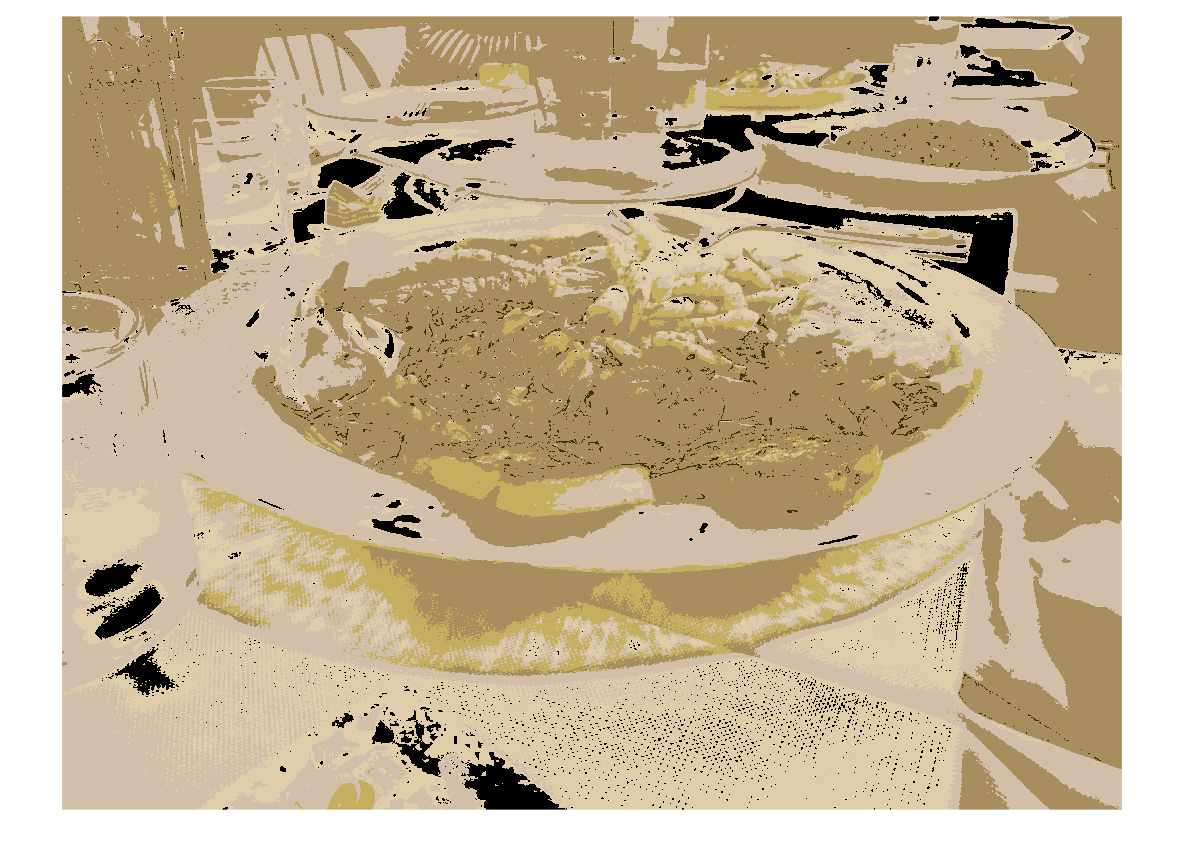






Bit-depth = 4







Analysis: My whole program did not use recursive programming. That is why I did not achieve bit-depth = 8 successfully. I write part of program of bit-depth = 8. This algorithm is right. But it has no efficiency.