Homework 2: Huffman Coding and Vector Quantization

M10907324 吳俊逸

Problem 1:

Implementation of Huffman Coding

Participants can do simple implementation for alphabets with any probability vector as presented in the class.

Code：

Basic program structure：

Coding: Count the number of occurrences-->Probability-->Create codebook-->Replace text with codebook content

Decoding: count a few 1-->do all 1 processing-->get the index of each code-->replace the codebook content with text

```
clc;clear all;close all;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%           input English and space only           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

String_input ="a pig in my room";



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                coding start                      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

eng_list=["a","b","c","d","e","f","g","h","i","j","k","l","m","n","o"
,"p","q","r","s","t","u","v","w","x","y","z"," "];
count_val=0;                                          % Word counter
count_zero=0;                                         % Has 0 counter
count_coding=0;                                       % Calculate the coding
number counter

for i=1:1:27
   n(2,i) = count(String_input, eng_list(1,i));      % Count the
number of occurrences of each letter
   count_val=count_val+n(2,i);                        % Count the total
```

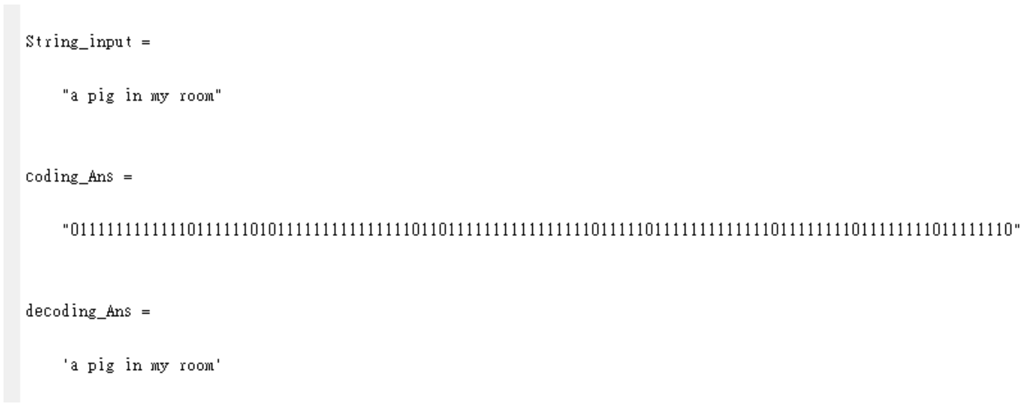
```
number of words
end

for i=1:1:27
    n(3,i) = n(2,i) / count_val;                    % Calculate the
probability of each letter
    pa(1,i)=n(3,i);
end

[B,I]=sort(pa);                                     % Sort B = Probability
of occurrence I = Index after sorting

for i=1:1:27
    if B(1,i)==0
        count_zero=count_zero+1;                    % Count a few zeros
    end
end

for i=1:1:27
    if B(1,i)~=0
        sort_num(1,i-count_zero)=eng_list(1,I(1,i)); % Sort after
removing zero, index-->symbol
    end
end
count_coding=27-count_zero;
k=0;
for i=1:1:count_coding
    for j = 1:1:i-1
        k=k*10+1;
    end
    if i ~= count_coding
        coding_num1(1,i)=k*10;
    else
        coding_num1(1,i)=k;
    end
    k=0;
end
```

```
coding_Ans=String_input;
for j=1:1:count_coding

coding_Ans=strrep(coding_Ans,sort_num(1,j),int2str(coding_num1(1,j)))
; %Letters are converted into huffman coding
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                coding end                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%              decoding_start                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

decoding_val=findstr(coding_Ans, '0');
%Found 0 address
decoding_sub(1,1)=decoding_val(1,1);
%To know the address difference between two adjacent zeros, first
fill in the prime minister, because the array does not have the
zeroth item
a=0;b=0;
for i=2:1:length(decoding_val)
   decoding_sub(1,i)=decoding_val(1,i)-decoding_val(1,i-1);
end

for i=2:1:length(decoding_val)
   if decoding_sub(1,i) > (count_coding-1)
%If the subtracted number is greater than the code length of the
largest codebook content

      a=a+1;                                                  %Know
that this is the code length greater than the largest codebook
content
      index(1,a)=i;
%Store its index item
```

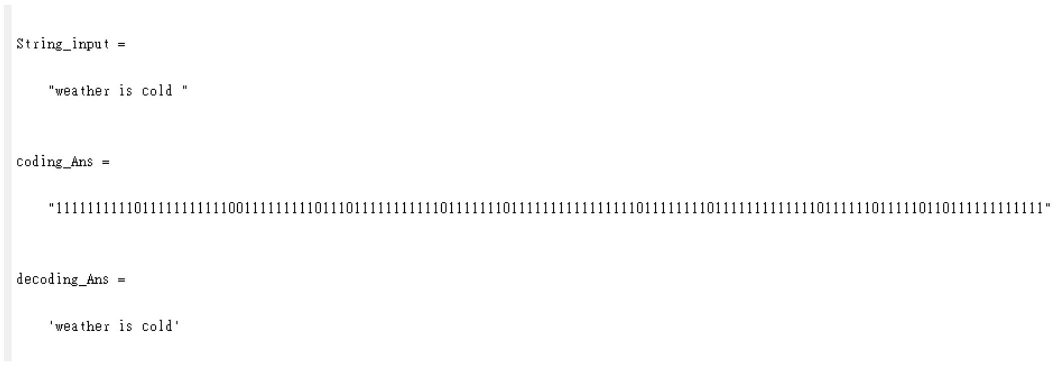
```
       sub_val(1,a)=fix(decoding_sub(1,i)/(count_coding-1));
%How many older
       mod_val(1,a)=mod(decoding_sub(1,i),(count_coding-1));
%The last yard left

       for k=1:1:sub_val(1,a)
           add(1,k)=count_coding;
%Combine the codes to be replaced (all ones)
       end

       if mod_val(1,a) ~= 0
           add(1,sub_val(1,a)+1)=mod_val(1,a);
%Combine the code to be replaced (the last code)
       else
           add(1,sub_val(1,a))=count_coding-1;
%Combine the code to be replaced (the last code)
       end

       if a==1
            decoding= [decoding_sub(1:index(1,1)-1) add(1,1:end)
decoding_sub(index(1,1)+1:end)];%Put the first code to be replaced
into
       else
           decoding= [decoding(1:index(1,a)-1+b) add(1,1:end)
decoding(index(1,a)+1+b:end)];%Put other codes to be replaced into
       end
       b=b+length(add)-1; %Calculate the value of index to be
inserted into the array
       add=[];%Empty storage
   end
end
decoding_Ans=decoding; %Copy the data to another array
for i=1:1:length(decoding)

decoding_Ans=strrep(decoding_Ans,decoding(1,i),sort_num(1,decoding(1,
i)));%huffman coding into letters
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                decoding_end                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                 Show the answer              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
String_input
%English words to be encoded
coding_Ans                                                    %The
result after Huffman encoding
decoding_Ans                                                  %The
decoded text
```

example：



```
String_input =

    "a pig in my room"


coding_Ans =

    "01111111111110111111010111111111111111101101111111111111111101111011111111111110111111101111111101111110"


decoding_Ans =

    'a pig in my room'
```



```
String_input =

    "weather is cold "


coding_Ans =

    "111111111101111111111110011111111110111011111111111101111111011111111111111111101111111101111111111111101111110111101101111111111111"


decoding_Ans =

    'weather is cold'
```

Problem 2:

Linde-Buzo-Gray (LBG) algorithm to perform Vector Quantization or any variants of LBG can also be considered for implementation.
Participants can work on any image data-sets (even for a single image is fine) for training and implementation.

I prepare two methods

1. Compress the image directly

The codebook CB is obtained through the LBG () classification method. CB is composed of N k-dimensional codewords (N=256, k=16), i=1,2,3,...N-1.Compressed image: Suppose a grayscale image T of size T is divided into image blocks B of size. Then find the index table Gi through optimization.Image decompression: A compressed image can be generated according to the index table and CW.

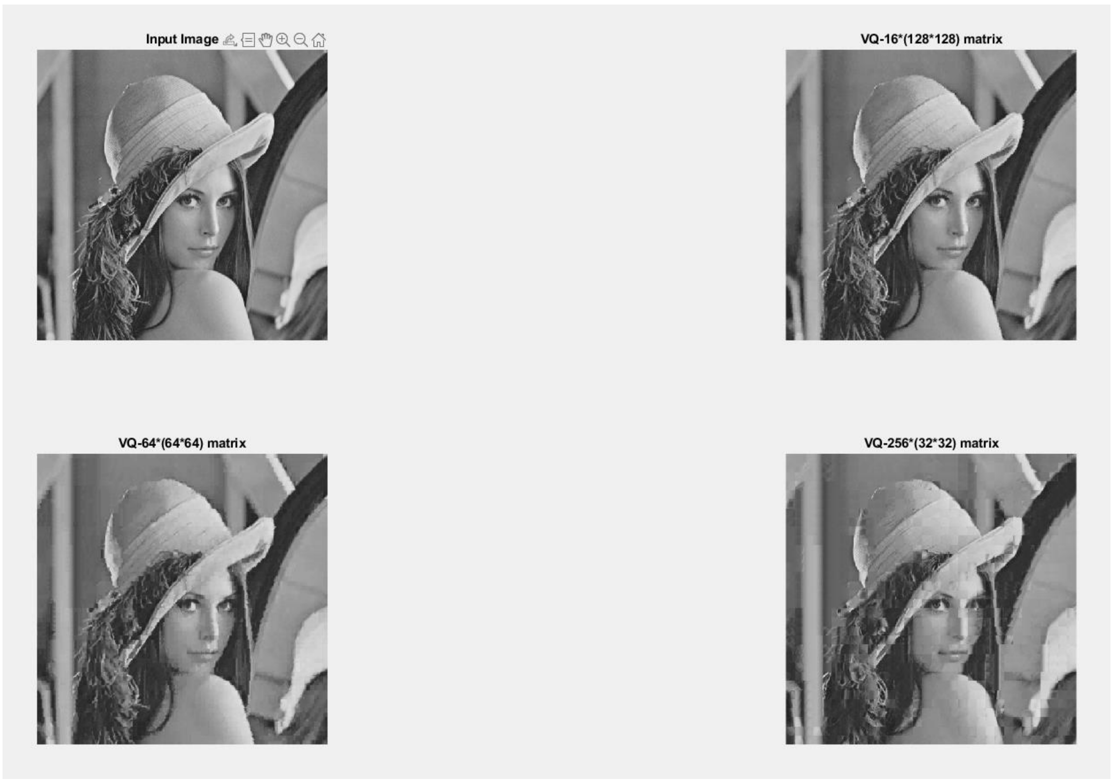
Code：

```
clc;clear all;close all;
%%
tic;
a=4;
b=a*a;
input=imread('lena.bmp');
I=double(imread('lena.bmp'));
B=im2col(I,[a,a],'distinct');% Decompose the image into a
16*(128*128) matrix
[m,n]=size(B);
N=256;
CB=zeros(m,N);
CW=zeros(1,n);
rng(999);
CB_cnt=randperm(n);
CB=B(:,CB_cnt(1:N));
for x=1:10%10 iterations
    for y=1:n%training
        p=B(1:b,y)*ones(1,N);
        [~,yy]=min(sum((p-CB).^2));
        CW(y)=yy;
    end
```

```
      for z=1:N%Select
         v=find(CW==z);
         for k=1:m
              nv=sum(B(k,v))/numel(v);
              CB(k,z)=nv;
         end
      end
end
toc
DE=zeros(m,n);
for i=1:n
    DE(:,i)=CB(:,CW(i));
end
img_4=col2im(DE,[a,a],[512,512],'distinct');


%%
tic;
a=8;
b=a*a;
input=imread('lena.bmp');
I=double(imread('lena.bmp'));
B=im2col(I,[a,a],'distinct');% Decompose the image into a 64*(64*64)
matrix
[m,n]=size(B);
N=256;
CB=zeros(m,N);
CW=zeros(1,n);
rng(999);
CB_cnt=randperm(n);
CB=B(:,CB_cnt(1:N));
for x=1:10%10 iterations
    for y=1:n%training
        p=B(1:b,y)*ones(1,N);
        [~,yy]=min(sum((p-CB).^2));
        CW(y)=yy;
    end
      for z=1:N%Select
```

```
            v=find(CW==z);
            for k=1:m
                nv=sum(B(k,v))/numel(v);
                CB(k,z)=nv;
            end
         end
end
toc
DE=zeros(m,n);
for i=1:n
    DE(:,i)=CB(:,CW(i));
end
img_8=col2im(DE,[a,a],[512,512],'distinct');

%%
tic;
a=16;
b=a*a;
input=imread('lena.bmp');
I=double(imread('lena.bmp'));
B=im2col(I,[a,a],'distinct');% Decompose the image into a 256*(32*32)
matrix
[m,n]=size(B);
N=256;
CB=zeros(m,N);
CW=zeros(1,n);
rng(999);
CB_cnt=randperm(n);
CB=B(:,CB_cnt(1:N));
for x=1:10%10 iterations
    for y=1:n%training
        p=B(1:b,y)*ones(1,N);
        [~,yy]=min(sum((p-CB).^2));
        CW(y)=yy;
    end
       for z=1:N%Select
            v=find(CW==z);
            for k=1:m
            nv=sum(B(k,v))/numel(v);
            CB(k,z)=nv;
        end
      end
end
toc
DE=zeros(m,n);
for i=1:n
   DE(:,i)=CB(:,CW(i));
end
img_16=col2im(DE,[a,a],[512,512],'distinct');

%%
subplot(221);
imshow(input);title('Input Image');
subplot(222);
imshow(uint8(img_4));title('VQ-16*(128*128) matrix');
subplot(223);
imshow(uint8(img_8));title('VQ-64*(64*64) matrix');
subplot(224);
imshow(uint8(img_16));title('VQ-256*(32*32) matrix');
```

Results of the code：



2.

Code1→Codebook training, take codebook=256 as an example：

Basic program structure：

-->Take the two adjacent points in the photo as a group and send it to the codebook to evaluate the distance

--> rank the results after evaluation and take the first index

--> add this group of data to code_avg, and calculate the number of times in code_time Medium

-->Calculate the average of all points closer to the node

-->Replace the result in the original codebook

-->The preset number of training is ten times, which means that the above actions are repeated ten times

-->The final codebook Save in txt

-->END

```
clc;clear all;close all;
A2=[27  107  13  39  133  194  164  39  167  148  223  108  112  70  113
180  169  192  203  104  116  240  218  37  160  23  89  4  213  127  100
117  218  64  0  65  156  189  122  53  100  165  153  33  23  226  116
74  99  196  170  48  54  17  168  162  119  81  140  86  44  208  108
102  166  222  177  175  84  174  248  96  154  132  233  239  163  80
172  157  55  221  106  95  91  63  12  137  26  164  85  52  136  218
158  7  90  203  88  163  160  223  140  236  144  231  198  93  24  15
80  15  17  191  90  228  64  18  6  54  72  116  155  199  93  250  237
226  ;
91  27  13  37  127  117  164  165  47  148  221  103  129  71  192  180
250  119  194  217  181  163  102  159  28  228  226  118  125  58  52
160  121  244  97  91  149  226  62  106  163  226  144  225  114  240  78
150  137  97  97  0  166  194  97  30  76  7  3  201  205  222  163  50
66  64  196  132  158  59  44  165  40  26  90  25  1  252  230  80  81
149  30  236  144  82  60  86  109  173  208  198  240  142  152  107  77
193  202  166  238  77  30  14  155  201  90  188  242  160  199  22  230
45  114  79  145  15  246  43  188  36  121  88  85  215  78  215
]; %Codebook after taking random random numbers

for i=1:1:10
    code_avg=[0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  ;
        0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0];
    code_time=[0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  ;
        0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0];
    data=double(imread("./imgdata/cameraman.tif"));
    nx=256;
    ny=256;
    for i = 1:1:nx
       for j=1:2:ny%Throw the two points in the picture as a group
into the codebook to evaluate the distance
          for k = 1:1:16
```

```
              sub(1,k)=(data(i,j)-A2(1,k))^2+(data(i,j+1)-
A2(2,k))^2;%Calculate the distance between each point and the node
          end
          [B,I]=sort(sub);%"B" is the look after sorting, "I" is the
index of the original position before sorting
          code_avg(1,I(1,1))=code_avg(1,I(1,1))+data(i,j);%Add the
points closest to the node into the variable code_avg
          code_avg(2,I(1,1))=code_avg(2,I(1,1))+data(i,j+1);
          code_time(1,I(1,1))=code_time(1,I(1,1))+1;%Calculate the
accumulation times in code_time
       end
    end
    for j=1:1:2
        for i = 1:1:128
            code_avg(j,i)=code_avg(j,i)/code_time(1,i);%Average the
points near the node
            if(uint8(code_avg(j,i))~=0)
                A2(j,i)=uint8(code_avg(j,i));%Write the changed data
into the codebook variable
            end
        end
    end
end
fid=fopen(['./','codebook_256.txt'],'w');%Write file path
[r,c]=size(A2);% Get the number of rows and columns of the matrix
 for i=1:r
  for j=1:c
  fprintf(fid,'%f\t',uint8(A2(i,j)));%Write data to txt file
  end
  if(i==1)
      fprintf(fid,';\n');%Matlab notation for conversion to 2-
dimensional matrix reduction
  else
      fprintf(fid,'\n');
  end
 end
fclose(fid);
```

Code2→Use self-built codebook to apply to VQ compression：

```
clc;clear all;close all;
ccc=512;
input=imread('lena.bmp');

A_16=[27 70 110 113 149 153 169 180 183 188 192 215 223 228 240 248];
aaa_16=imread('lena.bmp');
for i=1:1:ccc
   for j = 1:1:ccc
       aaa_16(i,j);
      for k=1:1:16
          if(aaa_16(i,j)>A_16(1,k))
              sub_16(1,k)=aaa_16(i,j)-A_16(1,k);%Calculate the distance
between each point and the node
          else
              sub_16(1,k)=A_16(1,k)-aaa_16(i,j);%Calculate the distance
between each point and the node
          end
      end
      [B,I]=sort(sub_16);%"B" is the look after sorting, "I" is the
index of the original position before sorting
      aaa_16(i,j)=A_16(I(1,1));
   end
end



A_64=[5 12 13 19 20 23 24 27 39 41 46 47 51 53 55 59 61 64 65 72 74
76 81 82 91 92 99  106 107 110 111 114 115 118 121 136 146 148 149
152 155 156 157 160 163 169 173 174 177 179 182 183 185 187 205 210
215 222 224 228 232 242 246 254];
aaa_64=imread('lena.bmp');
for i=1:1:ccc
   for j = 1:1:ccc
       aaa_64(i,j);
      for k=1:1:64
          if(aaa_64(i,j)>A_64(1,k))
              sub_64(1,k)=aaa_64(i,j)-A_64(1,k);%Calculate the distance
between each point and the node
```

```
        else
            sub_64(1,k)=A_64(1,k)-aaa_64(i,j);%Calculate the distance
between each point and the node
        end
      end
      [B,I]=sort(sub_64);%"B" is the look after sorting, "I" is the
index of the original position before sorting
      aaa_64(i,j)=A_64(I(1,1));
   end
end

A_256=[27  107  13  39  133  194  164  39  167  148  223  108  112  70
113  180  169  192  203  104  116  240  218  37  160  23  89  4  213  127
100  117  218  64  0  65  156  189  122  53  100  165  153  33  23  226
116  74  99  196  170  48  54  17  168  162  119  81  140  86  44  208
108  102  166  222  177  175  84  174  248  96  154  132  233  239  163
80  172  157  55  221  106  95  91  63  12  137  26  164  85  52  136  218
158  7  90  203  88  163  160  223  140  236  144  231  198  93  24  15
80  15  17  191  90  228  64  18  6  54  72  116  155  199  93  250  237
226  ;
91  27  13  37  127  117  164  165  47  148  221  103  129  71  192  180
250  119  194  217  181  163  102  159  28  228  226  118  125  58  52
160  121  244  97  91  149  226  62  106  163  226  144  225  114  240  78
150  137  97  97  0  166  194  97  30  76  7  3  201  205  222  163  50
66  64  196  132  158  59  44  165  40  26  90  25  1  252  230  80  81
149  30  236  144  82  60  86  109  173  208  198  240  142  152  107  77
193  202  166  238  77  30  14  155  201  90  188  242  160  199  22  230
45  114  79  145  15  246  43  188  36  121  88  85  215  78  215  ];
aaa_256 = imread('lena.bmp');
nx=256;
ny=256;
for i = 1:1:nx
   for j=1:2:ny%Throw the two points in the picture as a group into
the codebook to evaluate the distance
      for k = 1:1:16
          sub(1,k)=(aaa_256(i,j)-A_256(1,k))^2+(aaa_256(i,j+1)-
A_256(2,k))^2;%Calculate the distance between each point and the node
      end
```

```
    [B,I]=sort(sub);%"B" is the look after sorting, "I" is the index
of the original position before sorting
    aaa_256(i,j)=A_256(1,I(1,1));
    aaa_256(i,j+1)=A_256(2,I(1,1));
  end
end
%%
subplot(221);
imshow(input);title('Input Image');
subplot(222);
imshow(uint8(aaa_16));title('VQ-Use 16 codebooks');
subplot(223);
imshow(uint8(aaa_64));title('VQ-Use 64 codebooks');
subplot(224);
imshow(uint8(aaa_256));title('VQ-Use 256 codebooks');
```

Results of the code：