DSP Homework 09

Su, Rundong

10

October 31, 2022

Contents

Abstract

In this article, I first write summaries and my further thoughts about the video we watch, and then, I states how IEEE754 define double type variable and its error when present a number. Then, I use M bits($M \in [2, 10]$) do a image quantization by Lloyd-Max algorithm, write my quantization result and discovery after quantization. Last, I derive the optimal quantization strategy for $x \in [0, 1]$ with pdf $p(x) = 1$.

1 About videos

1.1 Summaries about videos

1.1.1 The bob up of analog computer

It tells us the analog computer popular again in recent year due to the development of AI. The video tells us the development of AI and how analog computer is good at dealing neural networks. And the future of analog computer is prospected by the speaker.

1.1.2 Adults can also grow new nerve cells

At the beginning of the video, the speaker gives us an example that patients of her friend their disease cured but they get depressed. The speaker says this is because the medicine applied on patients stops the cancer cells multiplying but also stop newborn neurons being generated in their brain. Then the speaker introduce neurogenesis to us. And we all know that hippocampus can influence our learn and memory, it also influence our mood and emotion. But in a new trail conduct on mouse, they find that its new function is neurogenesis. And our brain produce about 700 new neurons per day, it this speed, when we get to 50 years old, the neurologist we are born with will be replaced by new neurons produced in the hippocampus. After it the speaker talks about the relation between neurogenesis and depression, and he says that anti-depression medicine helps produce new neuron, prevent neurogenesis will prevent the medicine effect. Last, she talks about how to promote neurogenesis from sport we do and food we eat. To sum up, any sport help our blood flow to brain is good for neurogenesis, and food rich in flavonoid is good for neurogenesis while food rich in lipid is not friendly to neurogenesis.

1.1.3 Jungle areas

This video is a documentary about jungles in our planet. it shows us Congo jungle which is the youngest jungle in the world, New-Guinea which is the world's largest jungle-covered island, jungle in Borneo, jungle sense near Philippine islands, jungles in Amazon basin and swampy forests of northern Sumatra, Indonesia. It shows animals living in the these jungles, and it tell us our human influence to them, and appeal us to protect jungles.

1.2 Further thoughts about videos

1.2.1 The bob up of analog computer

After watching this video, I feel the rise of analog computers is on the horizon, and in the past decades years, we usually think digital is advanced, analog computer is off the radar, but in recent years, it is popular again. It reminds one old and typical Chinese saying: "Changeable in prosperity and decline capricious in rise and fall(Thirty years in east bank of the river, thirty years in west bank of the river)". It is a bit like the development of our individuals, maybe now we are facing a lot troubles, ordeals, failure and so on. Sometimes we too care about the result, which lead us ignore the current time. If we make the best of everything we're faced with peacefully(include breathing, eating, drinking, working. . .), it will turned out very well after all, like the analog computer. This video makes me less impatient, and just do things we are facing with, as for result it will be fine.

1.2.2 Adults can also grow new nerve cells

It's kind of like a health video, and after watching this video, I learned a new way of keeping in good health. Sports can help us generate new neurons, and eat food rich in flavonoid can also achieve this. It also say light-diet is not beneficial to neurogenesis. So we should balance our diet, extreme ideas are undesirable.

1.2.3 Jungle areas

After watching this video, I know the importance of jungles. The sense impress me most is unsuspecting ants are unknowingly infected with spores spread by the parasitic fungus Cordyceps sinensis. The spores attach to the ant and germinate, spreading through the host through "long curly whiskers" called hyphae. Cordyceps essentially turns its host into a zombie slave, forcing the ant to climb to the top of the nearest plant and clench its jaws around a leaf or branch. The fungus then slowly eats up the ant, emerging from its head. The bulb at the end of the fungus's mycelium then grows and explodes, releasing more spores into the air to infect more unsuspecting ants. It is really scaring, is this fungus eat our humans too? I searching the relevant articles online and get the answer: there are more than 400 species of cordyceps fungi, each targeting specific insects such as ants, dragonflies, cockroaches, aphids and beetles [1]. So it is only effective to insects, luckily.

2 About digital number representation

2.1 Description for exact meanings of number type

In different program language, data types has different meanings, here we take c as an example.

1. byte

   Byte is a unit of measurement used in computer information technology to measure storage capacity. It also represents data types and language characters in some computer programming languages [2]. One byte is 8 bits.

2. short integer

   Short integer means this type variable is an integer, and it takes **2 bytes** memory space in computer, and its value is from -32728 to 32767.

3. integer

   An integer means this type variable is an integer, and it takes **4 bytes** memory space for 64-bits processor, and **2 bytes** for 32-bits processor in computer, and its value is from -2147483648 to 2147483647 for 64-bits processor and from -32728 to 32767 for 32-bits processor.

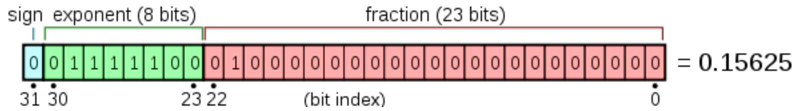


Figure 1: Distribution of 4 byte(cite from [3])

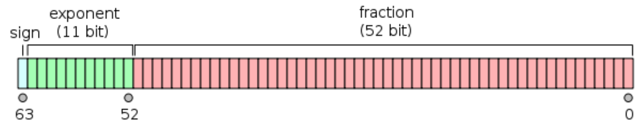


Figure 2: Distribution of 8byte(cite from [3])

4. float

   Float means this type variable has fractional part, and it takes **4 bytes** memory space in computer, the distribution of this 4 byte is shown as the Figure 1. Its value can be calculate by $v = (-1)^s * M * 2^E$, where s is the sign bit, stored in the highest bit of that 4 bytes, and M meas fractional part, stored in the lowest 23 bits of that 4 bytes, and E means exponent part, stored as Figure 1 shows. What's more, a float type variable has at least 6 sites effective fractional part, its value is from 1.2E-38 to 3.4E+38.

5. double

   Double means this type variable has fractional part, too. It takes **8 bytes** memory space in computer, the distribution of this 8 byte is shown as the Figure 2. Its value can be calculate by $v = (-1)^s * M * 2^E$, s, M, E has the same meaning as them in float part. What's more, a float type variable has at least 15 sites effective fractional part, its value is from its value is from 1.2E-38 to 3.4E+38.

6. quadruple types

   A quadruple types means it takes **16 bytes** memory space in computer, and in C only long double meet this condition. The value of this type is from 3.4E-4932 to 1.1E+4932, and it has at least 19 sites effective fractional part.

2.2 Description of the IEEE 754 definition for double type floating point numbers

IEEE 754 provides some specifications and standards on how to store floating point numbers of different precision. It doesn't give the definition of double type floating point numbers(on *IEEE Standard for Binary Floating-Point Arithmetic* published on 1985). It stipulate how to store double type floating point numbers.

The double type floating point numbers takes up 8 bytes storage space in computer. It can be shown as the Figure 3. s

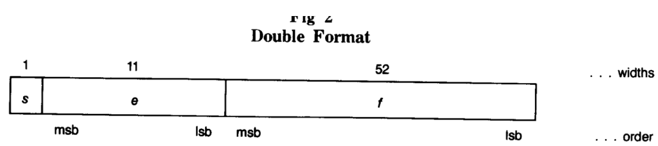


Figure 3: How a double type floating point number store(cite from [4])

means sign bit, $s = 0$, means it is a positive number, or else it is a negative number. **For normalized floating point number** e meas biased exponent, for double type floating point numbers e = E + 1023, and f means fraction, it only stores the fraction part of one number, and if it is not enough, we add zeros at the low part, if it is overflow we judge if 52th and 53th bit, if 53th is zero, we cut off the rest part; if 52th is 0 and 53th is 1(but after 53th bit it is all 0), we cut off over flow part; if both of them are 1(after 53th bit it is all 0), then we add 1 in 52th bit; if 53th bit is 1 and bit after 53th are not all 0, then we add 1 in 52th bit. The value can be calculate by $(-1)^s \times (1.f) \times 2^{e-1023}$. Also when e=2047(all 11 indices are 1):

(1) If f>0, f>0, f>0, indicates NaN

(2) If f=0,s=0, f=0,s=0, f=0,s=0, that means +Infinity

(3) If f=0,s=1, f=0,s=1, f=0,s=1, that means -Infinity

Here we take 20.5 as an example, $(20.5)_D = (10100.1)_b$, $10100.1 = 1.01001 \times 2^4$, so for 20.5, $s = 0$, the biased exponent $e = (4 + 1023)_D = (1027)_D = (000 \quad 0100 \quad 0000 \quad 0011)_B$, and for the fraction we need to hide the highest bit 1, so $f =$ 0100 1000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000, thus 20.5 store as a double type floating point number is 0000 0100 0000 0011 0100 1000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000.

For unnormalized floating point number, e is zero, and f is only fraction part, we can calculate the value $(-1)^s \times (0.f) \times 2^{e-1023}$.

2.3 Analysis error fo IEEE 754 standard

What these 64-bits binary numbers mean? They means the quantization values, so to analyze its error when representing numbers, we just need to find the distance between two adjacent quantization values. And the error can be represent by $error = 2^{e-1023-52}$, so the error will be larger and larger when the number we represent is larger and larger. The max distance is in normalized numbers, because the has exponent, and two adjacent number only have 1 bit different in 52th bit. So the max distance is $d = 2^{2046-1023} \times (0.0000000000001)_H = 2^{2046-1023} \times 2^{-52} = 2^{971}$, so the max error is $d/2 = 2^{970}$, this is incredible big. I didn't believe it at first, and I compute it again and again, I could find where is wrong, so I write a program to test, and I find it is true. I define three double type floating point number: $a = 2^{1023}, b = 2^{1023} + 2^{970}, c = 2^{1023} + 2^{971}$, then I print them out the result can be shown as the Figure 4. We can see that the computer can not distinguish a and b.

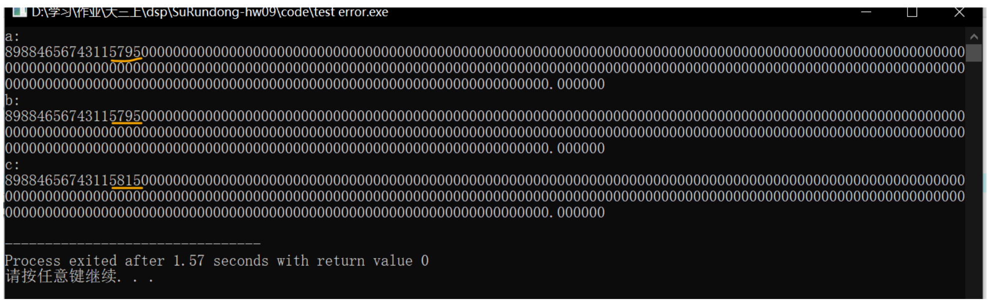


Figure 4: The max error of IEEE 754

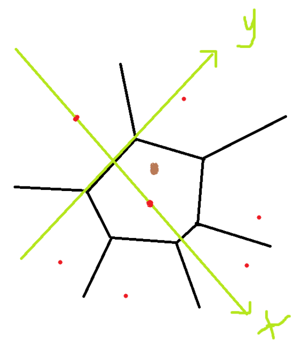


Figure 5: Quantization sections

2.4 Propose and analyze a better way of representation

In our daily using of computer, we may not process too large or too mini numbers, the too large or to mini number are only in some specific sense, so may be we can make a standard for these specific sense, if we want to do with small numbers, we using all memory of a double type float to represent a range of small numbers, when we do with big numbers, we can only memory of a double type float to represent a range of large numbers we want. Or else we can use more memory space to represent a number.

3 Image quantization using Lloyd-Max quantization algorithm

3.1 How to achieve it

A pixel is 3D vector, so that at the beginning, we need to randomly choose 3D vectors as quantization values. The next step we discussed in class is to refine the region boundary b_m, for 3D space it is a plane. To calculate these boundary planes, we need both point on the plane and normal vector of this plane, and when programming, I don't know how to teach our computer when to calculate the plane between which two quantization values(if two quantization is far enough we don't need to calculate it), I think it's out of my league. Then, I take another thinking way: what do these boundary plane do? We use these boundary plane to pixel quantization, if a pixel belongs to one region, then we use the quantization value in this region, so these boundary planes helps us do quantization. Can we do quantization without these boundary plane? Yes, here we take 2 two-dimension as example to state. As the Figure 5 shows, the black lines are boundary of each region, and the red points are quantization values, the brown point P is waiting for quantization, green lines are coordinate axis. We use boundary lines we can judge that P belongs to region 2 rather than region 1, if we calculate the distance between P and q_1, q_2, we can get the same result. $d_1 = \sqrt{(x_p - x_{q_1})^2 + (y_p - y_{q_1})^2}, \quad d_2 = \sqrt{(x_p - x_{q_2})^2 + (y_p - y_{q_2})^2}$, we can see that, if we connect q_1 and q_2 as the X-axis and P is in region 2, the distance on y axis between P and q_1, q_2 are same, but the distance on x axis between P and q_2 is smaller, thus, the distance between P and q_2 is smaller, using the same way we compare distance of P between other quantization value in regions adjacent to region 2 and finally we can get the distance between P and q_2 is minimum so we choose q_2 as its quantization values(also suitable for 3D, we can connect any two quantization value points as X-axis, and let the perpendicular bisector of two points is the yoz plane, we can prove it). Thus, to confirm a quantization value for a pixel, we just need to calculate which quantization is closest to this pixel, and then we can know which region it belongs to.

The next step is to refine boundary plane, but we don't use the boundary plane at all, we need to think another way. The final purpose of refining boundary plane is to refine the quantization values, and we have discuss that the optimal quantization value of a region is the mean value of points in this region, so in this step, we can calculate the mean value of pixels in this region and use it replace the old quantization value. After it, we repeat classify each pixel and each quantization value, until the difference of quantization error between adjacent two operation is small we stop(in program we set 1 at the threshold).

3.2 My discovery

First, the quantization error we define is $J = \int ||Q(x) - x||^2 p(x)dx$, but here comes a problem, like we discussed at the beginning of our class, our computer can only deal with discrete signal, namely the image our computer read is discrete numbers, I don't know to get its pdf $p(x)$. Next make integral for discrete number is zero since the pixel value is finite($[0, 255]$), so in my homework, I use an opportunistic approach, I didn't calculate integral, the integral of a continuous function corresponds to the sum of a discrete function, so I use the sum of two order norm's square between each pixel and its quantization value and then divide the total pixel number as the quantization error. The final result can be shown as the Figure 6. The quantization error and iteration time can be shown as the Figure 7.


(a) M = 2

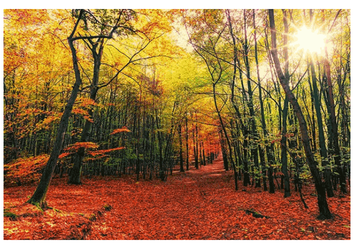

(b) M = 3

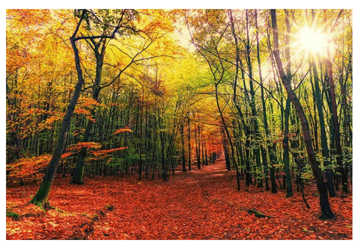

(c) M = 4


(d) M = 5


(e) M = 6


(f) M = 7


(g) M = 8


(h) M = 9

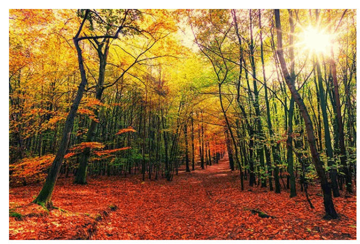

(i) M = 10

Figure 6: Quantization result

I discovered that the more quantization bit we use, it is closer to original image, and the more bit we use, the more color it will have after quantization. Also, the more bit we use, the less error it will have, and the more bit we use, the less iteration time it will have. But, the more bit we use, the more time will be cost, because I run the program from M = 2 to M = 10, and find it.

Better use a table.

```
*************************NO.19 iteration*************************
Done!
Totall error when M = 2:        2396.5848640966424
```
(a) M = 2

```
*************************NO.19 iteration*************************
Done!
Totall error when M = 3:        1623.1906519808138
```
(b) M = 3

```
*************************NO.18 iteration*************************
Done!
Totall error when M = 4:        847.0595274471486
```
(c) M = 4

```
*************************NO.23 iteration*************************
Done!
Totall error when M = 5:        704.454023805294
```
(d) M = 5

```
*************************NO.8 iteration*************************
Done!
Totall error when M = 6:        532.3455249600285
```
(e) M = 6

```
*************************NO.6 iteration*************************
Done!
Totall error when M = 7:        344.7389980458341
```
(f) M = 7

```
*************************NO.4 iteration*************************
Done!
Totall error when M = 8:        269.5115047077634
```
(g) M = 8

```
*************************NO.3 iteration*************************
Done!
Totall error when M = 9:        164.40569905844734
```
(h) M = 9

```
*************************NO.2 iteration*************************
Done!
Totall error when M = 10:        126.2232616805827
```
(i) M = 10

Figure 7: Quantization error and iteration time

4 Derivation of the optimal quantization strategy for $p(x) = 1$

Due to $x \in [0, 1]$, and $p(x) = 1$, so the quantization can be shown as the Equation 1.

$$J = \int_0^1 (Q(x) - x)^2 dx \tag{1}$$

To choose Q with M quantization points, we need insert M-1 points into $[0, 1]$, suppose they are $b_1, b_2, \dots b_{M-1}$, then we let $b_0 = 0, b_M = 1$, suppose that the quantization value of section $[b_{i-1}, b_i]$ is q_i, then in each section, the quantization error can be renewed by Equation 2.

$$\begin{aligned} J_i &= \int_{b_{i-1}}^{b_i} (q_i - x)^2 dx \\ &= \frac{1}{3}(x - q_i)^3 |_{b_{i-1}}^{b_i} \\ &= \frac{1}{3}[(b_i - q_i)^3 - (b_{i-1} - q_i)^3] \end{aligned} \tag{2}$$

Then the whole quantization error can be renewed by Equation 3.

$$\begin{aligned} J &= \sum_{i=1}^{M} J_i \\ &= \frac{1}{3}[(b_1 - q_1)^3 - (b_0 - q_1)^3] + \frac{1}{3}[(b_2 - q_2)^3 - (b_1 - q_2)^3] + \cdots + \frac{1}{3}[(b_M - q_M)^3 - (b_{M-1} - q_M)^3] \end{aligned} \tag{3}$$

We can see that the quantization of each section is independent, so to get the minimum quantization error, we need to set quantization error in each section is minimum. In equation 2, we take the derivative of the left and right sides with respect to q_i, as the Equation 4 shows. To get the minimum quantization error, $J_i^{'}$ should be zero at the point has minimum quantization error, so $-(b_i - q_i)^2 + (b_{i-1} - q_i)^2 = 0$, $q_i = \frac{b_i + b_{i-1}}{2}$.

$$J_i^{'} = -(b_i - q_i)^2 + (b_{i-1} - q_i)^2 \tag{4}$$

Now let's substitute q_i into Equation 2, we get the section quantization can be shown as the Equation 5.

$$\begin{aligned} J_i &= \frac{1}{3}[(b_i - \frac{b_i + b_{i-1}}{2})^3 - (b_{i-1} - \frac{b_i + b_{i-1}}{2})^3] \\ &= \frac{1}{3}[(\frac{b_i - b_{i-1}}{2})^3 - (\frac{-b_i + b_{i-1}}{2})^3] \\ &= \frac{2}{3}(\frac{b_i - b_{i-1}}{2})^3 \end{aligned} \tag{5}$$

We substitute the new J_i into Equation 3, and get the whole quantization error can be shown as Equation 6, where $d_i = b_i - b_{i-1}$ and $\sum_{i=1}^{M} d_i = 1$.

$$\begin{aligned} J &= \sum_{i=1}^{M} J_i \\ &= \sum_{i=1}^{M} \frac{2}{3}(\frac{b_i - b_{i-1}}{2})^3 \\ &= \frac{1}{12} \sum_{i=1}^{M} (b_i - b_{i-1})^3 \\ &= \frac{1}{12} \sum_{i=1}^{M} d_i^3 \end{aligned} \tag{6}$$

Now we use the Lagrange multiplier method to get the condition of J has minimum value, we let $f(d_1, d_2, \cdots, d_M) = J, \phi(d_, d_1, \cdots, d_M) = \sum_{i=1}^{M} d_i - 1$, then we let $L = f(d_, d_1, \cdots, d_M) + \lambda\phi(d_1, d_2, \cdots, d_M)$, and take the derivative of L with respect to $d_1, d_2, \cdots, d_M$ and λ, and let each derivative result equals to zero, we can get the Equation 7.

$$\begin{cases} \frac{1}{4}d_1^2 + \lambda = 0 \\ \frac{1}{4}d_2^2 + \lambda = 0 \\ \qquad \vdots \\ \frac{1}{4}d_M^2 + \lambda = 0 \\ d_1 + d_2 + \cdots + d_M - 1 = 0 \end{cases} \tag{7}$$

Good.

We solve this equation set, we can get $d_1 = d_2 = \cdots = d_M = \frac{1}{M}$, thus $b_1 - b_0 = b_2 - b_1 = \cdots = b_M - b_{M-1} = \frac{1}{M}$. So if we want to fetch the optimal quantization the b_i we insert should be equally spaced.

To summarize, with uniform distribution($p(x) = 1$), the optimal segmentation regions are of equal length, and the quantization value in each subsection should be the mean value of this section($q_i = \frac{b_i + b_{i-1}}{2}$).

References

[1] 冬虫夏草这么恐怖？能绕过宿主大脑将蚂蚁变僵尸. 知否.

[2] 张军朝. *Arduino* 技术及应用. 2017.

[3] http://t.csdn.cn/l27dh. C 数据类型 (bit，byte，word；char，int，long；float，double).

[4] Ieee standard for binary floating-point arithmetic.

[5] Ieee standard for floating-point arithmetic.

Appendix A Code listings for IEEE754 error

```
   #include<stdio.h>
   #include<math.h>
   int main()
   {
      double a, b, c;
      a = pow(2,1023);
      b = pow(2,1023) + pow(2,970);
      c = pow(2,1023) + pow(2,971);
      printf("a:%f\nb:%f\nc:%f\n", a,b,c);
      return 0;
    }
```

Appendix B Code listings for image quantization

```
import numpy as np
import cv2
from numba import jit
from numba import cuda

def initQuantValues(k):
   temp = np.random.randint(0,256,(k,3)) #random choose 0-255 3 dimension vector
   return temp

def classify(xVectors, QuantValues, sectionNum, sectionSum, sectionError):
   labels = []                  #labels are index of quantization vaule correspond to each pixel
   for xVector in xVectors:
      mini = float('inf')       #mini iniial as the maximum value
      index = None
      for i in range(len(QuantValues)):
         dist = np.linalg.norm(np.subtract(xVector,QuantValues[i])) #calculate two order norm
         if dist < mini:                          #if the quantization value is smaller than the old, we take down its index
            mini = dist
            index = i
      labels.append(index)
      sectionNum[index] += 1                      #the pixel number in a region
      sectionSum[index] = np.add(sectionSum[index], xVector)
      sectionError[index] += mini ** 2            #total quantization error in each region
   return labels

def updateQuantValues(QuantValues, sectionSum, sectionNum, secNum): #use mean value of pixels in this region to refine the quantization value in this region
   for i in range(len(QuantValues)):
```

```python
        if sectionNum[i]:
            QuantValues[i] = sectionSum[i] / sectionNum[i]
    sectionNum = np.zeros(secNum)                              #clear sectionNum and sectionSum for next refine
    sectionSum = np.zeros((secNum,3))

def goOnOrNot(lastSectionError, sectionError, pixelNum):
    '''
    if the error difference between to adjacent iteration
    is smaller than 1 we don't iterate anymore
    '''
    temp1 = sum(lastSectionError)
    temp2 = sum(sectionError)
    if (temp1 - temp2)/pixelNum > 10:
            return True
    return False

M = 10   #!!!change M = 2, 3, 4, 5, 6, 7, 8, 9, 10 to see each result
secNum = 2 ** M #number of quantization values
iterationTime = 0
QuantValues = initQuantValues(secNum) #randomly choose at beginning
img = cv2.imread('test_img.jpg')
x,y,z = img.shape
pixelNum = x*y                     #pixel number
xVectors = img.reshape(-1,3) #set the img as (*, 3)vector, * the computer will calculate it automaticly
sectionNum = np.zeros(secNum)
sectionError = np.zeros(secNum) # each section has a quantization error
lastSectionError = np.zeros(secNum)
sectionSum = np.zeros((secNum,3))
flag = True
labels = classify(xVectors, QuantValues, sectionNum, sectionSum, sectionError) # classify each pixel to different regions
updateQuantValues(QuantValues,sectionSum, sectionNum, secNum)    # refine the quantization values
lastSectionError = sectionError                              # last section quantization error
sectionError = np.zeros(secNum)

print(f"************************NO.{iterationTime} iteration************************")
while True:            #iteration until the quantization error between adjacent classify is no larger than 1
    labels = classify(xVectors, QuantValues, sectionNum, sectionSum, sectionError)
    flag = goOnOrNot(lastSectionError, sectionError, pixelNum)
    if flag == False:
        break
    updateQuantValues(QuantValues,sectionSum, sectionNum, secNum)
    lastSectionError = sectionError
    sectionError = np.zeros(secNum)
    iterationTime += 1
    print(f"************************NO.{iterationTime} iteration************************")
print('Done!')

error = sum(sectionError)/pixelNum # final quantization error
print(f'Totall error when M = {M}:\t{error}')
print(QuantValues)
labels = np.array(labels)
QuantValues = np.uint8(QuantValues)
res = QuantValues[labels.flatten()] #transform to quantization image
dst = res.reshape((img.shape)) #anti-reshape opeartion against we have done above
while 1:                 #window wait for show
    cv2.imshow('origin', img)
    cv2.imshow(f'Quantization result M={M}', dst)
    key = cv2.waitKey(1)
    if key == ord('q'):     #press q on keyboard to stop
        break
cv2.destroyAllWindows()
cv2.imwrite(f'img/{M}bits.png',dst) #save image after quantization
```