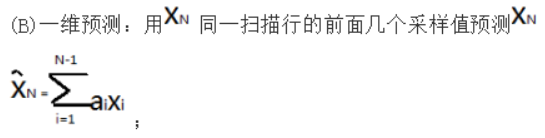
**实验3 数字图像编码实验**

本次作业需要提交以下内容：

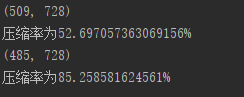
|  |  |
| --- | --- |
| 提交内容 | 详细要求 |
| 作业文档 | 对算法原理进行简单说明；  展示实验效果（每个实验的测试图片不得少于2张，实验用图像自行选择。）；  对实验结果进行分析。 |
| 程序源代码 | 相关程序的全部源代码，要求能够正常编译和运行。 |

**作业一: 无损编码/压缩算法实验**

问题1: 选择灰度图像，按照行的方式展开像得到一维的向量。按照一维预测的公式：，自行设计预测算法实现一维无损预测压缩。将预测压缩后的一维向量（由预测误差组成），进行一维行程/游程编码。计算原图、最终行程/游程编码压缩后数据所需要的存储空间，计算压缩率.

原理：首先计算前N=3个像素和，来预测当前像素，求得误差err，得到压缩后的一维向量，再对其进行行程编码，统计像素连续出现次数，最后进行相应的还原。

|  |  |
| --- | --- |
| 原图 | 还原图像 |
|  |  |
|  |  |



体会：通过一维无损编码和行程编码进行无损压缩，当相同连续像素比较多时，压缩效果较好，并且图像质量没有损失，是很好的无损压缩方式。

**作业二: 有损压缩算法实验**

查阅JPEG编码的有关资料，对图像进行JPEG压缩，算法步骤必须包括如下几个部分：图像分块，离散余弦变换，量化，ac和dc系数的Z字形编排。

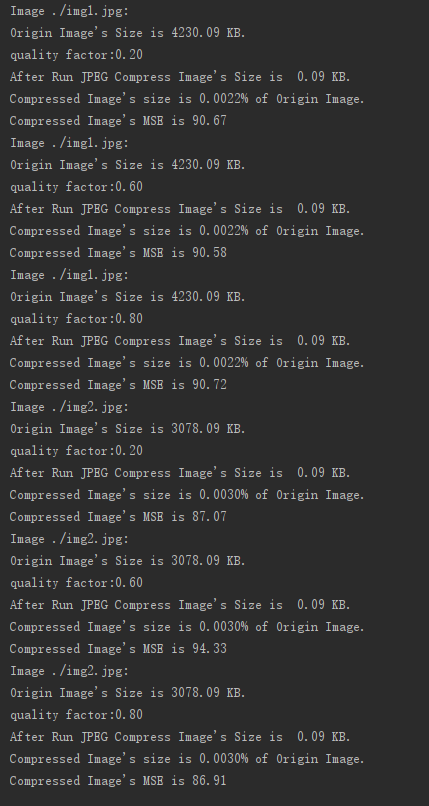
问题1: 质量因子分别选为20，60，80，显示原图与不同质量因子下解码后的图像；

原理：在转换到YCbCr颜色空间后，进行图像的分块，DCT变换，量化，为了保证低频分量先出现，高频分量后出现，以增加行程中连续“0”的个数，对每个量化后的8\*8的系数矩阵采用Zig-Zag扫描排列

左上角第一幅图为原图，右上角第二幅图为质量因子为20下解码后的效果图，左下角第三幅图为质量因子为60下解码后的效果图，右下角第四幅图为质量因子为80下解码后的效果图。

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

问题2: 计算图像压缩前后的压缩比，计算原图与解压图之间的均方根误差。



import cv2  
import numpy as np  
import matplotlib.pyplot as plt  
import random  
from skimage import exposure  
import math  
import os ,sys  
from rel import RLE  
  
  
import cv2  
import numpy as np  
from itertools import groupby  
import sys  
class RLE:  
 def \_\_init\_\_(self):  
 self.path = ''  
  
  
 def matrix2list(self, matirx):  
 *""" 按照行程编码样式将2维数组展开为一维数组 """* mrows, mcols = matirx.shape[:2]  
 mrows -= 1  
 mcols -= 1  
 mlen = min(mrows, mcols)  
  
 rmatrix = []  
 rmatrix.append(matirx[0][0])  
  
 rmatrix.extend(self.first\_encode(matirx, mlen))  
 if mcols > mrows:  
 rmatrix.extend(  
 self.colmore\_middle\_encode(matirx, mlen, mcols, mrows))  
 rmatrix.extend(self.colmore\_last\_encode(matirx, mlen, mcols,  
 mrows))  
  
 else:  
 rmatrix.extend(  
 self.rowmore\_middle\_encode(matirx, mlen, mcols, mrows))  
 rmatrix.extend(self.rowmore\_last\_encode(matirx, mlen, mcols,  
 mrows))  
  
 rmatrix.append(matirx[-1][-1])  
  
 return rmatrix  
  
 def first\_encode(self, matirx, mlen):  
 rmatrix = []  
 for len in range(1, mlen + 1):  
 if (len % 2 == 1):  
 for i in range(0, len + 1):  
 rmatrix.append(matirx[i][len - i])  
 else:  
 for i in range(0, len + 1):  
 rmatrix.append(matirx[len - i][i])  
  
 return rmatrix  
  
 def colmore\_middle\_encode(self, matirx, mlen, mcols, mrows):  
 rmatrix = []  
 if mlen % 2 == 0:  
 for extra in range(mcols - mrows):  
 if extra % 2 == 0:  
 for i in range(mlen + 1):  
 rmatrix.append(matirx[i][mlen - i + extra + 1])  
 else:  
 for i in range(mlen + 1):  
 rmatrix.append(matirx[mlen - i][i + extra + 1])  
 else:  
 for extra in range(mcols - mrows):  
 if extra % 2 == 1:  
 for i in range(mlen + 1):  
 rmatrix.append(matirx[i][mlen - i + extra + 1])  
 else:  
 for i in range(mlen + 1):  
 rmatrix.append(matirx[mlen - i][i + extra + 1])  
  
 return rmatrix  
  
 def colmore\_last\_encode(self, matirx, mlen, mcols, mrows):  
 rmatrix = []  
 if mcols % 2 == 0:  
 for len in range(0, mlen - 1):  
 if len % 2 == 0:  
 for i in range(mlen - len):  
 rmatrix.append(  
 matirx[mlen - (mlen - 1 - len - i)][mlen - i +  
 mcols - mrows])  
  
 else:  
 for i in range(mlen - len):  
 rmatrix.append(matirx[mlen -  
 i][mlen - (mlen - 1 - len - i) +  
 mcols - mrows])  
 else:  
 for len in range(0, mlen - 1):  
 if len % 2 == 1:  
 for i in range(mlen - len):  
 rmatrix.append(  
 matirx[mlen - (mlen - 1 - len - i)][mlen - i +  
 mcols - mrows])  
 else:  
 for i in range(mlen - len):  
 rmatrix.append(matirx[mlen -  
 i][mlen - (mlen - 1 - len - i) +  
  
 mcols - mrows])  
 return rmatrix  
  
 def rowmore\_middle\_encode(self, matirx, mlen, mcols, mrows):  
 rmatrix = []  
 if mlen % 2 == 0:  
 for extra in range(mrows - mcols):  
 if extra % 2 == 1:  
 for i in range(mlen + 1):  
 rmatrix.append(matirx[mlen - i + extra + 1][i])  
 else:  
 for i in range(mlen + 1):  
 rmatrix.append(matirx[i + extra + 1][mlen - i])  
 else:  
 for extra in range(mrows - mcols):  
 if extra % 2 == 0:  
 for i in range(mlen + 1):  
 rmatrix.append(matirx[mlen - i + extra + 1][i])  
 else:  
 for i in range(mlen + 1):  
 rmatrix.append(matirx[i + extra + 1][mlen - i])  
  
 return rmatrix  
  
  
 def rowmore\_last\_encode(self, matirx, mlen, mcols, mrows):  
 rmatrix = []  
 if mrows % 2 == 0:  
 for len in range(0, mlen - 1):  
 if len % 2 == 0:  
 for i in range(mlen - len):  
 rmatrix.append(matirx[mlen - (mlen - 1 - len - i) +  
 mrows - mcols][mlen - i])  
 else:  
 for i in range(mlen - len):  
 rmatrix.append(matirx[mlen - i + mrows -  
 mcols][mlen -  
 (mlen - 1 - len - i)])  
 else:  
 for len in range(0, mlen - 1):  
 if len % 2 == 1:  
 for i in range(mlen - len):  
 rmatrix.append(matirx[mlen - (mlen - 1 - len - i) +  
 mrows - mcols][mlen - i])  
 else:  
 for i in range(mlen - len):  
 rmatrix.append(matirx[mlen - i + mrows -  
 mcols][mlen -  
 (mlen - 1 - len - i)])  
 return rmatrix  
  
 def encode(self, lst):  
 lst\_encode = np.array([(len(list(group)), name)  
 for name, group in groupby(lst)])  
 return lst\_encode.flatten()  
  
  
 def decode(self, lst\_encode):  
 lst = []  
 for i in range(0, len(lst\_encode), 2):  
 print(lst\_encode[i])  
 length = int(lst\_encode[i])  
 for j in range(length):  
 lst.append(lst\_encode[i + 1])  
 return lst  
  
 def compressimg(self, img):  
 r\_img = self.encode(self.matrix2list(img)).astype(np.uint8)  
 return r\_img  
  
 def compress(self, imgpath):  
 image = cv2.imread(imgpath, 1)  
 image = cv2.resize(image, (200, 200))  
 size = sys.getsizeof((image.flatten()))  
 print("Image {}:".format(imgpath))  
 print("Origin Image's Size is {:.2f} KB.".format(size / 1024))  
 [b, g, r] = cv2.split(image)  
  
 r\_b = self.encode(self.matrix2list(b)).astype(np.uint8)  
 r\_g = self.encode(self.matrix2list(g)).astype(np.uint8)  
 r\_r = self.encode(self.matrix2list(r)).astype(np.uint8)  
  
 r\_size = sys.getsizeof((r\_b)) + sys.getsizeof(  
 (r\_g)) + sys.getsizeof((r\_r))  
  
 print(  
 "After Run Length Encoding Image's Size is {:.2f} KB.\nCompressed Image's size is {:.2%} of Origin Image."  
 .format(r\_size / 1024, r\_size / size))  
  
 print()

def show(img, name='img'):  
 cv2.imshow(name, img)  
 cv2.waitKey(0)  
 cv2.destroyAllWindows()  
  
def show\_plt(img):  
 plt.imshow(img, 'gray')  
 plt.show()  
  
  
def compress(path, q\_factor):  
 image = cv2.imread(path, 1)  
 # Step 1: convert rgb image space tp YCrCb space  
 image = cv2.cvtColor(image, cv2.COLOR\_BGR2YCrCb)  
 # 图像尺寸调整，以适应分块  
 height, width = image.shape[:2]  
 if height % 8 != 0 or width % 8 != 0:  
 image = np.pad(image, ((0, (8 - height % 8) % 8), (0, (8 - width % 8) % 8), (0, 0)),  
 "edge")  
 height, width = image.shape[:2]  
 size = sys.getsizeof((image.flatten()))  
  
 print("Image {}:".format(path))  
 print("Origin Image's Size is {:.2f} KB.".format(size / 1024))  
  
 [y, cr, cb] = cv2.split(image)  
 # Step 2: DCT decomposition, transform from time-domain to  
 # frequency-domain, and choose 8\*8 block  
 image\_dct = []  
 for img in [y, cr, cb]:  
 f\_patches = []  
 fi\_patches = []  
 # 图像分块  
 h\_patches = np.vsplit(img, height // 8)  
 for i in range(height // 8):  
 wh\_patches = np.hsplit(h\_patches[i], width // 8)  
 f\_patch = []  
 fi\_patch = []  
 for j in range(width // 8):  
 # DCT 变换  
 patch\_dct = cv2.dct(wh\_patches[j].astype(np.float))  
 f\_patch.append(patch\_dct)  
 f\_patchs = np.hstack(f\_patch)  
 f\_patches.append(f\_patchs)  
 img\_dct = np.vstack(f\_patches)  
 image\_dct.append(img\_dct)  
  
 image\_dct = np.moveaxis(image\_dct, 0, 2)  
  
 # Step 3: 量化  
 image\_dct = np.around(image\_dct / q\_factor)  
 # Step 4: 行程编码，转换为一维数组  
 rle = RLE()  
 [d\_y, d\_cr, d\_cb] = cv2.split(image\_dct)  
 image\_rle = []  
 for dct in [d\_y, d\_cr, d\_cb]:  
 dct\_rle = rle.compressimg(dct)  
 image\_rle.append(dct\_rle)  
  
 # 图像大小计算，压缩比计算  
 r\_size = sys.getsizeof((image\_rle))  
 print("quality factor:{:.2f}".format(q\_factor))  
 print("After Run JPEG Compress Image's Size is {:.2f} KB.\  
 \nCompressed Image's size is {:.4%} of Origin Image.".  
 format(r\_size / 1024, r\_size / size))  
  
 image\_iq = image\_dct \* q\_factor  
 [r\_y, r\_cr, r\_cb] = cv2.split(image\_iq)  
 image\_back = []  
 for img in [r\_y, r\_cr, r\_cb]:  
 f\_patches = []  
 # 图像分块  
 h\_patches = np.vsplit(img, height // 8)  
 for i in range(height // 8):  
 wh\_patches = np.hsplit(h\_patches[i], width // 8)  
 f\_patch = []  
 fi\_patch = []  
 for j in range(width // 8):  
 # IDCT 变换  
 patch\_dct = cv2.idct(wh\_patches[j].astype(np.float))  
 f\_patch.append(patch\_dct)  
 f\_patchs = np.hstack(f\_patch)  
 f\_patches.append(f\_patchs)  
 img\_back = np.vstack(f\_patches).astype(np.uint8)  
 image\_back.append(img\_back)  
 image\_back = np.moveaxis(image\_back, 0, 2)  
  
 # YCrCb 空间转换回 RGB 空间  
 image\_back = cv2.cvtColor(image\_back, cv2.COLOR\_YCrCb2RGB)  
  
 show\_plt(image\_back)  
 mse = ((image - image\_back)\*\*2).mean()  
 print("Compressed Image's MSE is {:.2f}".format(mse))  
  
  
  
imgs = ['./img1.jpg', './img2.jpg']  
for img in imgs:  
 compress(img, 0.2)  
 compress(img, 0.6)  
 compress(img, 0.8)

import cv2

import numpy as np

import matplotlib.pyplot as plt

import random

from skimage import exposure

import math

import os ,sys

from rel import RLE

def to\_gray(path):

img = cv2.imread(path)

img\_gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

return img\_gray

def show(img, name='img'):

cv2.imshow(name, img)

cv2.waitKey(0)

cv2.destroyAllWindows()

def show\_plt(img):

plt.imshow(img, 'gray')

plt.show()

def LPCencode(image, N=3):

data = [0 for \_ in range(len(image))]

for i in range(N):

data[i] = image[i]

for i in range(N, len(image)):

err = sum([image[j] for j in range(i-N, i)])/float(N) - image[i]

data[i] = err

return data

def LPC\_decode(data, N, row, col):

image = [0 for \_ in range(len(data))]

for i in range(N):

image[i] = data[i]

for i in range(N, len(data)):

image[i] = sum([image[j] for j in range(i-N, i)])/float(N) - data[i]

return np.reshape(image, (row, col))

def RLE\_encode(image):

count = 1

data = []

image3 = []

for i in range(len(image)-1):

if count == 1:

image3.append(image[i])

if image[i] == image[i+1]:

count = count + 1

if i == len(image) - 2:

image3.append(image[i])

data.append(count)

else:

data.append(count)

count = 1

if image[len(image)-1] != image[-1]:

image3.append(image[len(image)-1])

data.append(1)

#压缩率

ys\_rate = len(image3)/len(image)\*100

print('压缩率为' + str(ys\_rate) + '%')

return image3, data

# 行程编码解码

def RLE\_decode(image3, data, row, col):

rec\_image = []

for i in range(len(data)):

for j in range(data[i]):

rec\_image.append(image3[i])

rec\_image.append(0)

rec\_image = np.reshape(rec\_image, (row, col))

return rec\_image

imgs = ['./gray1.jpg', './gray2.jpg']

for img in imgs:

image = to\_gray(img)

show\_plt(image)

row, col = image.shape

print(image.shape)

image = image.flatten()

lpc = LPCencode(image)

rle, data = RLE\_encode(lpc)

rec\_lpc = RLE\_decode(rle, data, row, col)

rec = LPC\_decode(lpc, 3, row, col)

# print(rec.flatten())

show\_plt(rec)