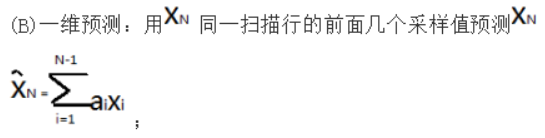
**实验3 数字图像编码实验**

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.

**作业二: 有损压缩算法实验**

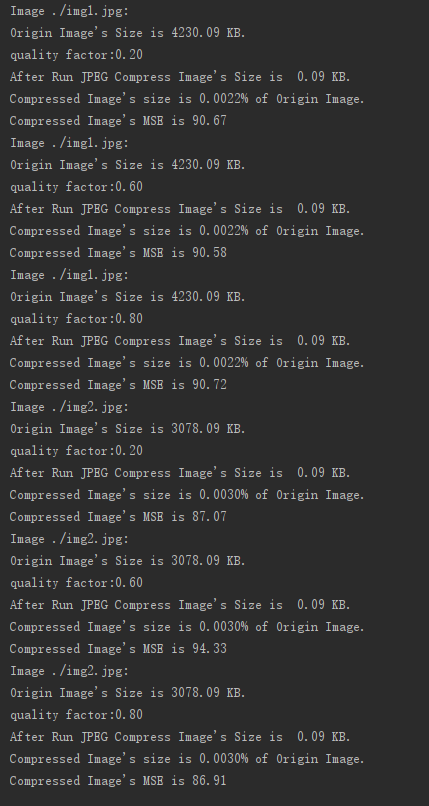
查阅JPEG编码的有关资料，对图像进行JPEG压缩，算法步骤必须包括如下几个部分：图像分块，离散余弦变换，量化，ac和dc系数的Z字形编排。

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；

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  
  
  
  
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)