图像视频编码大实验

陈嘉杰 2017011484

Exp1 Are they equivalent in effect?

子任务1 转换为灰度图片

见代码 `grayscale.py`，直接采用 PIL 的相关函数即可。

原图：

灰度：

子任务2 尝试用不同方式对图片进行 DCT

代码在 `lena_dct_exp1.py` 中。

第一个方法是，先对行再对列进行 DCT ，第二个方法是，对整个图进行 DCT，这两个方法在数学上是等价的，只不过在后面 1/4 1/16 和 1/64 时权值的选取上可以有不一样的结果，后面会继续讨论。

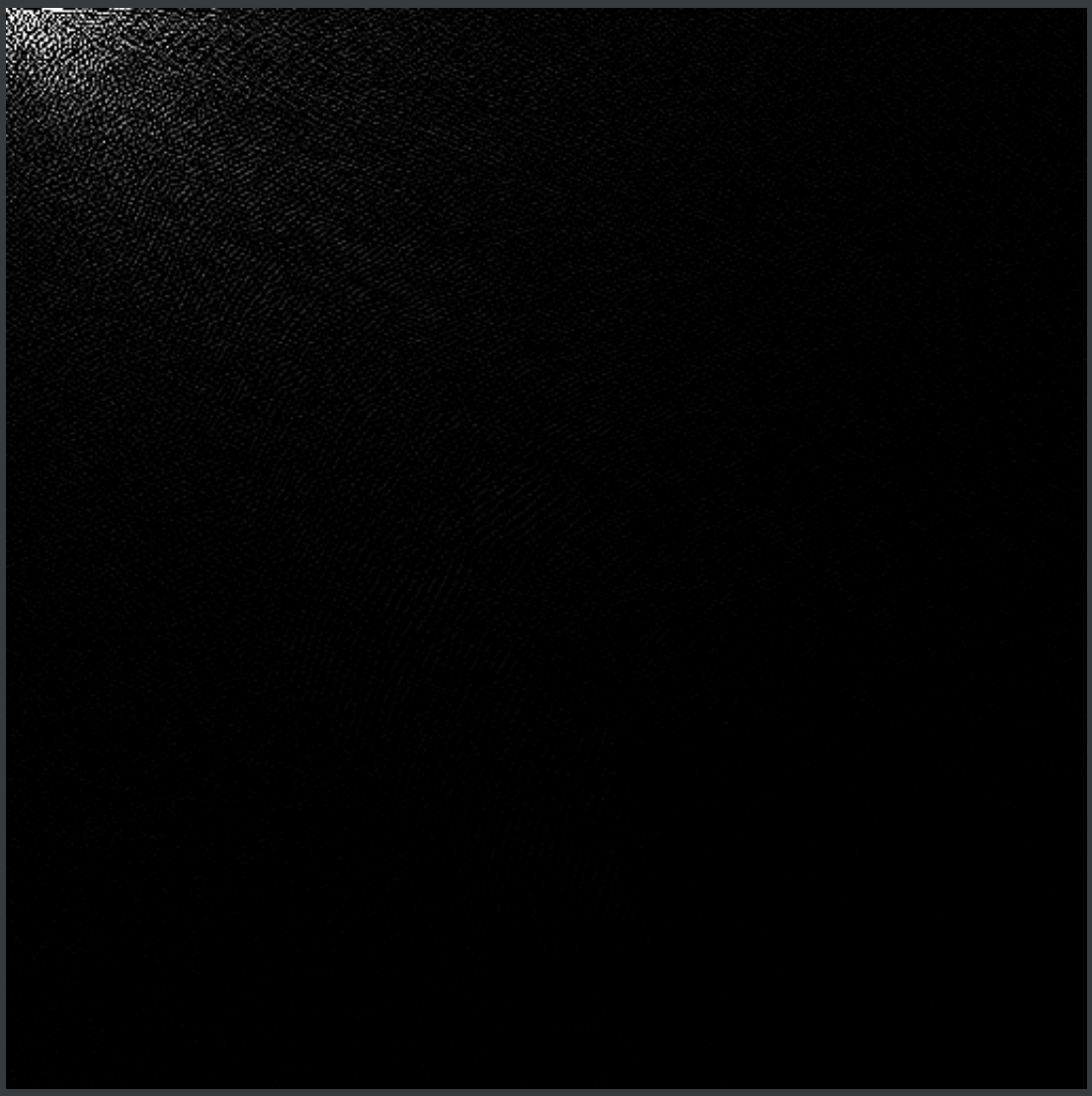
第三个方法则是分割成 8x8 以后进行 DCT。

相关代码：

```
def dct2d(data):
    return fftpack.dct(fftpack.dct(data, norm='ortho').T, norm='ortho').T

def idct2d(data):
    return fftpack.idct(fftpack.idct(data, norm='ortho').T, norm='ortho').T
```

第一个和第二个方法得到的DCT的图：



可以看到，左上角的数值是比较大的，其他地方的数都很小，比较符合 DCT 的特征。

切分为 8x8 以后也有类似的分布：

也是只有左上角一到两个像素比较大。

在运行时间上，设图片都是正方形，边长为 n ，那么第一种方法需要循环 n * n * n * 2 ，第二种方法需要 n * n * n * n 次，第三种方法需要 8 * 8 * 8 * 8 * (n/8) * (n / 8) = 64 * n * n ，当 n 比较大的时候第三种方法最快。

代码输出了对应的 PSNR 值。由于第一种方法和第二种方法在数学上是相等的，代码中只用了第一种方法进行计算，得到 PSNR 为 315.48 ，比第三种方法的 PSNR 315.45 略大，说明考虑到计算精度的时候，第一种方法比第三种方法能留下更精确的信息。

接着对 DCT 之后的系数进行了“压缩”，题目要求 1/4 1/16 和 1/64 ，首先对 8*8 的格子进行了 DCT 系数的选取，方法是，如果是 1/4 ，则选取左上角的 4*4，剩下为零，其它依此类推。再用 IDCT 恢复`到原来的图像，相关代码：

```
def matrix_select(data, side):
    x, y = data.shape
    result = np.zeros(data.shape)
    for i in range(int(x/side)):
        for j in range(int(y/side)):
            result[i,j] = data[i,j]
    return result

```

对比如下：

直接还原：

1/4 的情况：

1/16 的情况：

1/64 的情况：

可以看到，随着压缩率不断增加，图片清晰度也逐渐下降，但仍然保留了比较多原始的信息。由于分块是按照 8x8 的，所以最后 1/64 比例时，每个 8*8 的块都是同一个像素值，显示出了明显的颗粒感。

直接对 2D DCT 进行类似的系数选取后，即对整个图片计算 2D DCT 后，保留左上角的一片系数，剩下都设置为 0，再 IDCT 恢复：

```python
# 1/side^2 coefs
def full_compress(side):
    idct_4 = np.zeros(data.shape)
    dct_4 = matrix_select(dct,side)
    idct_4 = idct2d(dct_4)
    Image.fromarray(idct_4.clip(0,
255).astype('uint8')).save('lena_2ddct_%d_2didct.png' % (side ** 2))
    mse = np.mean((data - idct_4) ** 2)
    psnr = 10 * np.log10(255.0 ** 2 / mse)

full_compress(2) # 1/4
full_compress(4) # 1/16
full_compress(8) # 1/64
```

原始图片：

1/4 的情况：

1/16 的情况：

1/64 的情况：

可以看到，图片压缩率越高，清晰度也在不断下降，但是下降的形式和之前 8*8 时不大一样。因为是直接对全图的 DCT 系数进行压缩，所以在 1/64 的时候看到一些很明显的波纹，这些对应着留下来的部分 DCT 系数。

接下来是采用 PPT 文档中所描述的 DCT 系数选取方法，即先对行进行 DCT ，选取一半的列以后，对这一部分再进行 DCT ，剩余部分都为 0 。按照这样的策略，得到的图片为：

原始图片

1/4:

1/16:

1/64:

可以看到，也出现了一些比较明显的线条，和之前的结果类似。虽然操作顺序不同，但和之前 2D-DCT 最后取左上角的结果是一致的，所以最后得到的图片也是一样的，只是在运行时间上不一样而已。

Exp2 Why quantization is so important?

子任务 1 分块量化并计算平均 PSNR

代码在 `lena_dct_exp2.py` 中。

首先分块为 8x8 的小块，对每一块进行量化，代码如下：

```
def quantize(matrix, data, a):
    qq = a * matrix
    return np.round(data / qq) * qq
```

按照矩阵中对应的值，近似到最近的倍数上，a是Q的系数。通过计算，得到平均的 PSNR 为 38.28（a=1）时

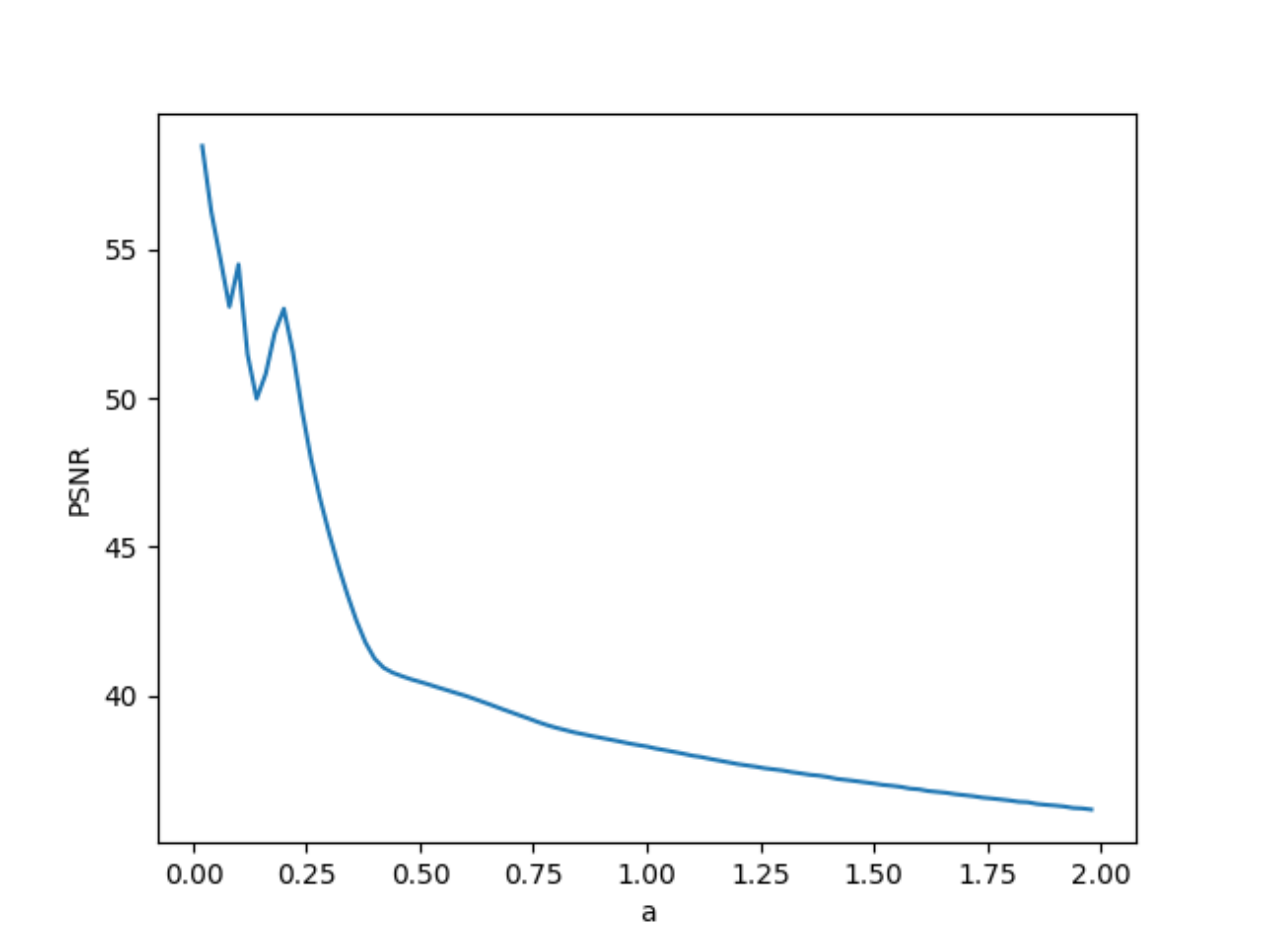
子任务 2 根据不同的 a 得到 PSNR 曲线

代码：

```
for i in range(int(x/8)):
    for j in range(int(y/8)):
        submatrix = data[i*8:(i+1)*8, j*8:(j+1)*8]
        dct = dct2d(submatrix)

        for quan in range(1, 100):
            idct_quan = idct2d(quantize(Q, dct, quan / 50.0))
            psnr_8x8_quan[quan] += psnr(submatrix, idct_quan)
```

延续上面的思路，通过改变 a， 得到不同的 PSNR ，得到图如下：



可以看到，当 a 比较小的时候，此时量化矩阵的系数比较小，所以对原来的 DCT 系数矩阵的值的影响也比较小，所以大趋势是，随着a增大，PSNR减小，失真程度越高。有趣的是，在 a=0.10 和 a=0.20 出出现了两个小的尖峰，可能正好有一些数据在相邻的 a 值下量化到了同一个区间的两边，导致取值偏差较大。

接下来，找了一张图，测试 Canon 和 Nikon 的量化矩阵：

首先灰度处理：

接着按照类似的方法进行量化（代码在 `lena_dct_exp2_2.py` 中），得到：

```
psnr 2ddct 8x8 canon: 50.276670307604974
psnr 2ddct 8x8 nikon: 50.78738331730147
```

可以看到对于这个图片， Nikon 比 Canon 会稍微好一些。

对于量化矩阵的选取，可以看到它里面的数值有的大有的小，小则说明这个位置的 DCT 系数对视觉效果的影响比较大，反之说明影响比较小，量化以后可以得到比较高的压缩率，同时保证人眼看到的样子。所以左上角的数字一般比较小，右下角的数字一般比较大，这和 DCT 的意义是符合的。

写了简单的随机，来获得一个对于上面这个图片 PSNR 比较高的 Q 矩阵：

```
psnr 2ddct 8x8 best: 50.805403292236115
Q: [[ 1.  2.  1.  2.  3.  4.  6.  7.]
 [ 1.  1.  2.  3.  3.  6.  7.  7.]
 [ 2.  1.  2.  3.  4.  9.  8.  9.]
 [ 1.  1.  2.  4.  7. 13.  9.  8.]
 [ 3.  4.  6.  9.  9. 16. 13. 11.]
 [ 3.  4.  6.  8.  9. 12. 14. 10.]
 [ 7. 10. 11. 13. 14. 15. 15. 12.]
 [11. 13. 11. 14. 15. 15. 14. 11.]]
```

其效果也不是很好，并且也只能说明对于当前的这个图片，这个量化矩阵比较适合，但是不能保证它的普遍性，即对于各种图片都有比较好的效果。