数字媒体处理技术实验报告——实验（一）

21R0361班 2021113117-王宇轩

目录

---

1. 实验环境

- 操作系统：Windows
- 编程语言：Python
- 所用软件：PyCharm
- 主要库调用：OpenCV、NumPy、PyWavelets、scikit-image、matplotlib、skimage、wave

2. 文件列表

文件名	内容
BMP.Py	BMP图像的读取，行列像素值、直方图绘制源程序
Permutation&Crop.py	BMP图像的分块置乱和区域裁剪
audio.py	WAV音频的读取和显示，DFT、DCT、DWT变换
imageTransform.py	图像的DFT变换，FFT变换测速，DCT变换压缩复原比较，DWT变换
signals.py	模拟信号实验

文件名	内容
image.bmp	输入的BMP图像
audio.wav	输入的WAV音频
实验报告.pdf	正在被阅读的实验报告
验收视频.mp4	运行实验程序的视频，用于实验验收

3. 实验内容

3.1 BMP图像的读取

对应源代码文件中的**BMP.py**。使用`open()`打开RGB图像，以二进制方式读取文件，参考BMP图像的格式获取图像的各种信息及像素值，首先将像素值存于嵌套数组`pixels[]`，之后分为RGB三通道分别存于`rc`、`gc`、`bc`，代码如下。

```
with open(PATH, 'rb') as f:
    # 读取文件头
    file_header = f.read(14)

    # 读取位图信息头
    bitmap_header = f.read(40)

    # 获取图像的宽度和高度
    width = struct.unpack('<i', bitmap_header[4:8])[0]
    height = struct.unpack('<i', bitmap_header[8:12])[0]

    # 获取像素数据
    f.seek(struct.unpack('<i', file_header[10:14])[0])
    pixels = []
    for h in range(height):
        row1 = []
        for w in range(width):
            b, g, r = struct.unpack('BBB', f.read(3))
            row1.append((r, g, b))
        pixels.append(row1)

    rc, gc, bc = [], [], []
    for row2 in pixels:
        for pixel in row2:
            rc.append(pixel[0])
            gc.append(pixel[1])
            bc.append(pixel[2])
```

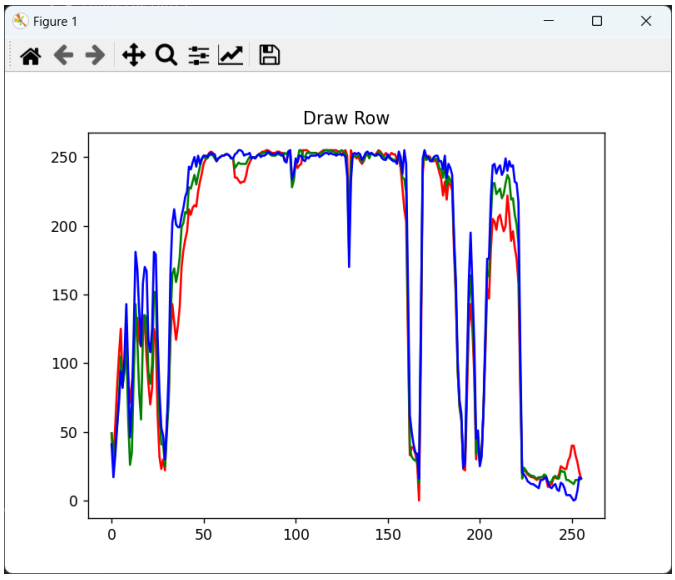
使用`getPixel(x, y)`**获取图像在(x,y)点的RGB三通道像素值**:

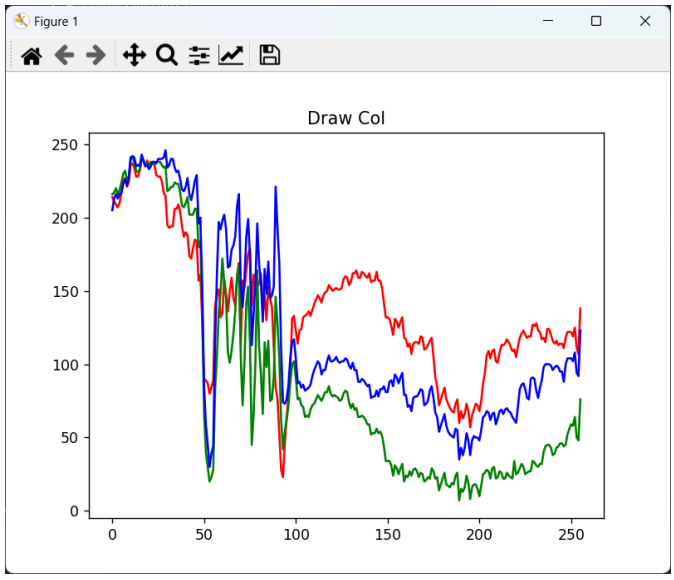
```
def getPixel(x, y):
    return pixels[y][x]
```

使用`drawRow(row)`和`drawCol(col)`分别**画出图像某一行某一列的像素值**:

```
def drawRow(row):
    x = range(width)
    ry, gy, by = [], [], []
    for i in range(width):
        ry.append(getPixel(i, row)[0])
        gy.append(getPixel(i, row)[1])
        by.append(getPixel(i, row)[2])
    plt.plot(x, ry, color='r')
    plt.plot(x, gy, color='g')
    plt.plot(x, by, color='b')
    plt.title("Draw Row")
    plt.show()


def drawCol(col):
    x = range(height)
    ry, gy, by = [], [], []
    for i in range(height):
        ry.append(getPixel(col, i)[0])
        gy.append(getPixel(col, i)[1])
        by.append(getPixel(col, i)[2])
    plt.plot(x, ry, color='r')
    plt.plot(x, gy, color='g')
    plt.plot(x, by, color='b')
    plt.title("Draw Col")
    plt.show()
```

运行结果:

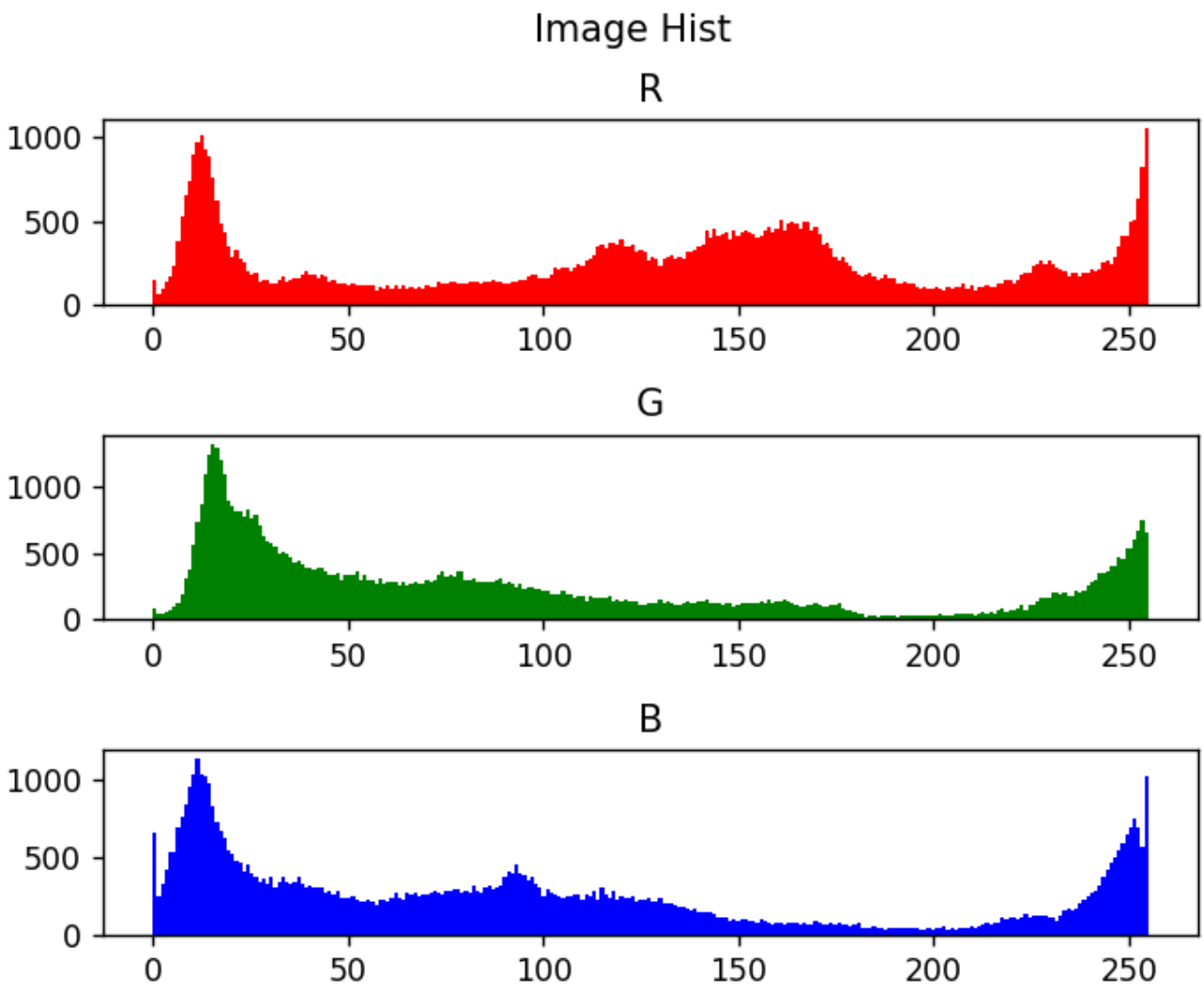




用`getHist()`**统计图像的像素直方图**:

```
def getHist():
    plt.subplot(3, 1, 1)
    plt.hist(rc, bins=256, color='r')
    plt.title("R")

    plt.subplot(3, 1, 2)
    plt.hist(gc, bins=256, color='g')
    plt.title("G")

    plt.subplot(3, 1, 3)
    plt.hist(bc, bins=256, color='b')
    plt.title("B")
    plt.subplots_adjust(hspace=0.7)
    plt.suptitle("Image Hist")
    plt.show()
```

运行结果:



用`getEntropy()`**计算图像的信息熵**:

```
def getEntropy():
    p = [0.0] * 256
    e = 0
    for i in range(256):
        p[i] = rc.count(i) / len(rc)
    for i in range(256):
        e = e - p[i] * math.log(p[i], 2)
    print(f"Red Entropy:{e}")
    for i in range(256):
        p[i] = gc.count(i) / len(rc)
    for i in range(256):
        e = e - p[i] * math.log(p[i], 2)
    print(f"Green Entropy:{e}")
    for i in range(256):
        p[i] = bc.count(i) / len(rc)
    for i in range(256):
        e = e - p[i] * math.log(p[i], 2)
    print(f"Blue Entropy:{e}")
```

运行结果：

```
Red Entropy:7.718921138898001
Green Entropy:15.217933469391227
Blue Entropy:22.807351523905233
```

有关**图像信息熵的物理意义**：

- 信息熵的公式：

$$H(X) = -\sum_{i=1}^{m} p_i(x) log p_i(x)$$

- 一个事件其不确定性越高，熵越大，所包含的信息量越大。对于图像而言，信息熵的大小反映了图像的丰富程度。较高的熵值表示图像像素值分布更加均匀和随机，而较低的熵值表示图像像素值分布更加集中和确定。

3.2 图像的分块置乱及区域裁剪

对应源文件**Permutation&Crop.py**，此部分调用了OpenCV库进行图片的读写，其原理也以BMP.py文件中已实现的方法类似。

使用 `PermutationFun(inputImage, blockwidth, blockheight,seed)` 对图片进行**分块和随机置乱**：

```python
def PermutationFun(inputImage, blockWidth, blockHeight, seed):
    # 读取输入图像
    img = cv2.imread(inputImage)

    # 获取图像的宽度和高度
    height, width = img.shape[:2]

    # 计算图像中块的数量
    num_blocks_x = int(np.ceil(width / blockWidth))
    num_blocks_y = int(np.ceil(height / blockHeight))
    blocks = []

    # 遍历每个块并置乱其位置
    for y in range(num_blocks_y):
        for x in range(num_blocks_x):
            # 计算当前块的左上角和右下角坐标
            x1 = x * blockWidth
            y1 = y * blockHeight
            x2 = min(x1 + blockWidth, width)
            y2 = min(y1 + blockHeight, height)
            # 提取当前块的像素值
            blocks.append(img[y1:y2, x1:x2].copy())
    np.random.seed(seed)
    np.random.shuffle(blocks)

    for y in range(num_blocks_y):
        for x in range(num_blocks_x):
            # 计算当前块的左上角和右下角坐标
            x1 = x * blockWidth
            y1 = y * blockHeight
            x2 = min(x1 + blockWidth, width)
            y2 = min(y1 + blockHeight, height)
            # 将打乱后的像素值写回到图像中
            img[y1:y2, x1:x2] = blocks.pop()

    # 显示置乱后的图像
    cv2.imshow('PermutationFun', img)
    cv2.waitKey(0)
```

运行结果如下：

原始图像	分块置乱图像
	

使用 `CropFun(inputImage, x1, y1, x2, y2, outputImage)` 对图片进行**区域裁剪**：

```
def CropFun(inputImage, x1, y1, x2, y2, outputImage):
    # 读取输入图像
    img = cv2.imread(inputImage)

    # 截取图像的一个区域
    roi = img[y1:y2, x1:x2]

    # 将截取的区域保存为另一幅图像
    cv2.imwrite(outputImage, roi)
```

对图片区域（100，100）、（200，200）进行截取结果如下：

3.3 WAV音频的处理

对应于源文件**audio.py**，使用wave模块**读取wav文件**：

```
with wave.open(PATH, 'rb') as wav_file:
    # 获取采样率和采样宽度
    frameRate = wav_file.getframerate()
    sample_width = wav_file.getsampwidth()

    # 读取PCM音频数据
    pcm_data = wav_file.readframes(-1)

# 将PCM数据转换为NumPy数组
pcm_array = np.frombuffer(pcm_data, dtype=np.int16)

# 将左右声道分别存储到两个数组中
left_channel = pcm_array[::2]
right_channel = pcm_array[1::2]
```

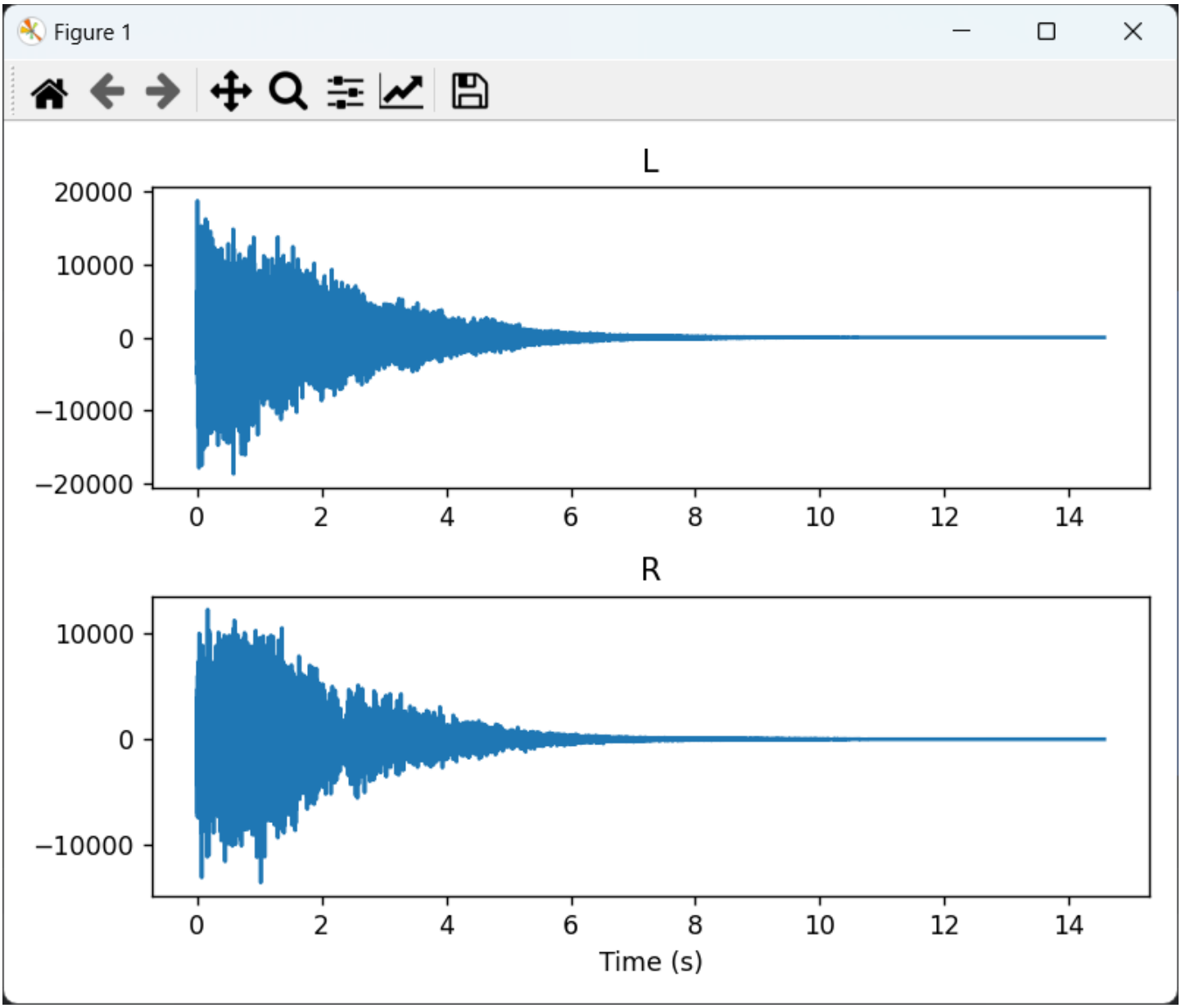
其中，**PCM音频数据**是一种数字音频表示方法，用于将模拟音频信号转换为数字形式，有如下要素：

- **采样率**（Sample Rate）：每秒钟对音频信号进行采样的次数。
- **采样宽度**（Sample Width）：也称位深（Bit Depth），表示每个采样值的精度，用多少比特（bit）来表示一个采样值。
- **声道数**（Channels）：音频信号的通道数。

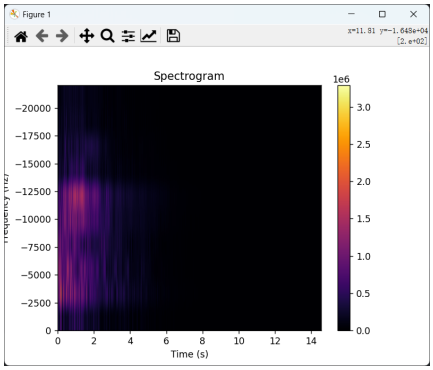
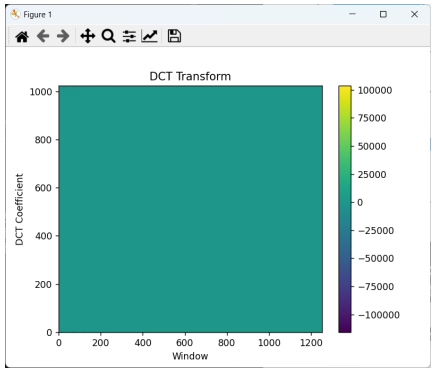
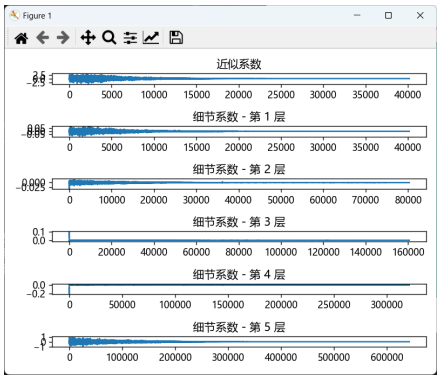
调用matplotlib.pyplot库**显示其一维的PCM数据**：

```
# 计算时间轴
time_axis = np.arange(len(left_channel)) / frameRate

# 绘制波形图
plt.subplot(2, 1, 1)
plt.plot(time_axis, left_channel)
plt.title("L")
plt.subplot(2, 1, 2)
plt.plot(time_axis, right_channel)
plt.title("R")
plt.xlabel('Time (s)')
plt.tight_layout()
plt.show()
```

结果如下：



接下来分别对其进行**DFT、DCT、DWT变换**，源文件audio.py中提供了具体代码，这里就不赘述了，结果如下：

DFT	DCT	DWT
		

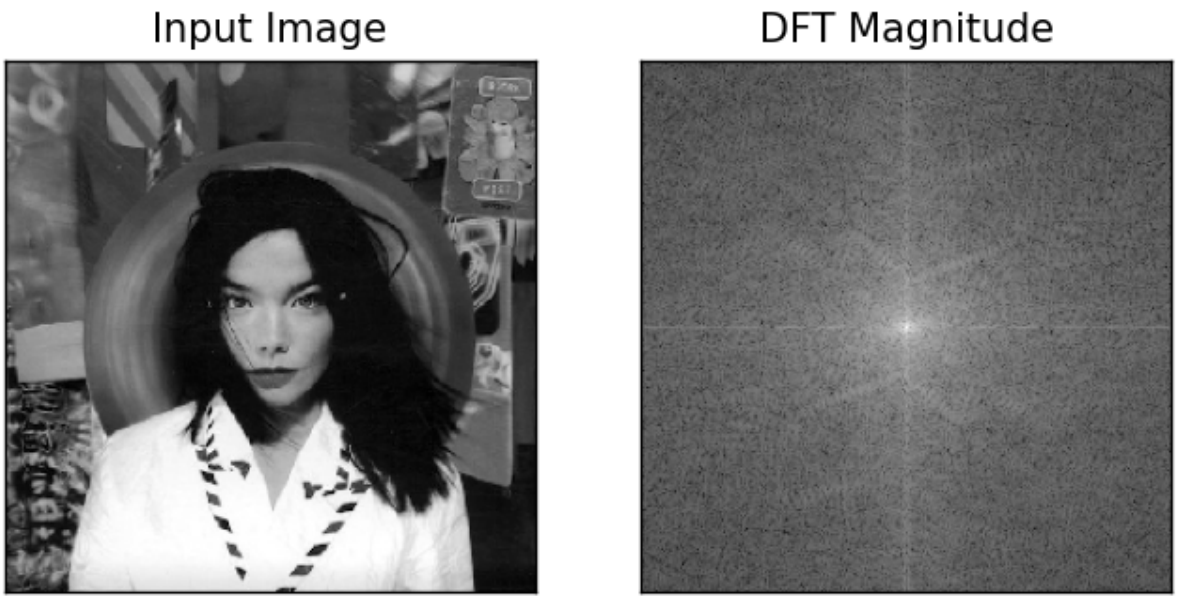
3.4 图片的二维变换

对应源文件**imageTransform.py**。

首先，使用OpenCV库进行图像文件的读取，调用NumPy库中的fft对图片进行傅里叶变换：

```
# 读取图像
image = cv2.imread('image.bmp', cv2.IMREAD_GRAYSCALE)

# 二维 DFT
f = np.fft.fft2(image)
fshift = np.fft.fftshift(f)
magnitude_spectrum = 20*np.log(np.abs(fshift))

plt.subplot(121), plt.imshow(image, cmap='gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122), plt.imshow(magnitude_spectrum, cmap='gray')
plt.title('DFT Magnitude'), plt.xticks([]), plt.yticks([])
plt.show()
```

结果如下：



调用timeit库**对FFT变换进行测速**：

```
def fft():
    np.fft.fft2(image)
# FFT 变换速度
fft_time = timeit.timeit(fft, number=1)
print("FFT变换时间：", fft_time, "秒")
```

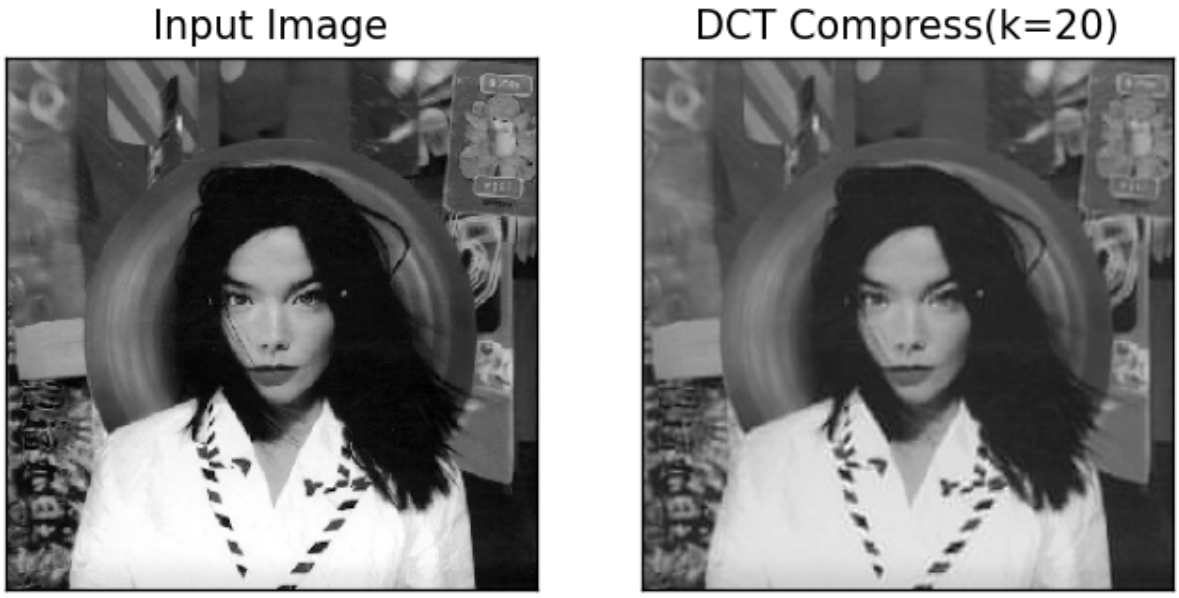
结果如下：

```
FFT变换时间： 0.0012229999992996454 秒
```

接下来，进行图片的**DCT变换**。DCT变换有能量聚集的特性，将低频信号集中在块的左上角，在图像压缩过程中予以保留。此压缩过程的代码如下。其中，我们使用`def zigzag_scan(matrix, k)`和`zigzag_restore(zigzag, rows, cols)`进行Zigzag扫描与恢复。

```python
def compress_image(image, k):
    height, width = image.shape
    compressed_image = np.zeros((height, width), dtype=np.float32)

    for i in range(0, height, 8):
        for j in range(0, width, 8):
            # 图像分块
            block = np.float32(image[i:i+8, j:j+8])
            # DCT变换
            dct_block = cv2.dct(block)
            # 做Zigzag扫描，保留左上角k个数据，进行逆变换
            compressed_block = zigzag_restore(zigzag_scan(dct_block, k), 8, 8)
            idct_block = cv2.idct(compressed_block)
            # 得到压缩后的图像
            compressed_image[i:i+8, j:j+8] = idct_block

    return compressed_image
```

进行DCT变换与复原的结果如下：



接下来，我们**比较原始图像与恢复图像的PSNR值和SSIM值。**

PSNR（Peak Signal-to-Noise Ratio，峰值信噪比）是一个表示信号最大可能功率和影响它的表示精度的破坏性噪声功率的比值的工程术语，用于衡量图像质量，其定义为：

$$PSNR = 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE}\right) = 20 \cdot \log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right)$$

其中，MAX_I表示图像点颜色的最大数值，在这里每个采样点用8bits表示，即255；MSE是均方误差（Mean Squared Error），表示原始图像与重建图像之间的差异：

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2$$

SSIM（Structural Similarity，结构相似性）用于两个图像之间的相似性。人类对像素的绝对亮度/颜色不敏感，但对边缘和纹理的位置非常敏感。SSIM通过主要关注边缘和纹理相似性来模仿人类感知，给定两个图像x和y, 两张图像的结构相似性可按照以下方式求出：

$$\text{SSIM}(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

其中μ_x是x的平均值，μ_y是y的平均值，σ_x^2是x的方差，σ_y^2是y的方差，σ_{xy}是x和y的协方差。$c_1 = (k_1 L)^2$，$c_2 = (k_2 L)^2$是用来维持稳定的常数。L是像素值的动态范围。$k_1 = 0.01$，$k_2 = 0.03$。

对于PSNR和SSIM的计算的实现，可以调用scikit-image库：

```
# 计算 PSNR 和 SSIM，同时指定 data_range
psnr = peak_signal_noise_ratio(image, compressed_image, data_range=255)
ssim = structural_similarity(image, compressed_image, data_range=255)

print('PSNR:', psnr)
print('SSIM:', ssim)
```

计算结果如下：

```
PSNR: 29.64516010129517
SSIM: 0.934139332447191
```

对于**DWT变换**，调用PyWavelets库：

```
# DWT 变换
wavelet = 'haar'  # 小波类型
coeffs = pywt.dwt2(image, wavelet)
# DWT 系数
cA, (cH, cV, cD) = coeffs
# 恢复图像
restored_image = pywt.idwt2((cA, (cH, cV, cD)), wavelet)
```

离散小波变换复原结果如下：

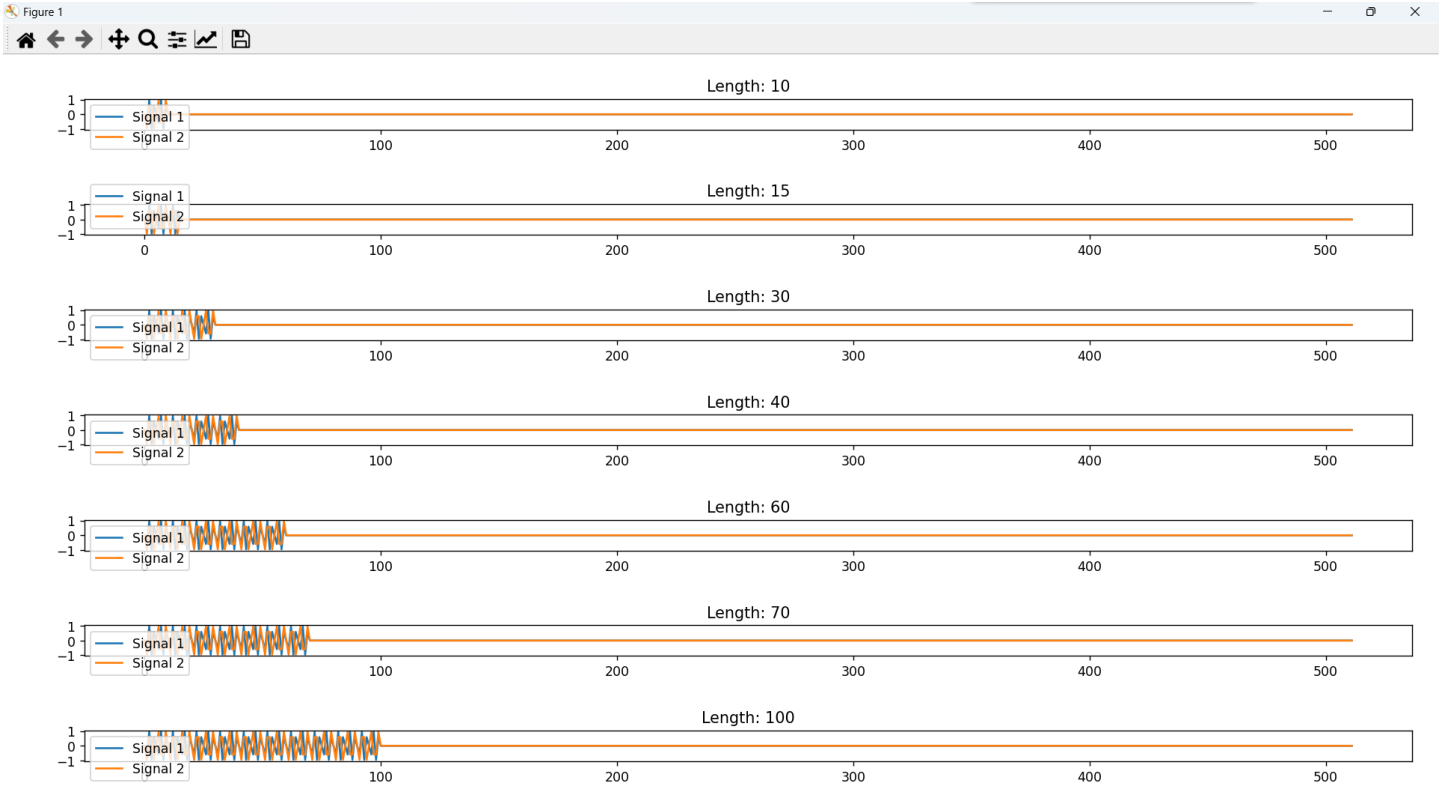
Input Image

DWT Result

3.5 模拟信号实验

```
import numpy as np
import matplotlib.pyplot as plt

# 信号频率
f1 = 30
f2 = 35

# 采样区间和采样频率
T = 0.02
fs = 1 / T

# 数据长度
lengths = [10, 15, 30, 40, 60, 70, 100]

# 填充长度
fill_length = 512

# 生成时间序列
t = np.arange(0, fill_length, 1 / fs)

# 初始化图像
fig, axs = plt.subplots(len(lengths), 1, figsize=(8, 6))

# 逐个数据长度进行采样和填充
for i, length in enumerate(lengths):
    # 生成信号
    signal1 = np.sin(2 * np.pi * f1 * t[:length])
    signal2 = np.sin(2 * np.pi * f2 * t[:length])

    # 零填充
    signal1_padded = np.pad(signal1, (0, fill_length - length), 'constant')
    signal2_padded = np.pad(signal2, (0, fill_length - length), 'constant')

    # 绘制信号图像
    axs[i].plot(signal1_padded, label='Signal 1')
    axs[i].plot(signal2_padded, label='Signal 2')
    axs[i].set_title(f'Length: {length}')
    axs[i].legend()

plt.tight_layout()
plt.show()
```

结果如下：



4. 实验总结

学会了使用OpenCV及python完成对数字媒体文件进行基础的读写、变换，掌握了一些常见文件的格式。