《生物医学图像处理实验》实验报告

(2021-2022 学年第 2 学期)

实验四：图像的二维傅里叶变换及变换域滤波

课程号：310025010　　**课序号**：01　　**任课教师**：林江莉　　**成绩**：

组长：徐广玄　　**小组成员**：**无**

实验日期：2023 年 4 月 30 日

一、实验内容：

1、 掌握 FFT 和 IFFT 的定义，使用 C 语言（或基于 matlab 语句）编写 FFT 和 IFFT 的程序，能对一幅二维数字图象进行移中和变换。(注意不能直接调用 matlab 的 fft 函数)

2、 编写理想滤波器和巴特沃斯低通滤波器函数（阶数可设），并对比两者滤波的效果。

3、 编写理想滤波器和巴特沃斯高通滤波器函数（阶数可设），并对比两者锐化图像的效果。

4、 编写同态滤波函数，并实现“PET_image.bmp”和“factory.bmp”图像的同态滤波。

注意：逐渐实现**能对 8 位，16 位，24 位，32 位图像进行操作。**

二、编程软件：

Python3.9，基于 pycharm，不使用 opencv

三、实验步骤、结果及关键代码：

1、FFT 和 IFFT

在这个实现中，我们首先定义了一个 ifft2d() 函数来计算二维灰度图像的逆 FFT。函数输入参数是一个二维的 Numpy 数组，表示一个灰度图像的 FFT 结果。函数首先对每一行进行一维的逆 FFT，然后对每一列进行一维的逆 FFT。我们使用 numpy.empty_like() 函数创建一个空的 Numpy 数组来保存逆 FFT 的结果，使用 dtype=np.complex128 来指定数据类型为复数类型。在这个函数中，我们调用了 ifft() 函数来计算一维的逆 FFT。在 ifft() 函数中，我们使用了 Cooley-Tukey 算法来计算一维的逆 FFT。输入参数 x 是一个一维的 Numpy 数组，输出结果是逆 FFT 后的结果。这个实现与之前提供的一维 FFT 实现类似，使用递归方法将序列分成偶数和奇数部分，并将它们组合起来。函数使用了 Numpy 的广播机制来避免一些重复计算。最终，我们得到了一个二维的复数数组，表示输入图像的空域表示。我们将结果进行缩放并取实部，得到了灰度图像的逆 FFT 结果。

```python
def fft2d(img):
    """Compute the 2D FFT of a grayscale image"""
    rows, cols = img.shape
    # Compute the FFT of each row
    temp = np.empty((512, 512), dtype=np.complex128)
    for i in range(rows):
        temp[i, :] = fft(img[i, :])
    # Compute the FFT of each column
    result = np.empty((512, 512), dtype=np.complex128)
    for j in range(cols):
        result[:, j] = fft(temp[:, j])
    result = abs(result)
    print(result)
    for i in range(512):
        for j in range(512):
            result[i][j] = int(result[i][j])
            if result[i][j] > 255:
                result[i][j] = 255
            elif result[i][j] < 0:
                result[i][j] = 0
    print(result)
    plt.subplot(421)
    plt.imshow(result, cmap='gray')
    plt.title("fft")
```

图 1 二维 fft 实现

```python
def fft(x):
    """Compute the 1D FFT of x using Cooley-Tukey algorithm"""
    x = zero_fill(x)
    n = x.shape[0]
    if n == 1:
        return x
    else:
        even = fft(x[0::2])
        odd = fft(x[1::2])
        factor = np.exp(-2j * np.pi * np.arange(n) / n)
        return np.concatenate([even + factor[:n//2] * odd, even + factor[n//2:] * odd])
```

图 2 一维 fft 实现

```python
def zero_fill(x):
    n = x.size
    m = n
    cnt = 0
    res = 1
    while res < n:
        res = res << 1
    if res == n:
        return x
    else:
        while m != 0:
            m = m // 2
            cnt += 1
        num = 2 ** cnt - n
        x = np.pad(x, (0, num), 'constant', constant_values=(0, 0))
        return x
```

图 3 fft 补位

2、理想滤波器和巴特沃斯低通滤波器函数

对于理想滤波器和 butterworth 滤波器来说，需要将时域图转换到频域，然后乘一个截断函数，再转回时域。

```
def ILPF(image, d0):
    img = image
    f = np.fft.fft2(img)
    fshift = np.fft.fftshift(f)
    rows, cols = img.shape
    for i in range(rows):
        for j in range(cols):
            if int(((i - rows / 2) ** 2 + (j - cols / 2) ** 2) ** 1/2) <= d0:
                fshift[i][j] = fshift[i][j]
            else:
                fshift[i][j] = 0
    ishift = np.fft.ifftshift(fshift)
    iimg = np.fft.ifft2(ishift)
    iimg = np.abs(iimg)
    plt.subplot(423), plt.imshow(iimg, 'gray'), plt.title('ILPF')
```

图 4 理想滤波器实现

```
def BWLPF(image, d0, n):
    img = image
    f = np.fft.fft2(img)
    fshift = np.fft.fftshift(f)
    rows, cols = img.shape

    for i in range(rows):
        for j in range(cols):
            d = ((i - rows / 2) ** 2 + (j - cols / 2) ** 2) ** 1 / 2
            fshift[i][j] *= 1 / (1 + (d / d0) ** (2 * n))

    ishift = np.fft.ifftshift(fshift)
    iimg = np.fft.ifft2(ishift)
    iimg = np.abs(iimg)
    plt.subplot(424), plt.imshow(iimg, 'gray'), plt.title('BWLPF')
```

图 5 butterworth 滤波器实现

3、编写理想滤波器和巴特沃斯高通滤波器函数

和 2 同理，仅需改变判断符号和 d 和 d0 的顺序即可。

4、编写同态滤波函数

对函数取以 e 为底的对数，然后进行类 butterworth 滤波器的操作，然后取以 e 为底的幂。

```python
def HF_1(img, cutoff_freq, order):
    img_log = np.log1p(np.array(img, dtype="float") / 255)
    img_fft = np.fft.fft2(img_log)
    H = np.zeros_like(img_fft)
    rows, cols = img_fft.shape

    for i in range(rows):
        for j in range(cols):
            H[i, j] = (1 - np.exp(-order * ((i - rows / 2)**2 + (j - cols / 2)**2) / (2 * cutoff_freq**2)))

    img_filtered = np.real(np.fft.ifft2(img_fft * H))
    img_exp = np.expm1(img_filtered) * 255
    img_out = np.uint8(np.clip(img_exp, 0, 255))

    plt.subplot(427), plt.imshow(img_out, 'gray'), plt.title('HF_1')
```

图 6 同态滤波函数实现

四、实验结果：

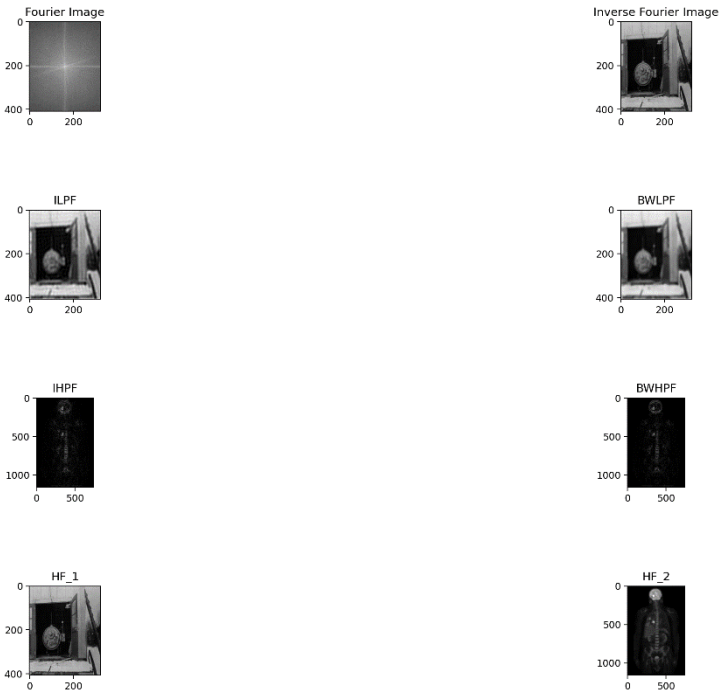

图 7 实现结果

如图可以看出可以看到 fft 之后的图像复原和之前的图像仍有细微区别；对理想和 butterworth 滤波器取不同的 d0 和参数都会对时域结果有一定的影响；使用同态滤波的带通函数使得图像增强效果最好。

五、结果分析及实验小结：

编程中问题及其解决：fft 实现时数组报错，解决方法：对数组末尾补零，使得其为二的幂。

编程中的注意事项：注意数组的大小

六、小组成员分工

由徐广玄独立完成