

图像处理与模式识别 技术报告

大作业主题：离散傅立叶变换与离散余弦变换
游子诺 17373321

图像处理与模式识别 技术报告

摘要

1 算法实现

- 1.1 DFT算法实现
- 1.2 DCT算法实现
- 1.3 频域数据的可视化

2 功能应用

- 2.1 低通滤波器与噪音消除
- 2.2 高通滤波器与图像锐化
- 2.3 滤波器综合运用的人像美化
- 2.4 DFT应用——图像放大与缩小（失败案例）
- 2.5 DCT应用 - 图像压缩
- 2.6 DCT应用 - 数字水印

3 结论

摘要

傅立叶变换（Fourier Transformation）是信号处理的基础理论，其表示任何时域或空域信号都能够一组三角函数的正交基通过线性变换组合得到，实现从时域（空域）到频域的转换，傅立叶反变换则发挥与FT相反的功能。

利用FT的正反变换，能够实现许多在单域中无法实现的功能。FT运用在计算机的实际处理时，根据其实际情况又衍生了离散傅立叶变换（DFT）以适应实际离散的计算；而若选择的核函数不带有复数，则实现了仅在实数范围内的更快运算，由此产生了离散余弦变换（DCT）。将图像作为信号进行二维DFT、DCT得到的频域，能够对图像中的高频信号（相邻信号差值大，即空域中的边缘）和低频信号很好地划分开，许多图像处理技术也是基于此原理展开的。

本报告将从算法实现和应用功能两个方面进行阐述，算法实现关注如何编写高效的转换算法，而应用功能则从尝试的角度对多种FT相关的领域进行简单实验，并针对实验的成功情况提出问题或解决方案。
应用实验的完整结果可通过jupyter笔记本查看。

程序语言：Python 3

包依赖：

- numpy：矩阵与复数运算支持，未直接调用封装的傅立叶变换函数。
- matplotlib.pyplot：绘图
- PIL.Image：图像读取与灰度图生成

实验图片：200x200或300x300灰度图像（受限于DFT和DCT变换的复杂度，未使用高分辨率的图片）。

- 算法实现：

- 基于Numpy矩阵运算的DFT和DCT运算。
- 频谱可视化显示。

- 应用功能：

- DFT：低通滤波器与噪音消除。
- DFT：高通滤波器与图像锐化。
- DFT：滤波器综合运用的人像美化。
- DFT：图像缩放与栅栏效应。
- DCT：图像压缩。
- DCT：数字水印。

1 算法实现

1.1 DFT算法实现

原理：对图像的DFT变换是逐维变换的，若以行优先进行变换，则假设行向量大小 m 恰好为周期函数的一个周期，在划分后了核函数的范围后，以向量的每个值为采样点，尝试求解一个 m 元的线性方程，获得线性组合量 c ， c 是复数形式，能够携带频域中的幅度信息和相位信息。详细的公式推导与原理解释不在此赘述，在此推荐一篇理论和实践结合的博文[知乎](#)。

DFT公式： $F(u, v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) e^{-j2\pi(\frac{ux}{M} + \frac{vy}{N})}$ （其中， $u = 0, 1, 2, \dots, M-1; v = 0, 1, 2, \dots, N-1$ ）

对于固定 u 和 v 的情况下，由其公式满足 $\sum_x \sum_y f(x, y) g(x) h(y)$ 的形式，因此能够使用矩阵运算 $a^T M b$ 进行求解，其核心内容如下：

```
1 # 由于Python的运行复杂度比较高，相较于直接使用n^4循环的复杂度，n^2+矩阵运算能够大幅度节省时间。
2 for _m in range(0, m):
3     # a 向量
4     row_vector = np.exp(-2.0 * np.pi * 1j * np.arange(0, m, 1.0) * _m / m)
5     for _n in range(0, n):
6         # b 向量
7         col_vector = np.exp(-2.0 * np.pi * 1j * np.arange(0, n, 1.0) * _n /
8         n)
9         # time_domain: 原图像
10        fre_result[_m][_n] = np.dot(np.dot(row_vector, time_domain_2d),
11        col_vector)
```

DFT反变换公式： $f(x, y) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u, v) e^{j2\pi(\frac{ux}{M} + \frac{vy}{N})}$

同样的可以使用矩阵运算进行

```

1 space_domain = np.zeros((m, n)).astype(complex)
2 for _m in range(0, m):
3     row_vector = np.exp(2 * np.pi * 1j * np.arange(0, fre_m, 1.0) * _m / m)
4     for _n in range(0, n):
5         col_vector = np.exp(2 * np.pi * 1j * np.arange(0, fre_n, 1.0) * _n
6 / n)
6         space_domain[_m][_n] = np.dot(np.dot(row_vector, fre_domain_2d),
col_vector)

```

Fourier类的设计：

我在编写代码时，将离散傅立叶变换的相关操作封装在了Fourier类中，并为后续的功能操作预留了接口，具体的方法与成员定义如下：

```

1 class Fourier:
2     def __init__(self, array_2d):
3         self.space_domain = array_2d
4         self.frequency_domain = self.fourier_transform(self.space_domain)
5         # cache 缓存使用滤波器后的空域和频域数据
6         self.cache_filtered_frequency = self.frequency_domain
7         self.cache_space_domain = None
8
9     # 在频域上应用滤波器，传入函数参数和参数
10    def apply_frequency_filter(self, func, **kwargs)
11        # 获取原始频域经过转化的空域数据
12        def get_raw_frequency_domain(self)
13            # 获取原始频域数据
14            def get_filter_frequency_domain(self)
15            # 获取滤波器后的时域数据
16            def get_raw_space_domain(self)
17            # 获取滤波器后的空域数据
18            def get_filter_space_domain(self, real=True)
19
20        # 傅立叶正变换
21        def fourier_transform(time_domain_2d)
22        # 傅立叶逆变换
23        def inverse_fourier_transform(fre_domain_2d, size=None)

```

1.2 DCT算法实现

原理：DCT的实现与DFT实现类似，通过 $a^T Mb$ 的矩阵乘法进行。

$$\text{DCT公式: } F(u, v) = \frac{2}{\sqrt{MN}} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} C(u)C(v)f(x, y) \cos\left(\frac{(2x+1)u\pi}{2M}\right) \cos\left(\frac{(2y+1)v\pi}{2N}\right)$$

```
1 # get_c函数为C(x)函数
2 for _m in range(0, m):
3     # 生成行向量 a
4     row_vector = get_c(_m)*np.cos((2*np.arange(0,m,1)+1)*_m*np.pi/(2*m))
5     for _n in range(0, n):
6         # 生成列向量 b
7         col_vector =
8             get_c(_n)*np.cos((2*np.arange(0,n,1)+1)*_n*np.pi/(2*n))
9         fre_result[_m][_n] = np.dot(np.dot(row_vector, time_domain_2d),
col_vector)
fre_result = fre_result * 2 / np.sqrt(n * m)
```

DCT反变换公式: $f(x, y) = \frac{2}{\sqrt{MN}} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} C(u)C(v)F(x, y) \cos\left(\frac{(2x+1)u\pi}{2M}\right) \cos\left(\frac{(2y+1)v\pi}{2N}\right)$

```
1 # DCT反变换与DCT类似，在实现时需要注意的是累计变量和DCT中相反的。DCT中为(x,y)而反DCT  
2 # 中为(u,v)。  
3  
4 for _m in range(0, m):  
5     c_m = np.ones(fre_m)  
6     c_m[0] = get_c(0)  
7     row_vector = c_m*np.cos((2*_m+1)*np.arange(0,fre_m,1)*np.pi/(2*m))  
8     for _n in range(0, n):  
9         c_n = np.ones(fre_n)  
10        c_n[0] = get_c(0)  
11        col_vector = c_n*np.cos((2*_n+1)*np.arange(0,fre_n,1)*np.pi/(2*n))  
12        space_domain[_m][_n] = np.dot(np.dot(row_vector, fre_domain_2d),  
13                                         col_vector)
```

Cosine类设计：与Fourier设计原理类似的，封装空域和频域的转换方法，并允许在频域上加载自定义的滤波器。

1.3 频域数据的可视化

DFT

对于经过变换得到的频域数据仍旧是复数形式，对数据的可视化存在很多形式。由于频域中能够很好地表示信号高低频的情况，因此常用信号的强度——幅度谱作为频域数据的可视化。

幅度的计算公式为 $F(u, v) = \sqrt{R^2(u, v) + I^2(u, v)}$, 利用numpy中的abs()方法能够直接得到, 在得到幅度谱后, 将其打印为PPT上常见的形式还需要如下的操作:

- 可视化范围：matplotlib的imshow函数支持的数字大小为(0-1)或(0-255)，然而所求出的幅度谱很大地超出了这个这两个范围。因此，经过查阅资料，我选择了使用 $\log(x+1)$ 和归一化处理。

```
1 # reader.py
2 def get_visual_frequency_domain(frequency_domain, center=True,
3 log=True):
4     """
```

```

4     Transform Frequency Domain Image to a visible format. [0 - 1]
5     :param frequency_domain: Frequency Domain
6     :param center: if True, reverse to highlight center
7     """
8     # 得到可视化范围
9     if log:
10        f = np.log1p(np.abs(frequency_domain))
11    else:
12        f = np.abs(frequency_domain)
13
14    f = (f - np.min(f)) / (np.max(f) - np.min(f))
15    # 是否等分为4部分进行对角线翻转
16    if center:
17        f = mirror(f)
18    return f

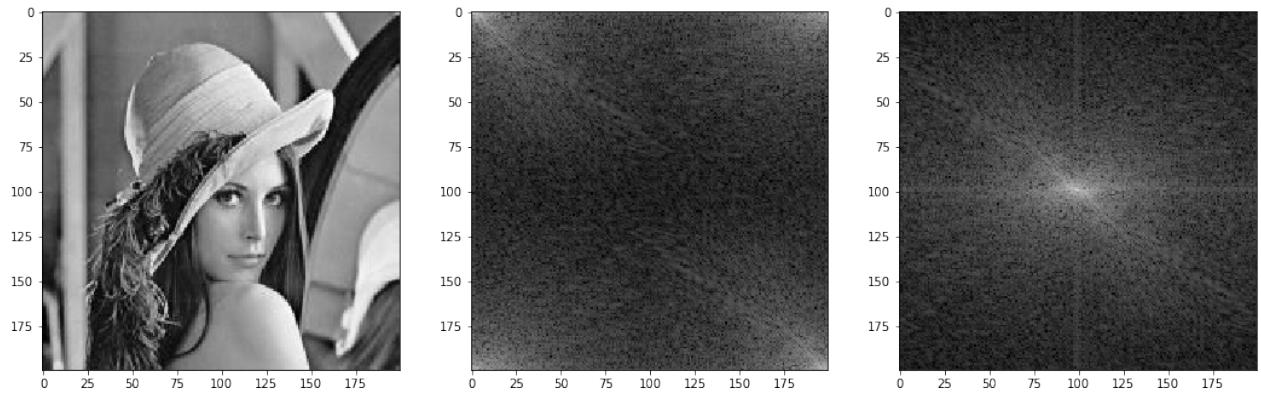
```

- **翻转**: 在图像中，低频信息多而高频信息少，因此对应低频信号的(u,v)——边缘部分（特别是四个角）会存在高亮度的情况，之所以是四个角是因为线性组合的解是共轭的，因此存在上下左右的堆成关系。为了将高能量集中于一处，需要将图形等大小拆分为4部分并对角线翻转。

```

1  # utils.py
2  def mirror(array):
3      """
4          Apply mirror transform to 2D array.
5          Application Area: 频谱图可视化, 滤波器应用
6          HINT: 此操作是可逆的, 两次应用后可以恢复原始情况。
7          :return:
8          """
9
10         f = array
11         width_half = f.shape[1] // 2
12         height_half = f.shape[0] // 2
13         # 先上下翻转, 再左右翻转
14         left_up = np.fliplr(np.flipud(f[0:height_half, 0:width_half]))
15         left_down = np.fliplr(np.flipud(f[height_half:, 0:width_half]))
16         right_up = np.fliplr(np.flipud(f[0:height_half, width_half:]))
17         right_down = np.fliplr(np.flipud(f[height_half:, width_half:]))
18         # 拼接
19         up = np.concatenate([left_up, right_up], axis=1)
20         down = np.concatenate([left_down, right_down], axis=1)
21         f = np.concatenate([up, down], axis=0)
22     return f

```

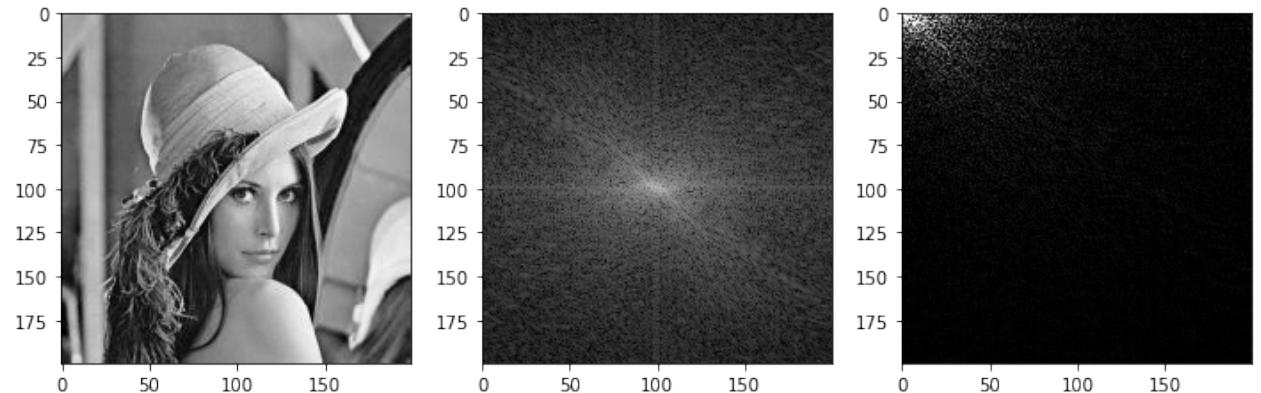


DCT

DCT依旧采用频谱图进行可视化，并使用对数和归一化处理求出(0-1)的可视化区间。但是由于其解空间不满足DFT的对称性质，因此并没有进行翻转操作，而是保留低频信息留在图像的左上角位置。

DFT和DCT的频谱图对比

下图占了Lena图片经过两种变换得到的结果，可以看到DCT的幅度比DFT更加集中，图像能够借助更少的信息表征更多的内容，在图像压缩方面有比较强的潜力。

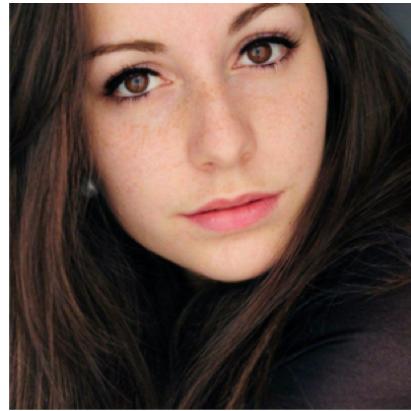


2 功能应用

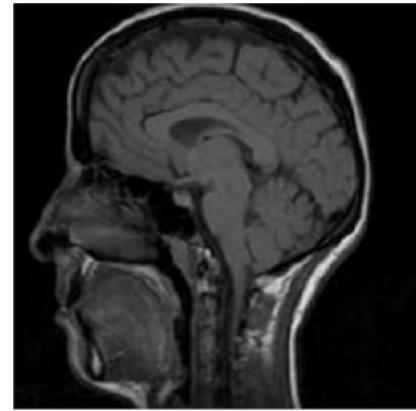
实验中所实验的素材主要有以下三张图片，此外，还包括一些使用矩阵手动生成的图片。



200x200
所有实验



300x300
人像美容实验



300x300
数字水印实验

2.1 低通滤波器与噪音消除

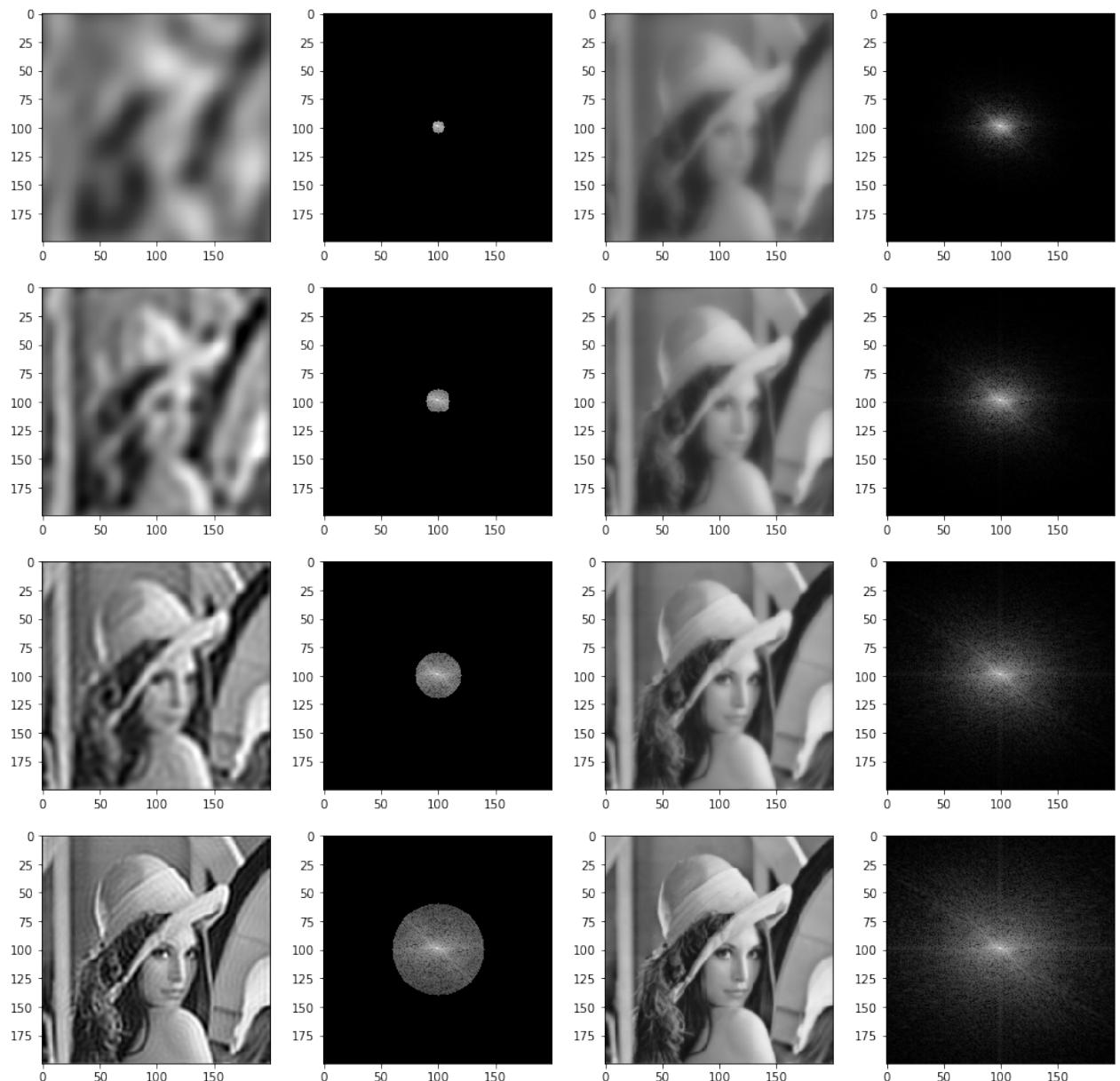
在实验中，我尝试了理想低通滤波器和**ButterWorth**滤波器，这类滤波器通过进一步弱化高频率的信息，达到图像模糊的效果。

理想LPF的公式表示为： $L(u, v) = D_0(u, v) \leq D(u, v) ? 1 : 0$

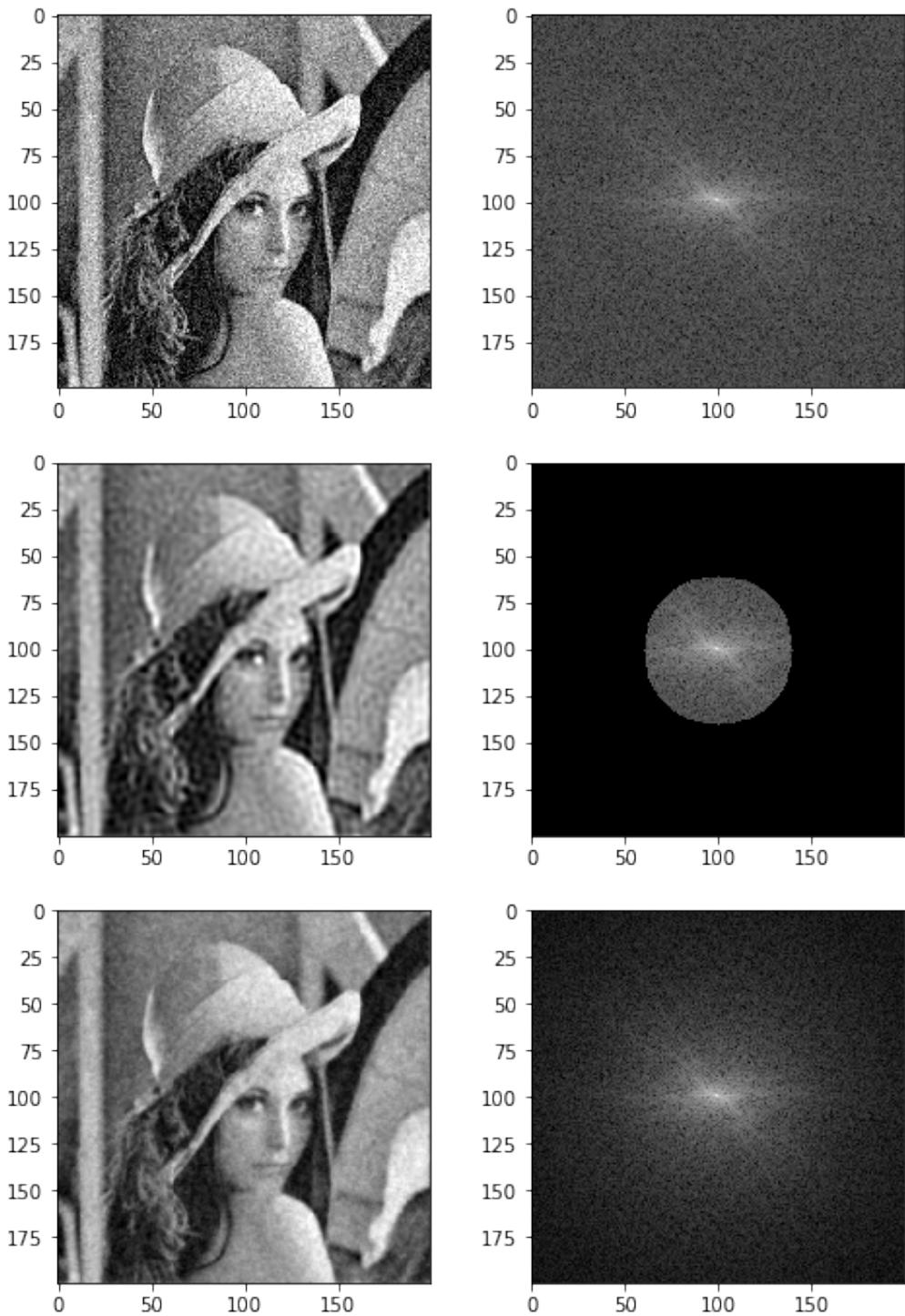
ButterWorthLPF的公式表示为： $L(u, v) = \frac{1}{1 + [\frac{D(u, v)}{D_0}]^2}$

在实验室我分别尝试了在相同截止频率 $D_0 = [5, 10, 20, 40]$ 下两种滤波器的效果，结果如下图。可以看到的是： -

- ButterWorth如课上所讲的没有震铃效应，并且边缘模糊更均匀。
- 在相同的截止频率下， ButterWorth的模糊没有理性滤波器强。



低通滤波器最常见的应用领域便是噪音消除，对于独立分布的信号噪音，通过削弱频域中高频信息，即可降低噪音。下图为以0.1的比例加入正太分布的噪音后，经过相同截止频率D0=40后，不同滤波器的结果。



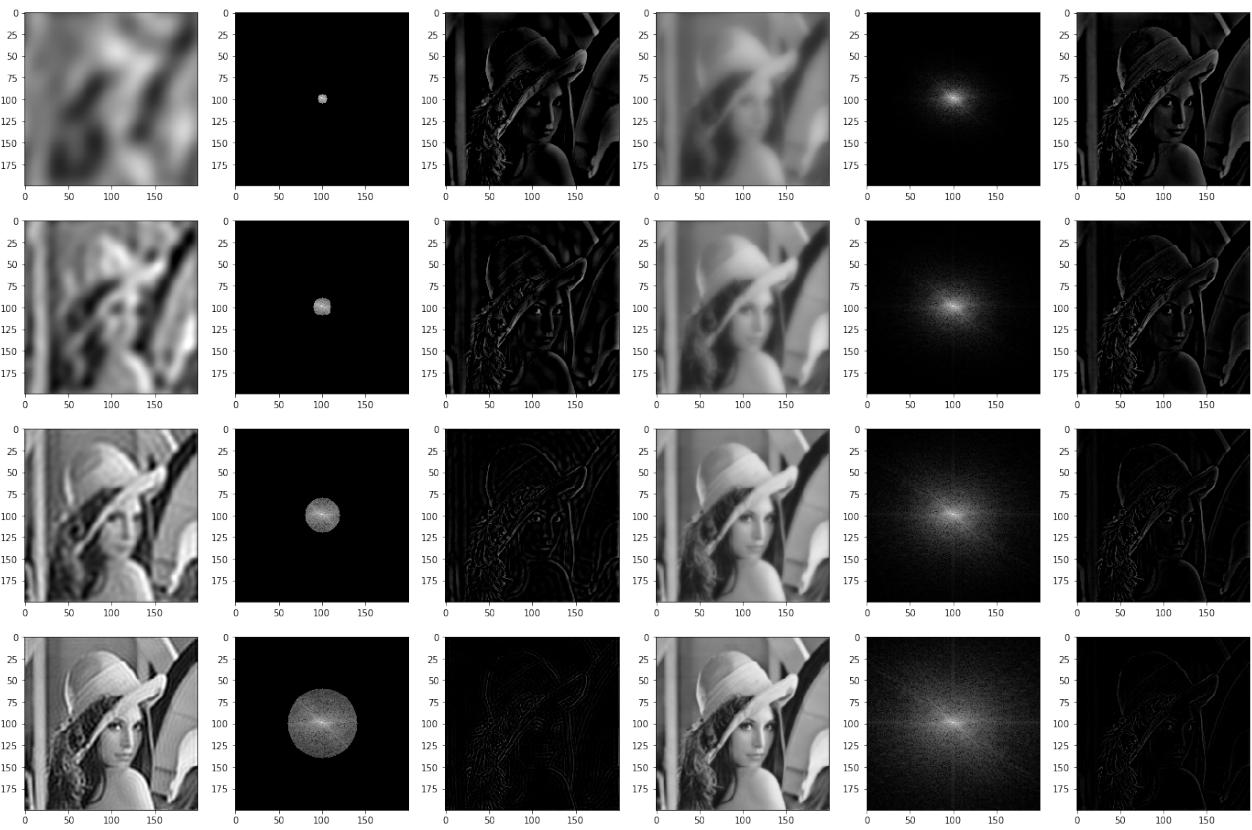
line 1: 带噪音的空域与频域信息

line 2: 理想LDP去噪结果

line 3: ButterWorthLDP去噪结果

2.2 高通滤波器与图像锐化

由于对信号的空域和频域的线性操作是完全相对应的，因此在实验理想高通滤波器和ButterWorth高通滤波器时，使用 $h(x, y) = \text{image}(x, y) - l(x, y)$ 的线性操作公式即可，下图为实验结果。

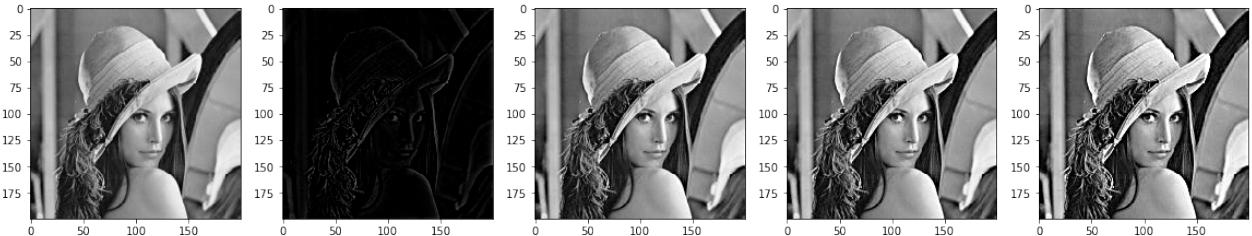


col 1: 理想低通滤波器空域; col 2: 理想低通滤波器频域; col 3: 理想高通滤波器空域

col 4: ButterWorth低通滤波器空域; col 5: ButterWorth低通滤波器频域; col 6: ButterWorth高通滤波器空域

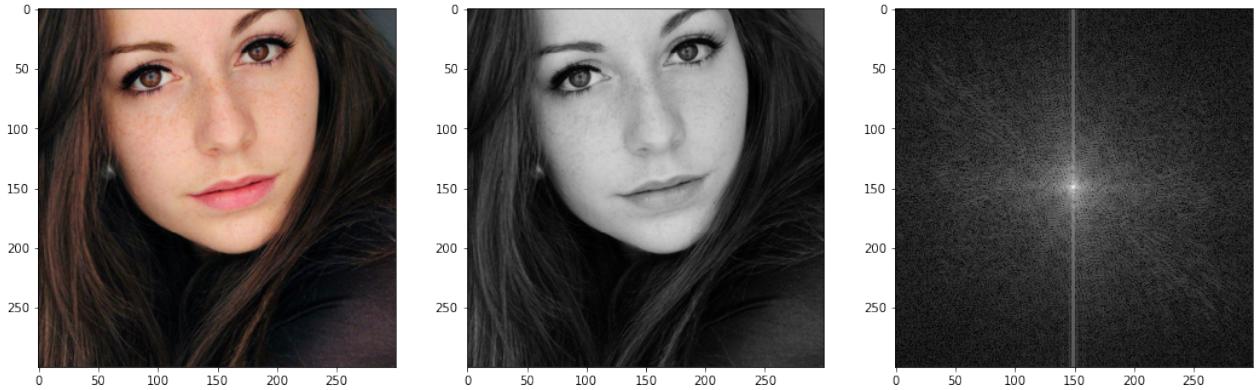
截止频率: [5, 10, 20, 40]

将原信号空域和高通信号按照比例叠加, 即可形成边缘锐化的效果。 (边缘信号加入比例: 0.3, 0.5, 1.0)



2.3 滤波器综合运用的人像美化

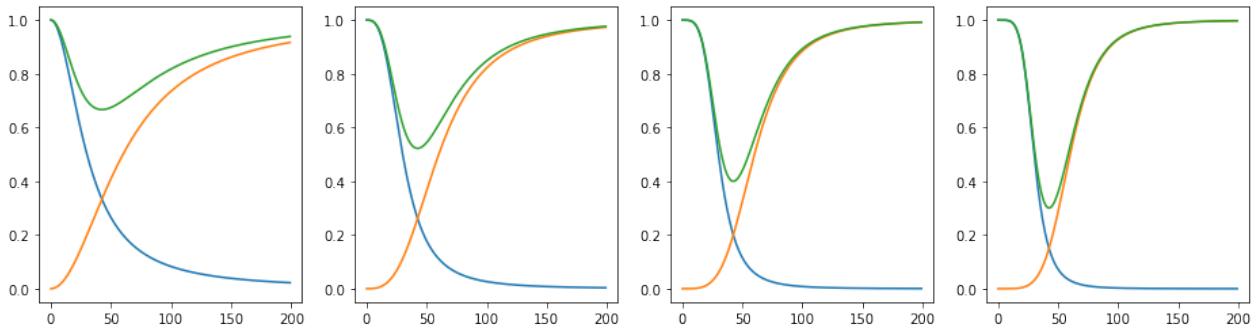
在上面单独的高低通滤波器实验中, 我们看到低通滤波器能够图像模糊, 有助于去除人像面部的雀斑; 而高通滤波器有助于边缘, 增强眼部和面部的层次感。然而, 若单独实验某种滤波器, 则不能达到两个目的。



上图的女性图片里，我们的目的便是将脸上细微的雀斑处理掉，而雀斑与肤色存在差异，属于一种较弱的高频信息。因此，为了能够均保留低频和高频的两头信息，而消弱中间频率的信息，我自主设计了一种双头ButterWorth滤波器。

$$\text{双头滤波器的公式定义: } H(u, v) = \frac{1}{1 + \left[\frac{D(u,v)}{D_l} \right]^e} + \left(1 - \frac{1}{1 + \left[\frac{D(u,v)}{D_h} \right]^e} \right)$$

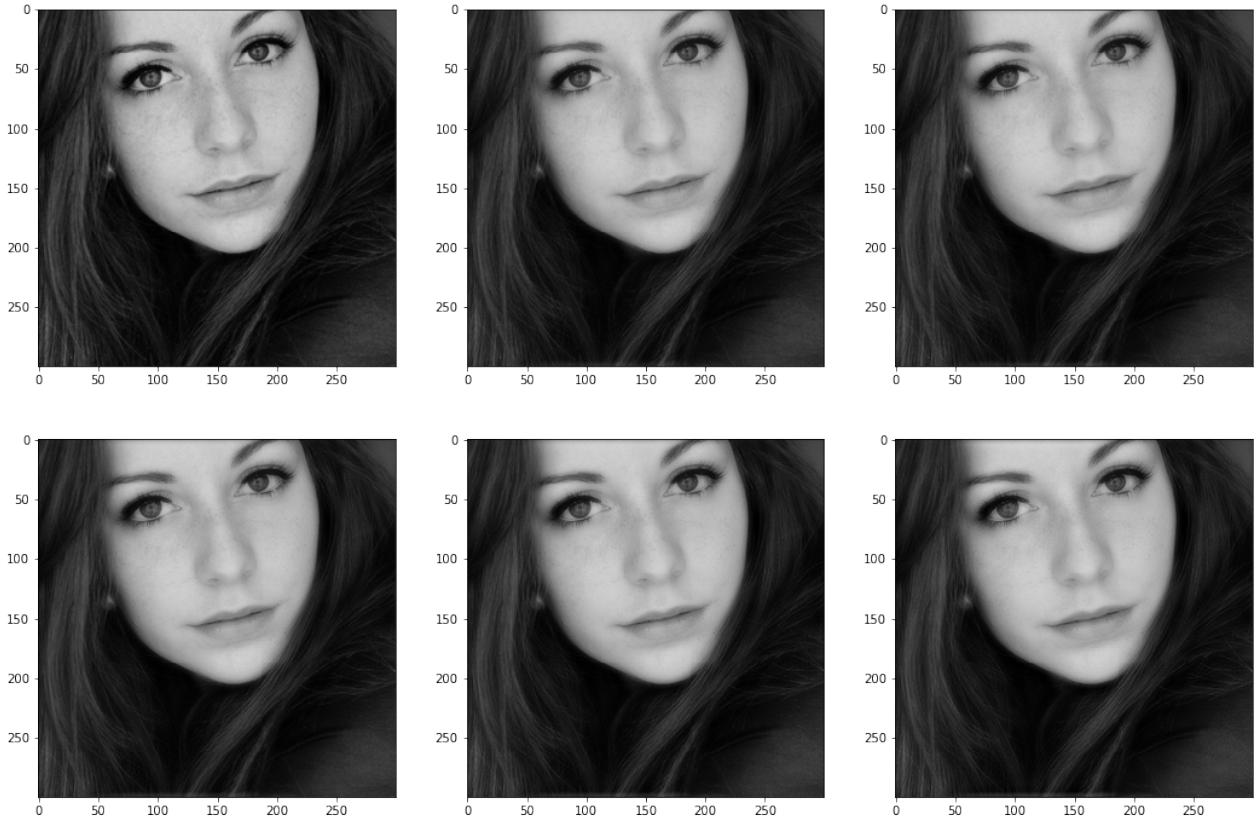
可变参数: D_l, D_h, e



x 轴: 距离频域中心距离 y 轴: 绿线 - $H(u,v)$

经过双头滤波器后的结果如下图所示，当截止频率区间设置为30-60时，随着指数的增大，人像面部的雀斑所代表的中频信息下降了，而其他高频信息则被一定程度地保留。

若想进一步地增强边缘，则可以再对生成图做锐化的处理，上述方法其实也可以等价于在空域中，将使用不同截止频率的滤波器经过线性组合后的结果。



line 1 : 原图, $e=2, e=2.3$

line 2: $e=2.6, e=2.9, e=3.5$

2.4 DFT应用——图像放大与缩小（失败案例）

在实验中，我使用了Lena图来实验，理想情况下，由空域的采样点得到了对于连续正交基的线性组合比例，在由频域变换到空域时，可以在频域上采样连续函数上的其他点，以完成图像的放大与缩小。对于加入放大和缩小功能的DFT逆变换，其公式发生了变换：

$$\text{原DFT逆变换: } f(x, y) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u, v) e^{j2\pi(\frac{ux}{M} + \frac{vy}{N})}$$

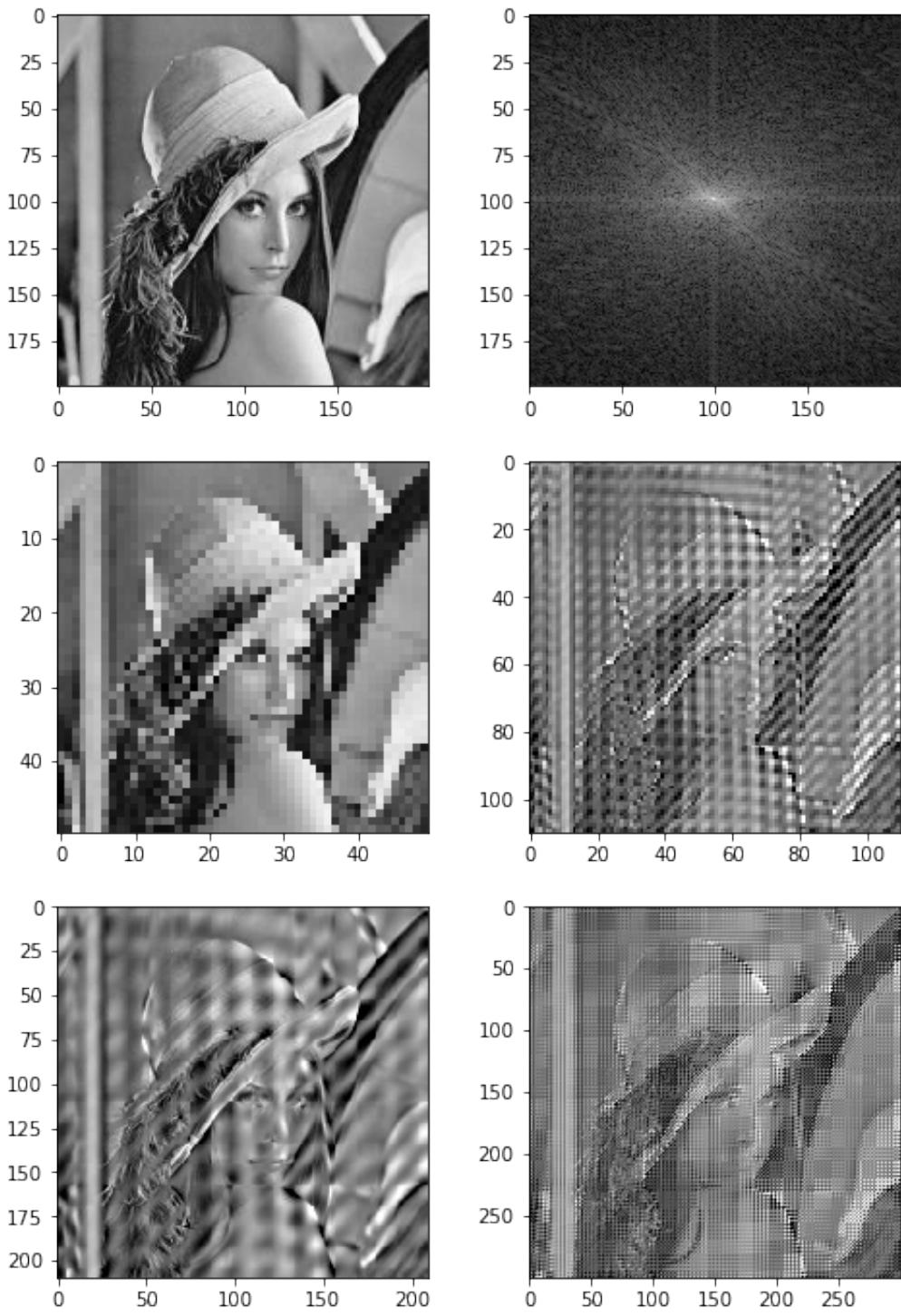
$$\text{支持放大缩小的DFT逆变换: } f(x, y) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u, v) e^{j2\pi(\frac{ux}{M'} + \frac{vy}{N'})}$$

注： M', N' 表示新图像的大小，与原图像的M, N大小不同。

Lena图的放大与缩小

此任务实际上是两个，即放大和缩小，我在实验中针对两个方向分别尝试了两个分辨率：

- 缩小：200x200 -> 50x50(等比例), 200x200 -> 110x110(不等比例)
- 放大：200x200 -> 300x300(等比例), 200x200 -> 210x210 (不等比例)



row 1 : 原图, 幅度谱

row 2: 缩小 $50 \times 50, 110 \times 110$

row 3: 放大 $210 \times 210, 300 \times 300$

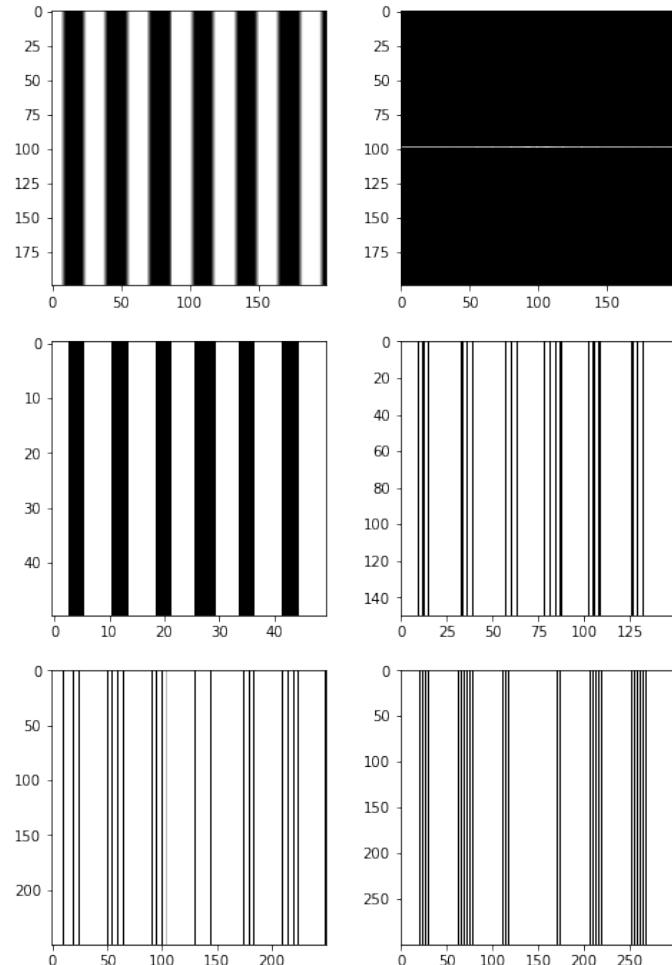
上述的图像放缩结论如下：

- 图像缩小：在部分情况下有较好的效果，当缩小图片时在频域上的采样点是原图采样点的子集效果较好。
- 图像放大：无论取值大小，效果均不佳。
- 问题分析：存在DFT的栅栏效应，由于离散傅立叶变换基于采样，等同于只能从信号的采样点（栅栏的缝隙）观测信号，仅仅凭借这些采样点是无法对复杂的图像信号进行很好效果的拟合。因此，当放大或缩小时，如果要求出和原采样点不对齐的信号（栅栏上的信号），就会效果很差。

简单图的放大与缩小

考虑到上面的Lena图信号比较复杂，仅靠200个采样点无法较好地拟合，因此我手工构建了一个琴键图，由幅度谱可以看到每行之间的内容完全是相同的。

采样和Lena图同样的参数进行实验，但是依旧没有很好的效果，因此也反映了DFT在基函数有限和采样点有限的情况下对信号存在过拟合的情况。



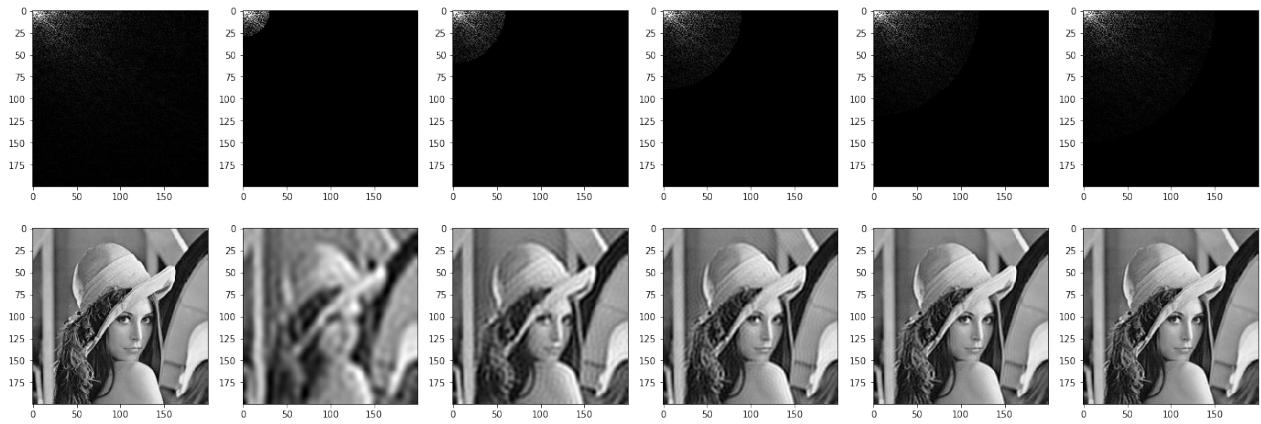
2.5 DCT应用 - 图像压缩

DCT相较于DFT，其能量更加集中，因此适用于将图像压缩，基本的压缩原理是仅保留矩阵中能量高的数据，其他数据置为0，由此实现高还原度的有损压缩。

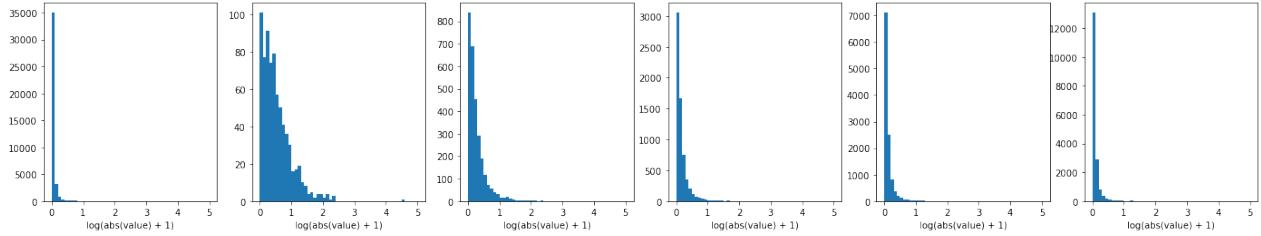
压缩方法其实可以类比在频域上的理想滤波器，对于滤波器的策略我首先设计了一种基础形式：

$$H(u, v) = D(u, v) <= D_0 \ ? \ 1 \ : \ 0$$

此形式的滤波器其实在左上角画了一个半径为 D_0 的四分之一圆，在圆范围内的被选择，不在范围内的则舍弃。



从左至右分别为：原图, $d0=30, d0=60, d0=90, d0=120, d0=150$

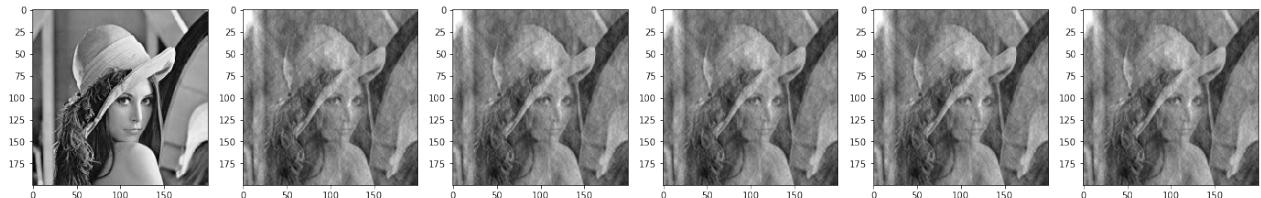


不同量化中，幅度谱值的对数分布情况，从左至右分别为：原图, $d0=30, d0=60, d0=90, d0=120, d0=150$

改进与提升：由分布情况可以看到，对于200的lena图，只有将直径设置为120时左右图像质量才能基本恢复正常，但是较大的半径中也对应了较多的低幅信息，对于空间还是存在很大的消耗。在此提供两种图像压缩的优化思路：

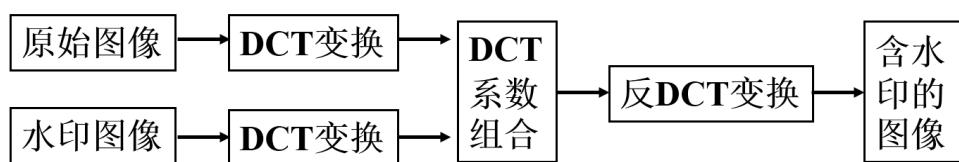
1. **分割压缩：**无论图像大小，按照8*8, 16*16等大小的方格进行划分并进行DCT，变换后选择左上角的有效信息，在还原时也按照划分的区域进行还原。这样做的优势在于能够进一步减小存储低幅信息的内容，存储的坐标基更少。
2. **三元组存储矩阵：**利用比例截断的方式选择整个DCT频谱图中的高幅信号（包含低频和部分高频），使用三元组——（横坐标，纵坐标，值）的方式存储有效信息，其余则直接丢弃。

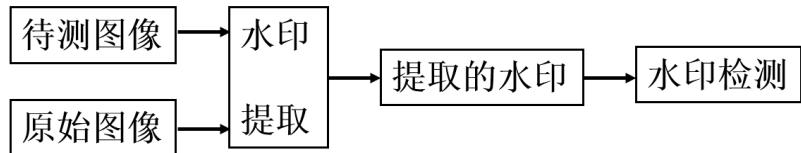
时间有限，仅简单尝试了三元组存储的方式，但是效果不佳。



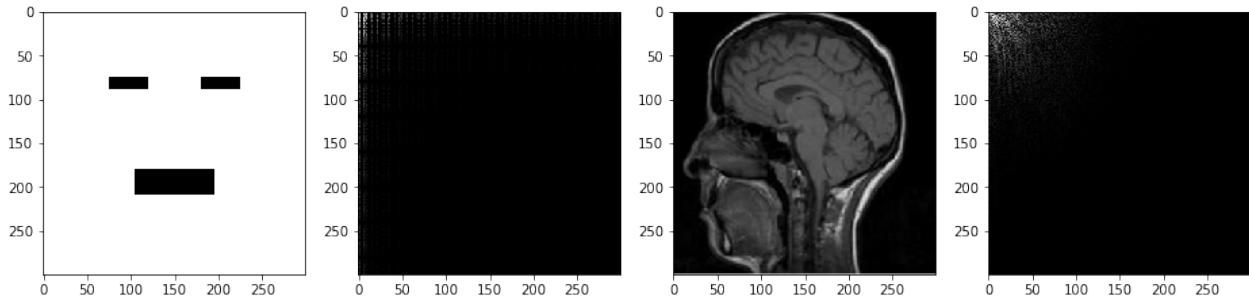
2.6 DCT应用 - 数字水印

最后一个应用实验是关于数字水印的，其也是DCT变换的一个常见应用。数字水印的加入和提取对应着以下的操作：

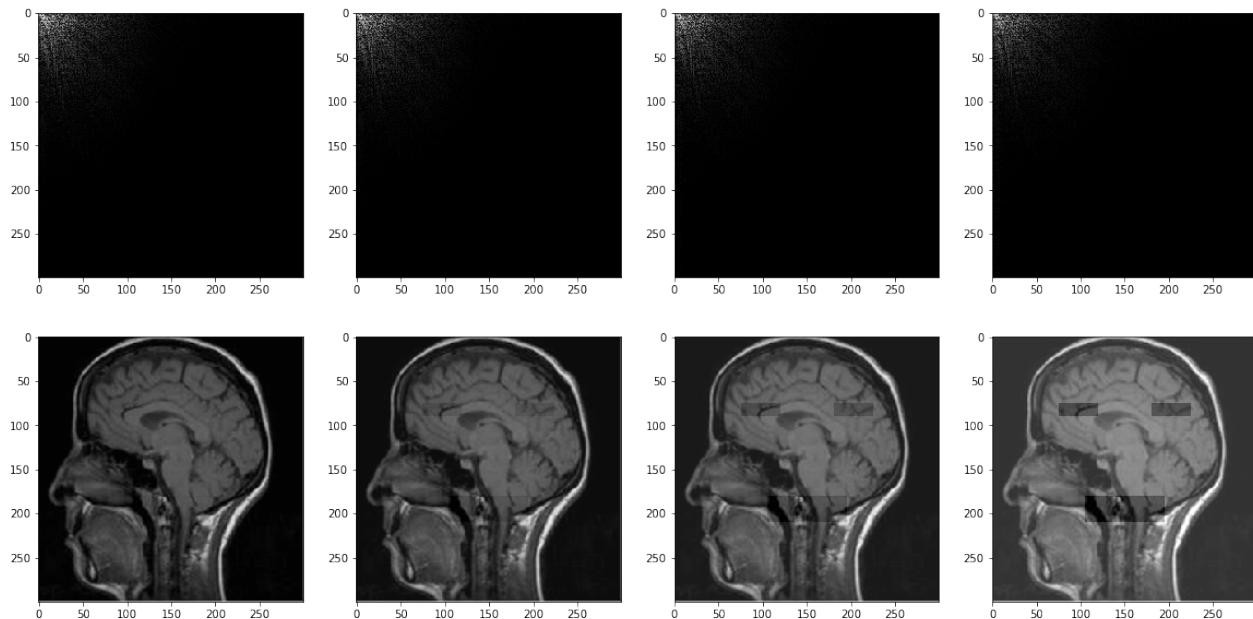




在制作含水印的图像阶段，我以 $\alpha = [0.01, 0.05, 0.1, 0.2]$ 的比例向原图像加入水印，呈现的结果中发现仅有0.01的案例无法被肉眼可见水印。



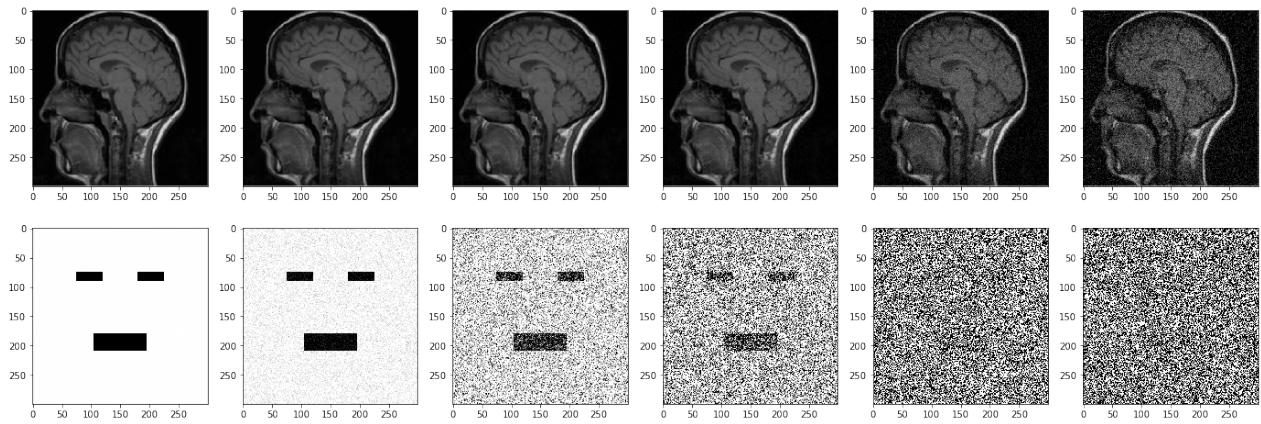
水印图与原图在空域和频域的结果



$\alpha=0.01, 0.05, 0.1, 0.2$ 时的生成图情况

选择上述结果中的 $\alpha=0.01$ 参与水印提取，在进行提取水印时，我尝试人为地引入了一些高斯噪音 $\beta = [0, 0.001, 0.005, 0.01, 0.05, 0.1]$ ，可以看到我的水印能够抵御最高到0.01比例的噪音干扰，但此时被损坏图像的空域也不存在肉眼可见的噪音点，因此目前抵抗人为破坏的能力还不够强。

因此，数字水印除了基础的操作外，从镶嵌和提取还有很多鲁棒性考虑和优化，需要更深的研究。



3 结论

基于傅立叶变换原理的FDT和FCT在图像处理领域有很重要的应用，我认为它其实是下沉到人对信号理解更本质的角度去处理图像：低频信号占比多，高频信号占比少，但是高频信号所代表的边缘则是信息量是很大的。从一致性到差异性的角度去“整理”图像，使得我们新的角度“操控”图像。

当然，就像上面所说，傅立叶变换从一个视角看图像，视角必然是存在局限的，因此功能也存在限制。就像人像美化，如果在空域通过识别器官，而后又针对性地用卷积核操作可以满足更多的需求，而深度学习则是通过自学习卷积核的方式从合适的角度认识图像并作处理。因此，在图像处理时，需要深刻理解到各种图像信号编码角度的优劣，进行合适的选择。