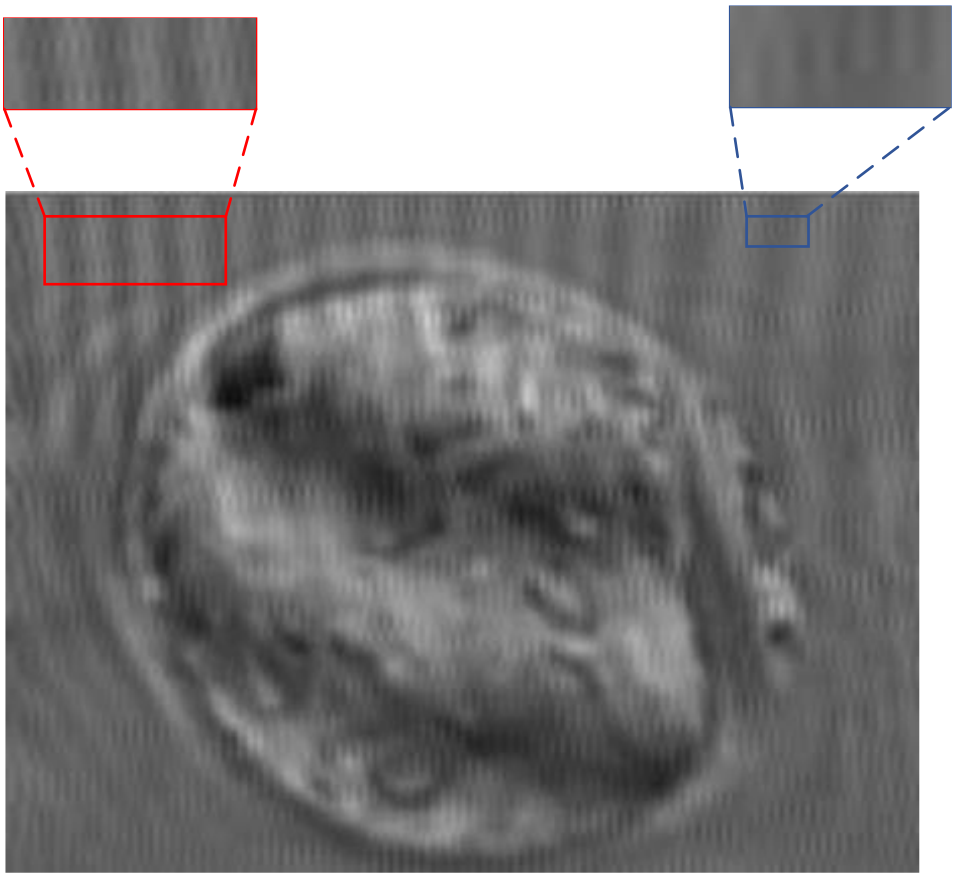
数字图像处理大作业

李岳 1811144

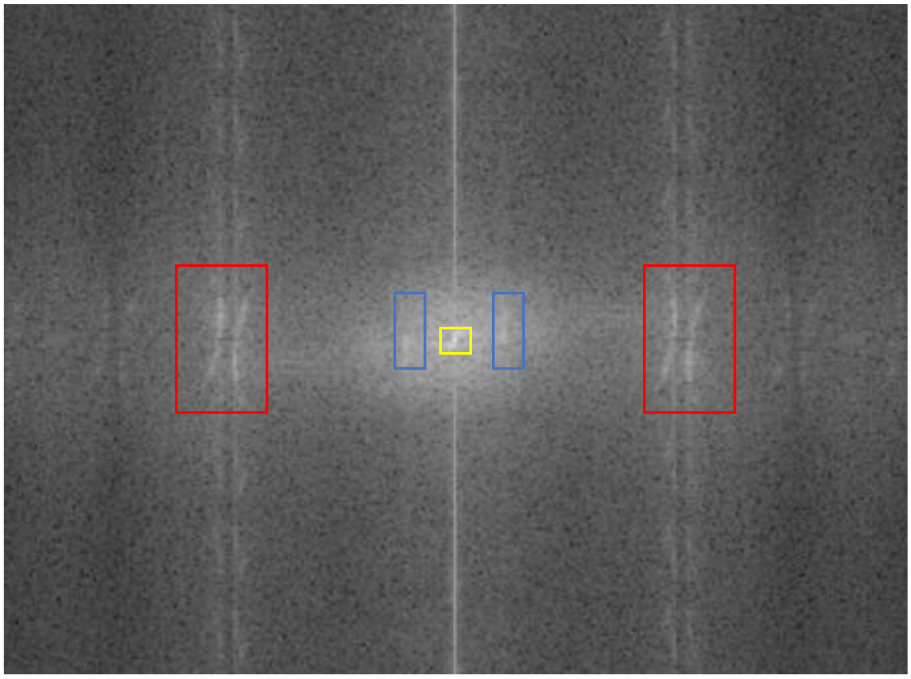
去除图像中的条纹

一、图像理论分析

在原始图像中存在几种类型的周期性条纹，红色框区域中的是大条纹，蓝色框中的是小条纹，以及图像中心的两条黑色条纹。根据这些条纹周期性的特点，很容易想到通过频域的方法来滤除这些条纹。



对图像进行傅里叶变换，在变换后的频域图中，可以看到有几处特别亮的区域，这些区域对应原图像中不同类型的周期性条纹。越靠近中心的亮斑频率越低，在原图像中对应变化缓慢的条纹；越远离中心得亮斑频率越高，在原图像中对应变化迅速的条纹。理论上在频域上去除对应的区域便可以消除条纹。



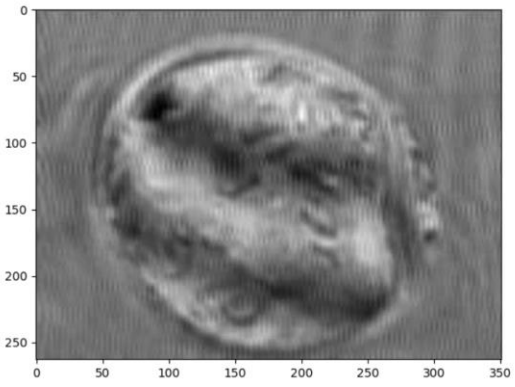
二、算法设计

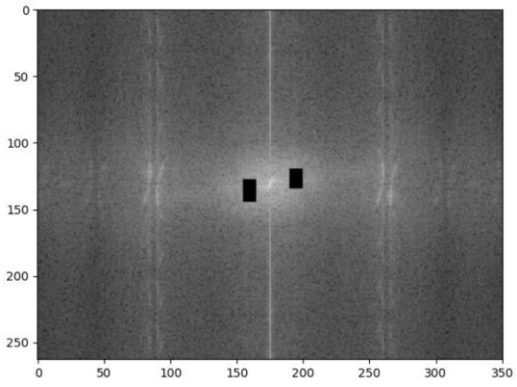
在频域上的处理，这里使用理想的陷波滤波器去除频域上的亮斑，达到消除相应条纹的目的。首先初始化掩膜 mask 为全 1.

1、竖直大条纹去除

竖直大条纹变化缓慢，对应的是频域中低频的部分，即靠近中心的位置。令 mask 中对应的位置置零。

```
mask[120:135, 190:200] = 0
mask[128:145, 155:165] = 0
```



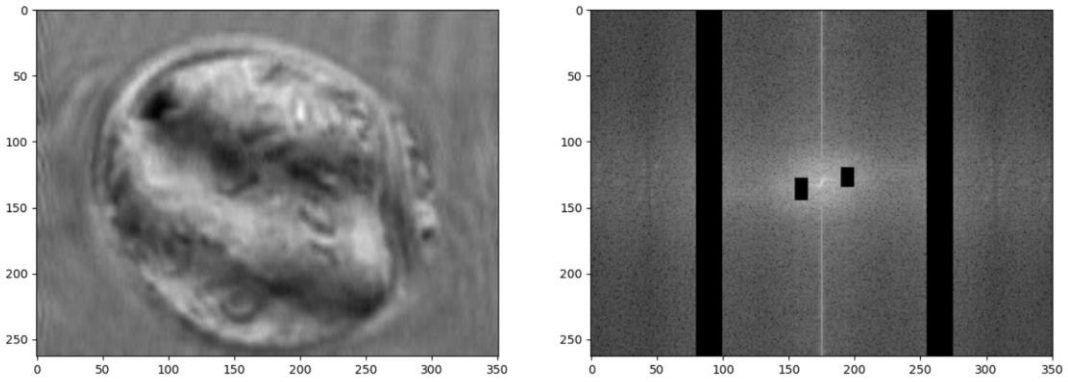


2、竖直小条纹去除

竖直小条纹变化快，对应的是频域中高频的部分，即如下图所示频谱图中远

离中心的两个亮斑区域。将对应部分置零，即可消除小条纹。

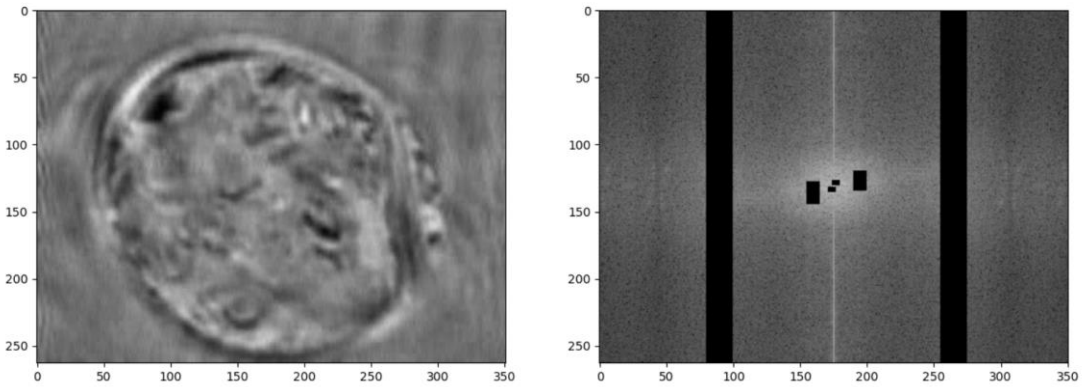
```
mask[:, 80:100] = 0
mask[:, 255:275] = 0
```



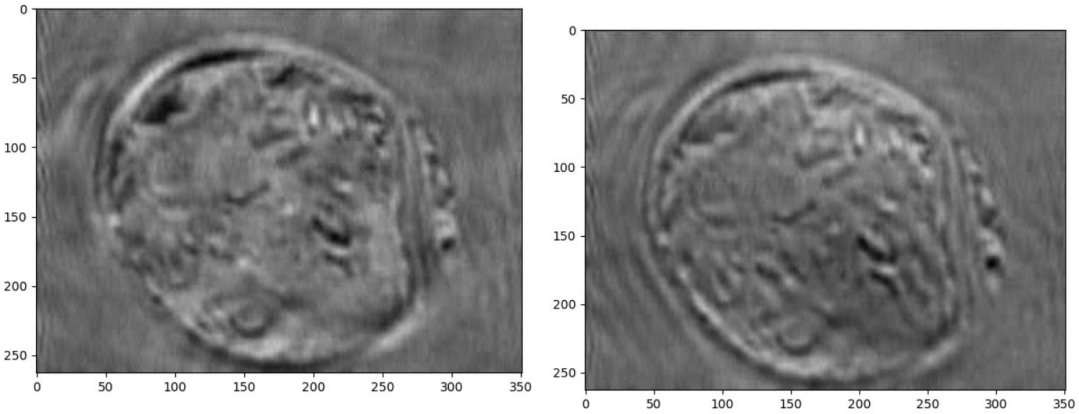
3、中心横向条纹去除

中心横向条纹对应频谱中靠近中心的区域

```
mask[127:131, 174:180] = 0
mask[132:136, 171:177] = 0
```



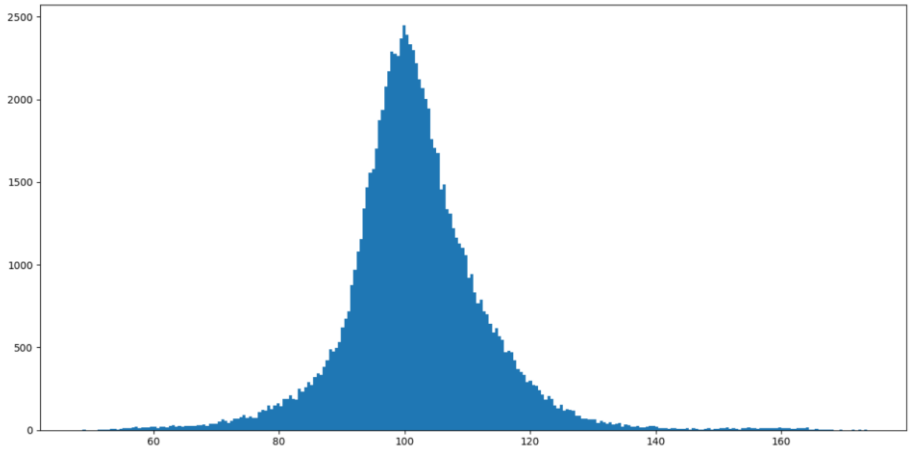
再适当调整滤波器范围，得到最终的图像



此时图像中的大小条纹基本去除，不过现在图像显得比较暗，对比度不强，因此接下来采用空域的方法对图像进行进一步处理。

4、直方图处理

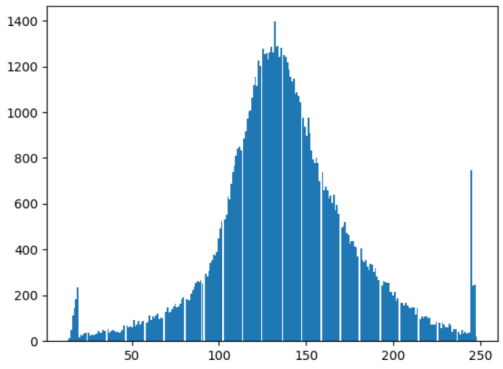
如图所示的是经过上述频域滤波后的直方图，从直方图中可以看出大多数像素点的灰度值处于 80 到 140 之间，如果将这些灰度值扩展到 0-255 的范围上，图像的视觉效果将会有很好的提升。



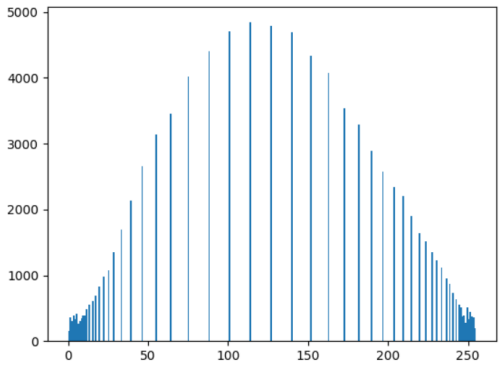
下面分别采用几种灰度变换，观察不同变换下的效果图

（1）自定义灰度变换函数

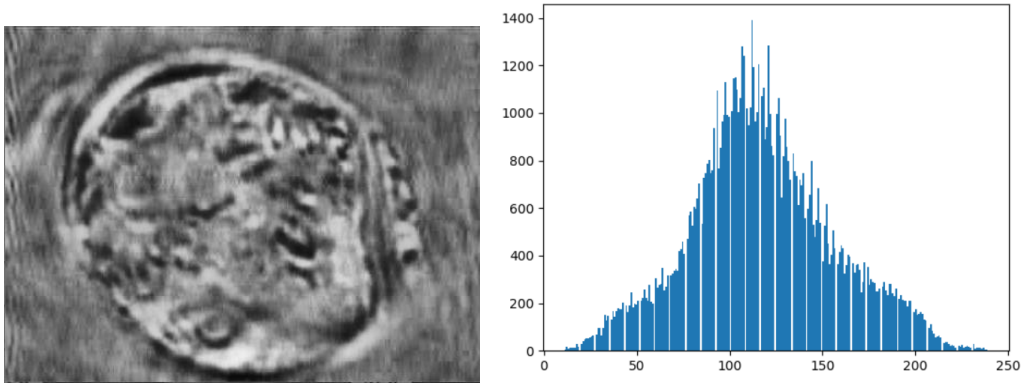
这里采用了分段线性变换对图形进行处理，分成三段，两个关键转折点为(70,20)以及(130,245)



（2）整体直方图均衡



（3）局部直方图均衡



从上面三种灰度变换的效果来看，个人认为自定义的灰度变换效果稍微好些，这可能和图像本身的特点有关。

5、图像增强

三、算法优缺点

优点：

1、在频域上采用理想矩形陷波滤波器，简单方便，根据关键区域能够较好的去除图像中的周期性条纹；

2、对频域处理后的图像进行灰度变换以及

缺点：

1、采用理想陷波滤波器虽然简单方便，但是容易造成空间域的振荡；

2、在频域选取关键区域时采用的是手动的方式，区域的位置和大小的判定均人为设置，得出的结果具有很大的偶然性；

3、算法适用性不强。

四、效果评价

对于效果的评价有主观评价以及客观评价。在客观评价又有全参考、部分参考、以及无参考三种评价方式。由于这里没有理想的图像作为参考，因此采用无参考方式对图像进行评价。常用的有均值、标准差、平均梯度以及 BRISQUE 等算法。这里选用平均梯度作为标准对图像进行质量评价，平均梯度能反映图像中细节反差和纹理变换，它在一定程度上反映了图像的清晰程度。其计算公式为：

$$\nabla G = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}\sqrt{\Delta xF(i,j)^2 + \Delta yF(i,j)^2}$$

经计算原图像的平均梯度为 4.12，经过处理后图像的平均梯度为 17.29。

五、算法改进

如之前所述的缺点，理想矩形滤波性会造成空间域的振荡，因此滤波器可以改为高斯滤波器、巴特沃斯滤波器等非理想滤波器，这样的空间效果会更加好；

此外，需要找出合适的自动算法，能够根据需求自动找出频域中导致产生条纹的区域，这用可以尽可能避免由于手动设置而产生效果的随机性和偶然性，同时也能将算法更好的用于其他图像的处理当中。

在空间域的处理中，经过直方图均衡以及图像增强后的效果依旧不太好，存在部分阴影，空间域的算法有待改进。同时可以增加缺陷识别与提取的算法，将图像中表示缺陷的地方提取出来，用作后续进一步操作。

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
import math

img = cv2.imread('img.jpg',cv2.IMREAD_GRAYSCALE)
rows, cols = img.shape
plt.figure()
plt.subplot(121),plt.imshow(img,cmap='gray')

##傅里叶变换
dft = cv2.dft(np.float32(img), flags=cv2.DFT_COMPLEX_OUTPUT)  # 将空间域转换为频率域
dft_shift = np.fft.fftshift(dft)  # 将低频部分移动到图像中心
f = cv2.magnitude(dft_shift[:, :, 0], dft_shift[:, :, 1])
plt.subplot(122),plt.imshow(20*np.log(f+1),cmap='gray')


##mask
mask = np.zeros((rows, cols, 2), np.float32)
for i in range(rows):
    for j in range(cols):
        mask[i][j] = 1


##竖直大条纹
mask[120:135, 190:200] = 0
mask[128:145, 155:165] = 0
mask[120:135, 184:188] = 0
mask[128:145, 167:171] = 0
##竖直小条纹
mask[:, 80:100] = 0 # mask[110:145, 80:100] = 0
mask[:, 255:275] = 0 # mask[110:145, 255:275] = 0
##大黑纹
mask[127:131, 174:180] = 0 # mask[127:131, 174:180] = 0
mask[132:136, 171:177] = 0 # mask[132:136, 171:177] = 0

##与 mask 相乘
fshift = dft_shift * mask
f2 = cv2.magnitude(fshift[:, :, 0], fshift[:, :, 1])
f_ishift = np.fft.ifftshift(fshift)
img_back = cv2.idft(f_ishift)/(rows*cols)
img_back2 = cv2.magnitude(img_back[:, :, 0], img_back[:, :, 1])

##直方图
```

```
plt.figure('origin hist')
plt.hist(((img_back2)).ravel(), 256)

##局部直方图均衡
img_back3 = np.uint8(img_back2)
clahe = cv2.createCLAHE(clipLimit=4, tileGridSize=(5,5))
img_back3 = clahe.apply(img_back3)
plt.figure('local_hist'),plt.hist((np.uint8(img_back3)).ravel(), 256)
plt.figure(),plt.imshow(img_back3,cmap='gray')


##整体直方图均衡
ehist = cv2.equalizeHist(np.uint8(img_back2))
plt.figure('_hist'),plt.hist(ehist.ravel(), 256)
cv2.imshow('ehist',(ehist))


##图像增强
img_back4 = np.uint8(img_back2)
kernel_1 = np.float32([[0,-1,0],
                       [-1,5,-1],
                       [0,-1,0]])
img_back4 = cv2.filter2D(img_back3,-1,kernel_1)
cv2.imshow('back4',(img_back4))

##对比度
def do_math(img,fn):
    rows, cols = img.shape
    nimg = np.zeros((rows, cols), np.float32)
    for i in range(rows):
        for j in range(cols):
            nimg[i][j] = fn(img[i][j])
    return nimg

def stretch(x):
    x1=70
    x2=130
    y1=20
    y2=245
    result = 0
    if x<=x1:
        result = x*y1/x1
    elif x1<x<=x2:
        result = (x-x1)*(y2-y1)/(x2-x1)+y1
    else:
```

```
        result = (x-x2)*(255-y2)/(255-x2)+y2
    if result<=0:return 0
    if result>=255:return 255
    return result

img_back5 = img_back2.copy()
img_back5 = do_math(img_back5,stretch)
plt.figure(),plt.hist((np.uint8(img_back5)).ravel(), 256)
cv2.imshow('stretch',np.uint8(img_back5))

mask2 = np.zeros((rows, cols), np.float32)
r = 120
for i in range(rows):
    for j in range(cols):
        if((i-rows/2)**2+(j-cols/2)**2)<r*r:
            mask2[i][j] = 1
img_cut = mask2*img_back2
plt.figure('cut'),plt.imshow(img_cut,cmap='gray')

# ##中值滤波
img_median = cv2.medianBlur(np.uint8(img_back2), 5)
#
plt.figure('median'),plt.imshow(img_median,cmap='gray'),plt.title('medi
an')
# clahe = cv2.createCLAHE(clipLimit=5, tileGridSize=(25,25))
# img_local = clahe.apply(img_median)
#
plt.figure('local'),plt.imshow(img_local,cmap='gray'),plt.title('local'
)

##二值
ret,thresh1 =
cv2.threshold(np.uint8(img_back2),90,255,cv2.THRESH_BINARY)
plt.figure('bin'),plt.imshow(thresh1,cmap='gray')
th3 =
cv2.adaptiveThreshold(img_median,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,\
            cv2.THRESH_BINARY,11,2)
th3 = cv2.medianBlur(th3, 5)
plt.figure('adptive'),plt.imshow(th3,cmap='gray')

def average_g(img):
    tmp = 0
    for i in range(rows-1):
        for j in range(cols-1):
```

```
            dx = np.float32(img[i, j + 1]) - np.float32(img[i, j])
            dy = np.float32(img[i + 1, j]) - np.float32(img[i, j])
            ds = math.sqrt((dx*dx + dy*dy) / 2);
            tmp+=ds
    g = tmp / (rows*cols)
    return g

print('origin g:',average_g(img))
print('output g:',average_g(ehist))

plt.figure()
plt.subplot(121),plt.imshow(img_back2,cmap='gray')
plt.subplot(122),plt.imshow(20*np.log(f2+1),cmap='gray')

plt.show()
```