Problem 2

Digital Image Processing

肖逸群 116033910112

Combining spatial enhancement methods

问题描述

> Implement the image enhancement task of Section 3.7 (Fig 3.43) (Section 3.8, Fig. 3.46 in our slides). The image to be enhanced is skeleton_orig.tif. You should implement all steps in Figure 3.43. (You are encouraged to implement all functions by yourself, attempting not to directly use Matlab functions such as imfilter or fspecial.)

算法思想

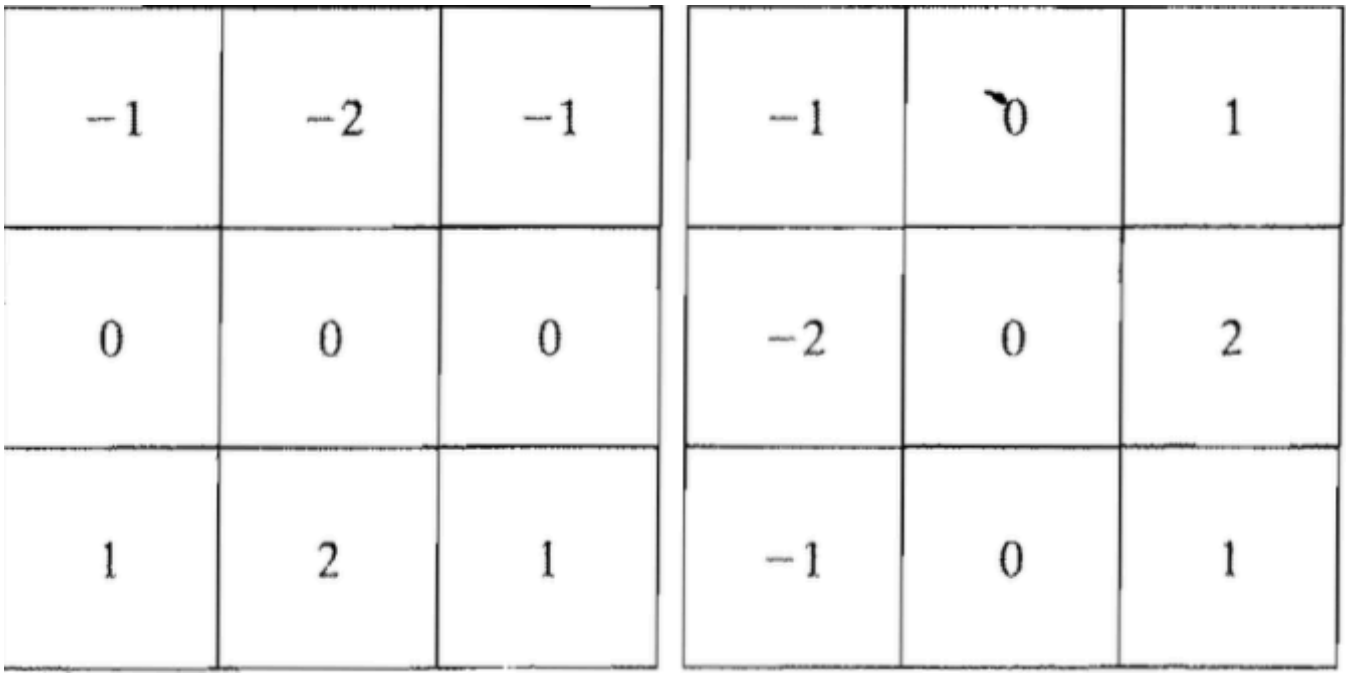
1. **拉普拉斯变换**
   本实验中采用的是 3 * 3 的拉普拉斯滤波器模板，以差分的形式近似得到拉普拉斯算子的值，滤波器模板如下图所示：

−1	−1	−1
−1	8	−1
−1	−1	−1

   对每一个像素点做一次拉普拉斯变换，遍可得到拉普拉斯图像，为了使结果更加具有对比度，我们还对该图像进行了标定。

2. **锐化变换**
   锐化变化只是简单地将原图像与拉普拉斯图像相加，得到锐化图像。

3. **Sobel变换**
   同拉普拉斯变换一样，也是用 3 * 3 的滤波器模板，模板如下：



左边为求 X 的偏导的 Sobel 滤波器模板，右边为求 Y 的偏导的 Sobel 滤波器模板，由此可得出每个像素点的偏导。
根据书本说明，我们取 X, Y 偏导的绝对值之和作为其梯度的近似值，得到 Sobel 变换后的图像。

4. **均值变换**
   简单地取像素周围的几个像素求出其均值，并以此均值替代掉原像素的灰度值，这么做可以降低噪声，对图像进行平滑处理。

5. **相乘变换**
   对两幅图像先做规整化，使其灰度范围在 [0 - 1]之间，再对各个像素点的值进行相乘，这么做可以突出部分区域。

6. **相加变换**
   对两幅图像各个像素进行相加。

7. **幂律变换**
   对各个像素点进行幂律函数的转换，得到幂律图像，转换函数如下：

$$s = c \times r^{\gamma}$$

其中 s 为幂律图像的灰度值，r 为原图像的灰度值， c 和 gamma 由书本说明可得取 1 和 0.5。

源码分析

- 预处理图像，对图像外围加上一层边，其灰度值等于其原本边界像素点的灰度值。

```
def imagePreprocess(input):
    row, col = input.shape
    output = np.zeros((row + 2, col + 2))
    for i in range(row):
        output[i, 0] = input[i, 0]
        output[i, col] = input[i, col - 1]
        for j in range(col):
            output[i + 1, j + 1] = input[i, j]
    for j in range(col):
        output[0, j] = input[0, j]
        output[row, j] = input[row - 1, j]
```

```
    return output
```

- 读取原图像，对图像进行各种处理，最后展示图片。

```
def main():
    originalImage = np.array(Image.open('./resource/skeleton_orig.tif'))
    plt.subplot(2, 4, 1)
    plt.imshow(originalImage, cmap = plt.get_cmap('gray'))
    plt.title('Original')

    preprocessImage = imagePreprocess(originalImage)
    row, col = originalImage.shape
    laplaceImage = np.zeros((row, col))
    Laplacian.laplace(preprocessImage, laplaceImage)
    plt.subplot(2, 4, 2)
    plt.imshow(laplaceImage, cmap=plt.get_cmap('gray'))
    plt.title('Laplacian')

    sharpenedImage = Laplacian.sharp(laplaceImage, originalImage)
    plt.subplot(2, 4, 3)
    plt.imshow(sharpenedImage, cmap=plt.get_cmap('gray'))
    plt.title('Sharpen')

    sobelImage = Sobel.SobelOperators(preprocessImage, mode = 'abs') #mode can choose 'abs' or 'square-root', default is 'abs'
    plt.subplot(2, 4, 4)
    plt.imshow(sobelImage, cmap=plt.get_cmap('gray'))
    plt.title('Sobel')

    tempImage = imagePreprocess(preprocessImage)
    preprocessAveragingFilterImage = imagePreprocess(tempImage)
    averagingFilterImage = AveragingFilter.averagingFilter(preprocessAveragingFilterImage, 5, 5)
    plt.subplot(2, 4, 5)
    plt.imshow(averagingFilterImage, cmap=plt.get_cmap('gray'))
    plt.title('Sobel With 5 * 5 Averaging Filter')

    maskProductImage = MatixOperate.product(sharpenedImage, averagingFilterImage)
    plt.subplot(2, 4, 6)
    plt.imshow(maskProductImage, cmap=plt.get_cmap('gray'))
    plt.title('Mask Product')

    maskSumImage = MatixOperate.sum(originalImage, maskProductImage)
    plt.subplot(2, 4, 7)
    plt.imshow(maskSumImage, cmap=plt.get_cmap('gray'))
    plt.title('Mask Sum')

    powerLawImage = MatixOperate.powerLawTransformation(maskSumImage, c = 1, gamma = 0.5)
    plt.subplot(2, 4, 8)
    plt.imshow(powerLawImage, cmap=plt.get_cmap('gray'))
    plt.title('Power Law')

    plt.show()
```

- 对图像进行拉普拉斯变换

```
def calculate(image, x, y):
    return 8 * image[x, y] - (image[x + 1, y] + image[x - 1, y] + image[x, y + 1] + image[x, y - 1] + image[x - 1, y - 1] + \
           image[x - 1, y + 1] + image[x + 1, y - 1] + image[x + 1, y + 1])
```

```
4
5  def laplace(input, output):
6      row, col = output.shape
7      for i in range(row):
8          for j in range(col):
9              output[i, j] = calculate(input, i + 1, j + 1)
10             if output[i, j] < 0:
11                 output[i, j] = 0
```

- 对图像进行锐化变换

```
1  def sharp(laplacianImage, originalImage):
2      row, col = laplacianImage.shape
3      result = np.zeros((row, col))
4      for i in range(row):
5          for j in range(col):
6              result[i, j] = originalImage[i, j] + laplacianImage[i, j]
7              if result[i, j] > 255:
8                  result[i, j] = 255
9      return result
```

- 对图像进行 Sobel 变换

```
1  def SobelOperators(preprocessImage, mode = 'abs'):
2      row, col = preprocessImage.shape
3      result = np.zeros((row - 2, col - 2))
4      for i in range(1, row - 1):
5          for j in range(1, col - 1):
6              gx = preprocessImage[i + 1, j - 1] + 2 * preprocessImage[i + 1, j] + preproc
   essImage[i + 1, j + 1] - \
7                  preprocessImage[i - 1, j - 1] - 2 * preprocessImage[i - 1, j] - preproce
   ssImage[i - 1, j + 1]
8              gy = preprocessImage[i - 1, j + 1] + 2 * preprocessImage[i, j + 1] + preproc
   essImage[i + 1, j + 1] - \
9                  preprocessImage[i - 1, j - 1] - 2 * preprocessImage[i, j - 1] - preproce
   ssImage[i + 1, j - 1]
10             if(mode == 'square-root'):
11                 result[i - 1, j - 1] = np.sqrt(np.power(gx, 2) + np.power(gy, 2))
12             elif(mode == 'abs'):
13                 result[i - 1, j - 1] = np.abs(gx) + np.abs(gy)
14     return  result
```

- 对图像进行 5 * 5 掩模的均值变换

```
1  def getMean(input, a, b, x, y):
2      dx = x / 2
3      dy = y / 2
4      list = []
5      for i in range(a - dx, a + dx + 1):
6          for j in range(b - dy, b + dy + 1):
7              list.append(input[i, j])
8      return int(np.mean(list))
9
10 def averagingFilter(input, x, y):
11     row, col = input.shape
12     result = np.zeros((row - 4, col - 4))
13     for i in range(2, row - 2):
14         for j in range(2, col - 2):
15             result[i - 2, j - 2] = getMean(input, i, j, x, y)
16     return result
```

- 对图像进行相乘变换

```
def product(image1, image2):
    row, col = image1.shape
    result = np.zeros((row, col))
    for i in range(row):
        for j in range(col):
            result[i, j] = image1[i, j] * image2[i, j] / 255
    return result
```

- 对图像进行相加变换

```
def sum(image1, image2):
    row, col = image1.shape
    result = np.zeros((row, col))
    for i in range(row):
        for j in range(col):
            result[i, j] = image1[i, j] + image2[i, j]
    return result
```

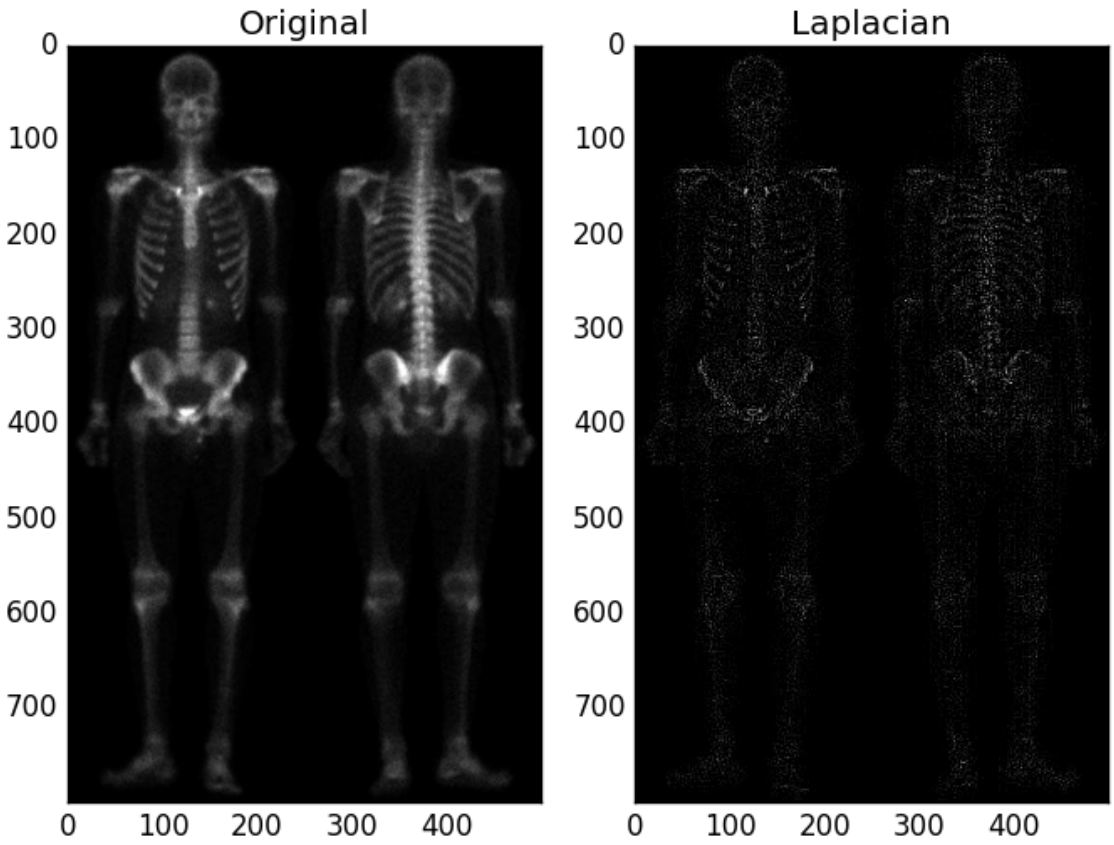
- 对图像进行幂律变换

```
def powerLawTransformation(image, c = 1, gamma = 0.5):
    row, col = image.shape
    result = np.zeros((row, col))
    for i in range(row):
        for j in range(col):
            result[i, j] = c * np.power(image[i, j], 0.5)
    return result
```

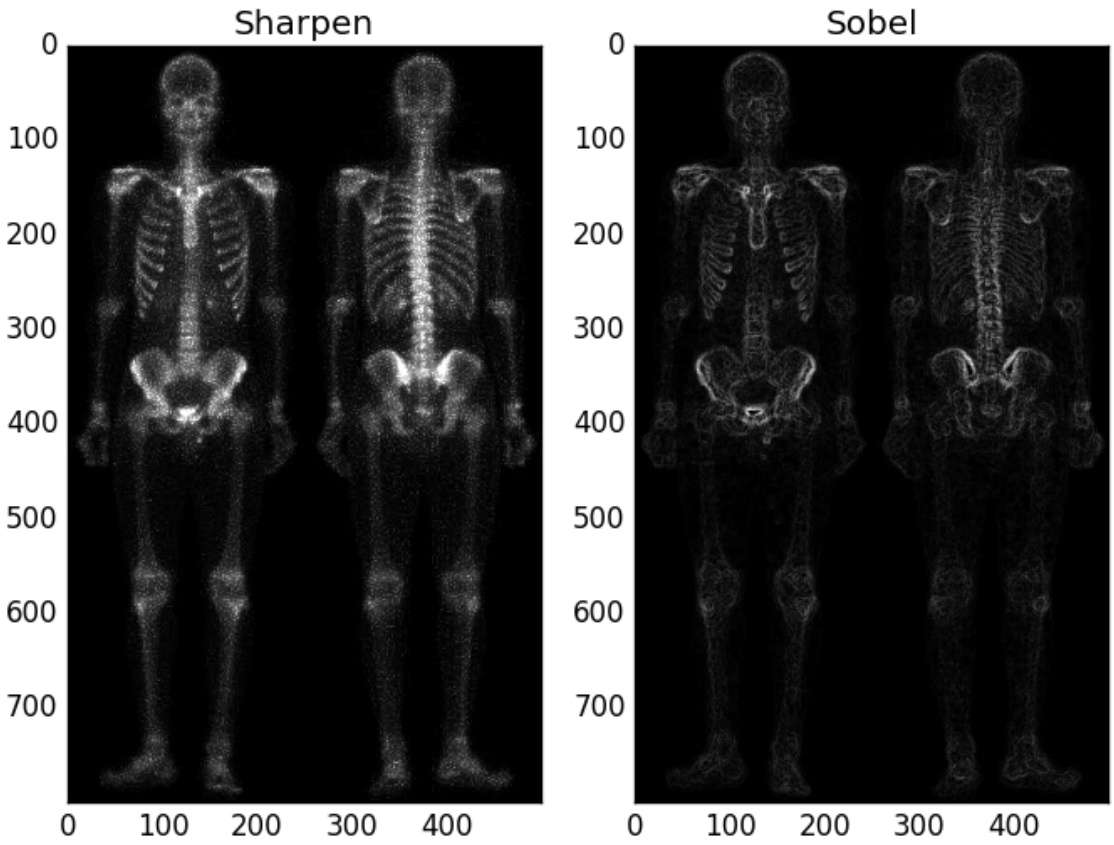
实验结果

实验的输出结果如下列图片所示：

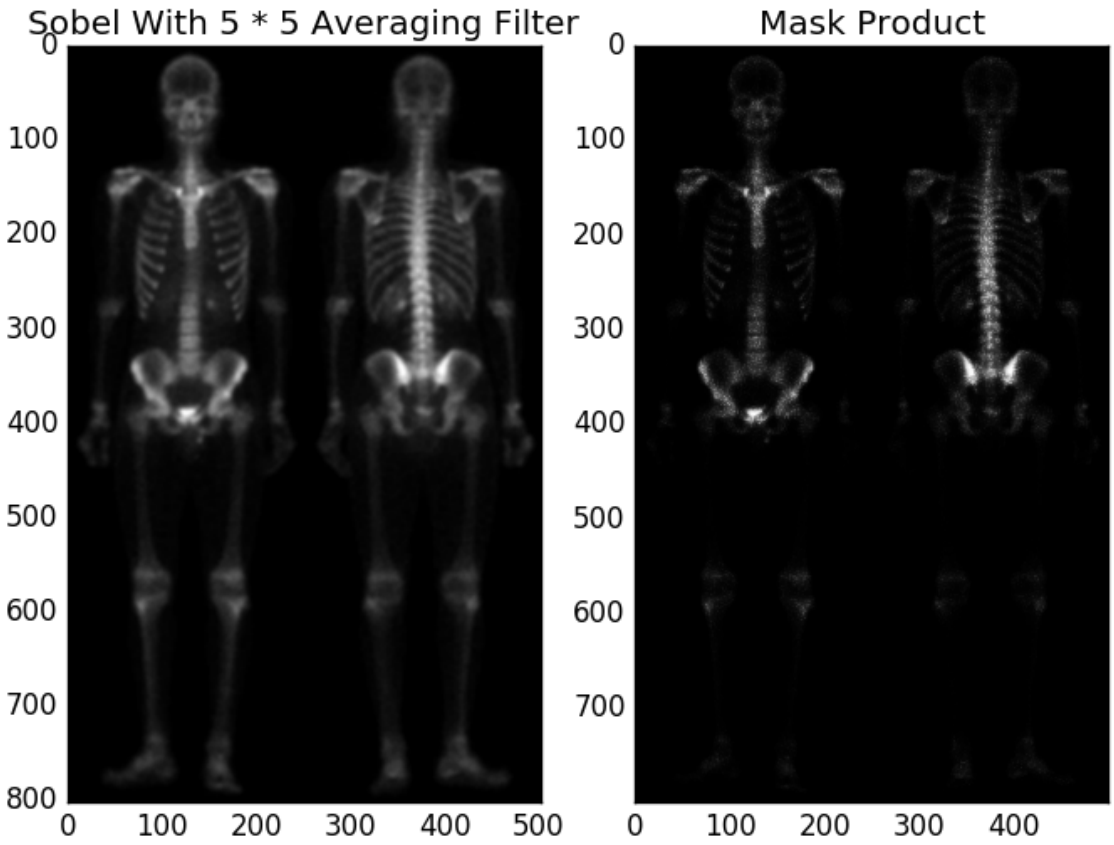
- 左右分别为原始图像与拉普拉斯变换后的图像，由此可见拉普拉斯变换后图像只剩下较为突出的像素点



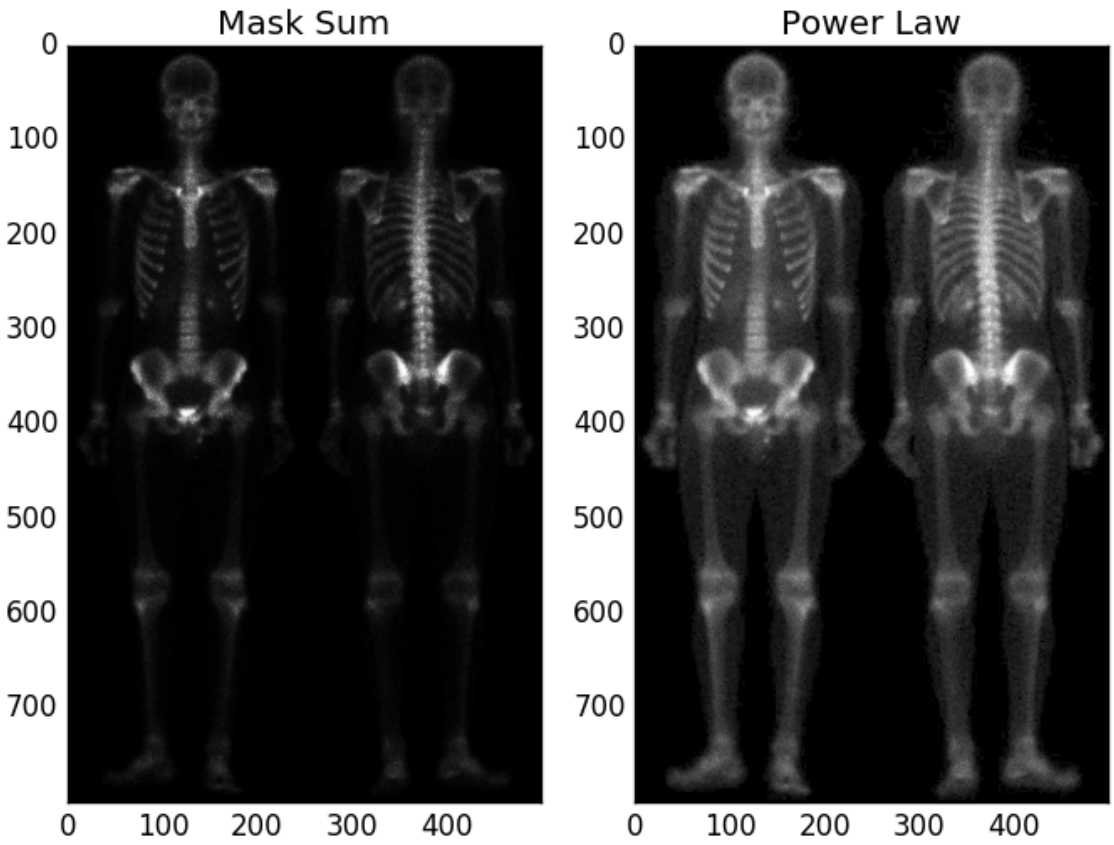
- 左图为原始图像与拉普拉斯变换后的图像相加得到的锐化图像，右图为对原始图像进行 Sobel 变换得到的图像



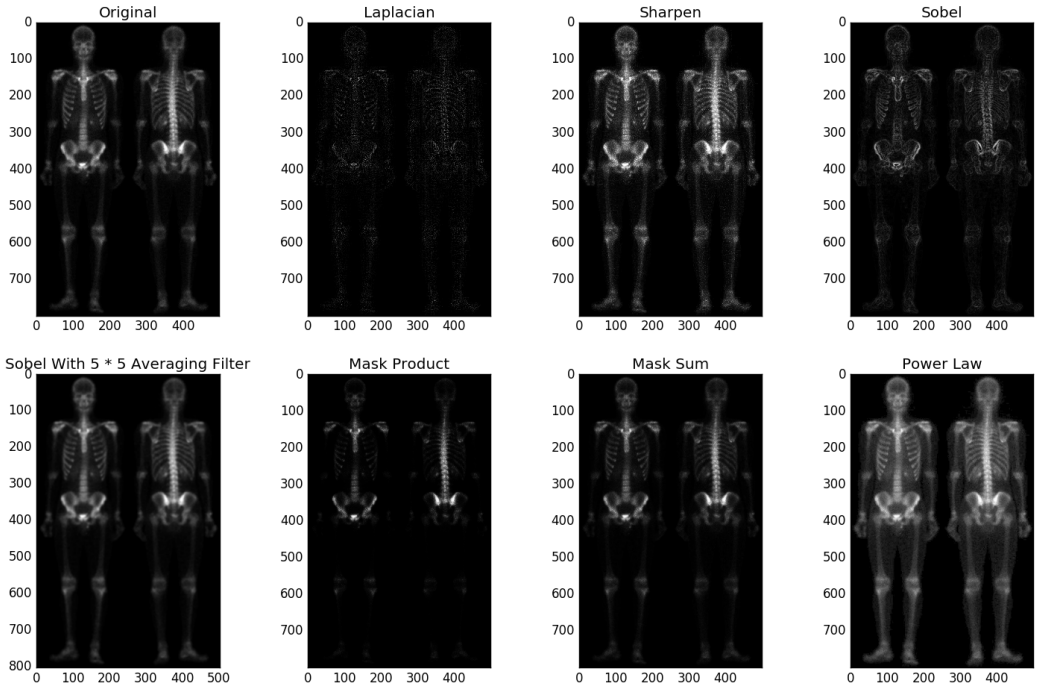
- 左图为对 Sobel 变换后得到的图像进行 5 * 5 的均值掩模变换得到的均值图像，进行均值处理后能够减少噪声，右图为锐化图像与均值图像相乘得到的乘法变换图像，由图片知变换后得到的图像为两图均较亮的像素点



- 左图为原始图像与乘法变换图像相加得到的加法变换图像，右图为对加法变换图像进行幂律变换得到的幂律图像，进行幂律变换后，图像的特征细节更加显著


Mask Sum
Power Law

- 整体展示


Original
Laplacian
Sharpen
Sobel
Sobel With 5 * 5 Averaging Filter
Mask Product
Mask Sum
Power Law