Part 1: Laplacian filter in spatial domain

Method:

Just simply implement convolution on original image and the Laplacian kernel. The sharpened image is acquired by subtracting the convolution result to original image. I used the following two Laplacian kernels.

```
laplacian_kernel = np.array([
    [0, 1, 0],
    [1, -4, 1],
    [0, 1, 0]
])
```

```
laplacian_kernel2 = np.array([
    [1, 1, 1],
    [1, -8, 1],
    [1, 1, 1]
])
```

The convolution function I implemented is as follows. First, compute the padding size according to the kernel size, and use np.pad() to apply the reflect padding. Next, just use a sliding window to compute all the pixel value.

```
def convolve2d(image, kernel):

    img_h, img_w = image.shape
    k_h, k_w = kernel.shape
    #Caculate padding size
    pad_h = k_h // 2
    pad_w = k_w // 2
    padded = np.pad(image, ((pad_h, pad_h), (pad_w, pad_w)), mode='reflect')
    output = np.zeros_like(image)
    for i in range(pad_h, pad_h + img_h):
        for j in range(pad_w, pad_w + img_w):
            tmp = padded[i - pad_h : i + pad_h + 1, j - pad_w : j + pad_w + 1]
            output[i-pad_h, j-pad_w] = np.sum(tmp * kernel)
    return output
```

Compute the Laplacian result using two filters and the sharpening image is the original image minus the Laplacian result.

```
img_gray_sharpening = convolve2d(img, laplacian_kernel)
img_gray_sharpening = img - img_gray_sharpening
img_gray_sharpening = np.clip(img_gray_sharpening, 0, 255).astype(np.uint8)
result_file = 'spatial_laplacian1_' + file
cv2.imwrite(os.path.join('./HW3', result_file), img_gray_sharpening)
```

Result: Original Image and the sharpening result using two different filters

Part2: Laplacian filter in frequency domain

Method:

I follow the following scenarios to acquire the result.

1. I multiply all the pixels with (-1)^(x+y) to let the low frequency part at the center of the original image

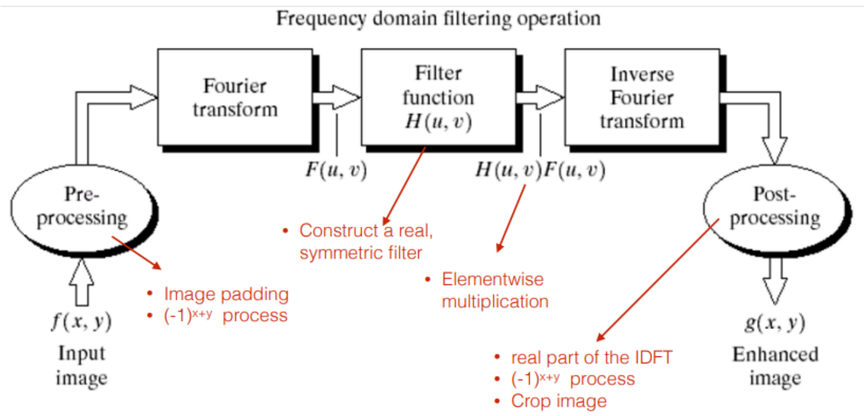
2. Apply Fourier transform on the preprocessed image

3. Construct the Laplacian filter at the frequency domain which the value is symmetric to the center of the image, and I normalize the value of H(u,v) to between (-1, 1)

4. Apply element-wise multiplication.

5. Apply inverse Fourier transforms to the result from 4.

6. Finally, multiply all the pixels with (-1)^(x+y) to acquire the convolution result.



On lecture slide

The following is the code for the above six steps

```
img = cv2.imread(os.path.join(path, file), cv2.IMREAD_GRAYSCALE)
img_float = np.float64(img)
row, col = img.shape
crow, ccol = row // 2, col // 2
# times(-1^(x+y)) to shift the low frequency to the center
for i in range(row):
    for j in range(col):
        img_float[i, j] *= ((-1) ** (i + j))
# Perform FFT
f = np.fft.fft2(img_float)
# construct laplacian filter in frequency domain
laplacian_filter = np.zeros((row, col), dtype=np.float64)
for u in range(row):
    for v in range(col):
        laplacian_filter[u, v] = -4 * np.pi**2 * ((u - crow) ** 2 + (v - ccol) ** 2)
laplacian_filter /= np.max(np.abs(laplacian_filter))
# bit-wise multiplication in frequency domain
f_laplacian = f * laplacian_filter
# Perform inverse FFT
img_back = np.fft.ifft2(f_laplacian)
img_back = np.real(img_back)
# times(-1^(x+y)) to shift the low frequency to the center
for i in range(row):
    for j in range(col):
        img_back[i, j] *= ((-1) ** (i + j))
```

After acquiring the convolution result, we need to subtract it to the original image. I multiply the convolution result by k = 10 to get a better result.

```
print(np.max(img_back))
print(np.min(img_back))
sharpened_img = img - k * img_back
sharpened_img = np.clip(sharpened_img, 0, 255).astype(np.uint8)
result_file = 'laplacian_frequency_' + file
cv2.imwrite(os.path.join('./HW3', result_file), sharpened_img)
```

Result:

Comparison of the two methods:

The Fourier transform method is relatively more complex to implement compared to spatial-domain approaches. However, it offers significantly higher computational efficiency when dealing with large kernel sizes.
In terms of visual results, the output image from the frequency-domain method appears slightly noisier. This may be attributed to its stronger amplification of high-frequency components.
On the other hand, convolution performed in the spatial domain tends to yield more natural-looking results, likely due to its localized nature and reduced sensitivity to high-frequency noise.

Feedback:

This assignment was my first time implementing image sharpening, and I was able to produce correct results. It was also the first time I truly implemented and observed the property that convolution in the spatial domain is equivalent to element-wise multiplication in the frequency domain. The output image closely matched the result of spatial convolution, which made me appreciate and believe in this fascinating property even more.