Homework 2: Image Enhancement Using Spatial Filters

Part I. Implementation (5%):

1. **padding:** I use zero padding here. First create a new numpy array with new_height and new_width. All the values are set to 0. The value of new_height and new_width is from kernel size. Copy the original image to the center of new numpy array.

```
def padding(input_img, kernel_height, kernel_width):
    ############### YOUR CODE STARTS HERE ###############
    i_height, i_width = input_img.shape
    pad_height, pad_witdh = kernel_height // 2, kernel_width // 2

    new_height, new_width = i_height + 2 * pad_height, i_width + 2 * pad_witdh
    output_img = np.zeros((new_height, new_width), dtype=input_img.dtype)

    output_img[pad_height : pad_height + i_height, pad_witdh : pad_witdh + i_width] = input_img
    ################ YOUR CODE ENDS HERE #################
    return output_img
```

2. **convolution:** The input is image and kernel array. First get new image by zero padding and then iterate each pixel of input image. Each pixel mutiply with kernel array to get new pixel value.

```
def convolution(input_img, kernel):
    ################ YOUR CODE STARTS HERE ################
    k_height, k_width = kernel.shape
    i_height, i_width = input_img.shape

    padded_img = padding(input_img, k_height, k_width)
    output_img = np.zeros_like(input_img, dtype=np.float64)

    for y in range(i_height):
        for x in range(i_width):
            roi = padded_img[y : y + k_height, x : x + k_width]
            output_img[y, x] = (roi * kernel).sum()
    ################ YOUR CODE ENDS HERE #################
    return output_img
```

3. **Gaussian filter:** First, get b, g, r array. Second, initialize gaussian kernel. Third, use b, g, r array and gaussian kernel to convolution. Finally, merge the filtered b, g, r to get gaussian filtered image.

```
def gaussian_filter(input_img, kernel_size, sigma):
    ################ YOUR CODE STARTS HERE ################
    b, g, r = cv2.split(input_img)

    center = kernel_size // 2
    x, y = np.mgrid[-center:center+1, -center:center+1]
    gaussian_kernel = np.exp(-(x**2+y**2) / (2 * sigma**2))

    gaussian_kernel = gaussian_kernel / gaussian_kernel.sum()

    b_filter = convolution(b, gaussian_kernel)
    g_filter = convolution(g, gaussian_kernel)
    r_filter = convolution(r, gaussian_kernel)

    output_img = cv2.merge((b_filter, g_filter, r_filter))
    output_img = np.clip(output_img, 0, 255)
    ################ YOUR CODE ENDS HERE ##################
    return output_img.astype(np.uint8)
```

4. Median filter:
   a. **median_filter_convolution:** The convolution which is specific for median filter. Ea ch pixel is set to the median value of the corresponding kernel.

```
def median_filter_convolution(input_img, kernel_size):
    i_height, i_width = input_img.shape

    padded_img = padding(input_img, kernel_size, kernel_size)
    output_img = np.zeros_like(input_img, dtype=np.float64)

    for y in range(i_height):
        for x in range(i_width):
            roi = padded_img[y: y + kernel_size, x : x + kernel_size]
            output_img[y, x] = np.median(roi)
    return output_img
```

   b. **median_filter:** First, get b, g, r numpy array. Second, get filtered b, g, r by convolu tion. Finally, merge b, g, r to get median filtered image.

```
def median_filter(input_img, kernel_size):
    ################ YOUR CODE STARTS HERE ################
    b, g, r = cv2.split(input_img)

    b_filter = median_filter_convolution(b, kernel_size)
    g_filter = median_filter_convolution(g, kernel_size)
    r_filter = median_filter_convolution(r, kernel_size)

    output_img = cv2.merge((b_filter, g_filter, r_filter))
    output_img = np.clip(output_img, 0, 255)
    ################ YOUR CODE ENDS HERE ##################
    return output_img.astype(np.uint8)
```

5. **Laplacian filter:** First, get b, g, r numpy array. Second, get filtered b, g, r by convolution. (The kernel is initialized in main) Finally, merge b, g, r to get median filtered image.

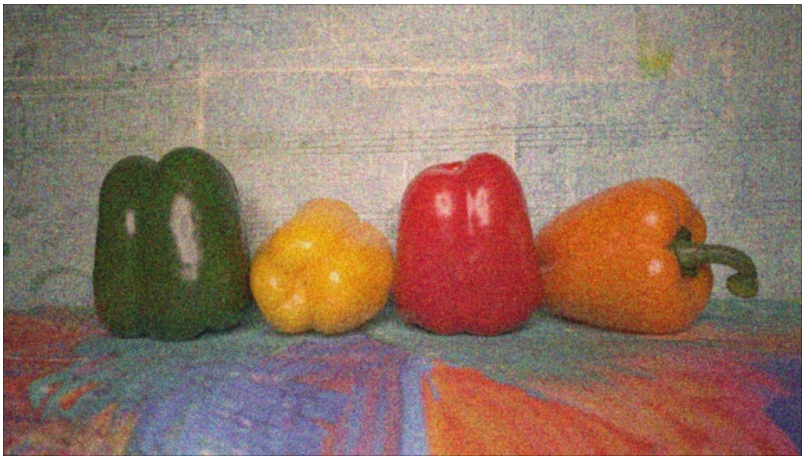
```python
def laplacian_sharpening(input_img, kernel):
    ################ YOUR CODE STARTS HERE ################
    b, g, r = cv2.split(input_img)

    b_filter = convolution(b, kernel)
    g_filter = convolution(g, kernel)
    r_filter = convolution(r, kernel)

    output_img = cv2.merge((b_filter, g_filter, r_filter))
    output_img = np.clip(output_img, 0, 255)
    ################ YOUR CODE ENDS HERE ##################
    return output_img.astype(np.uint8)
```

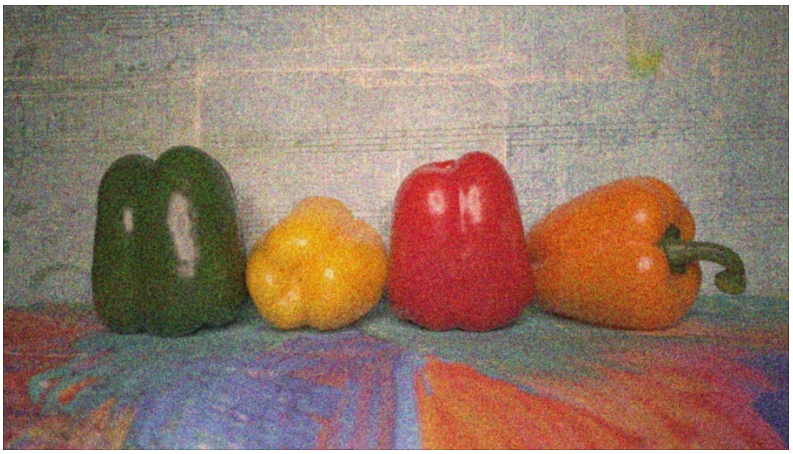
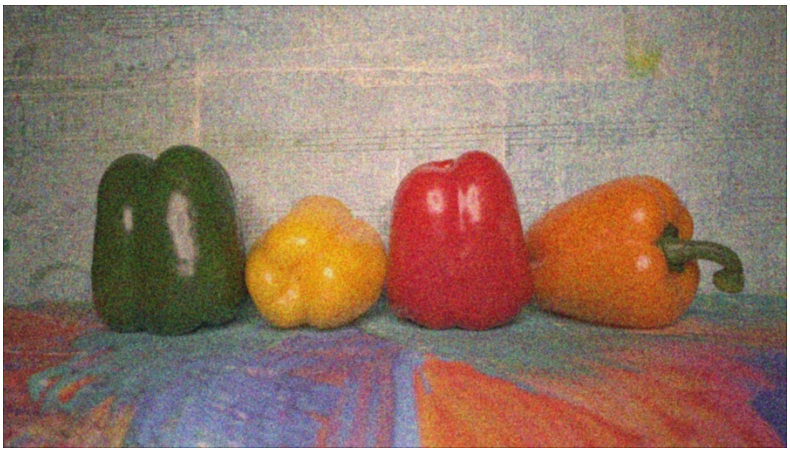
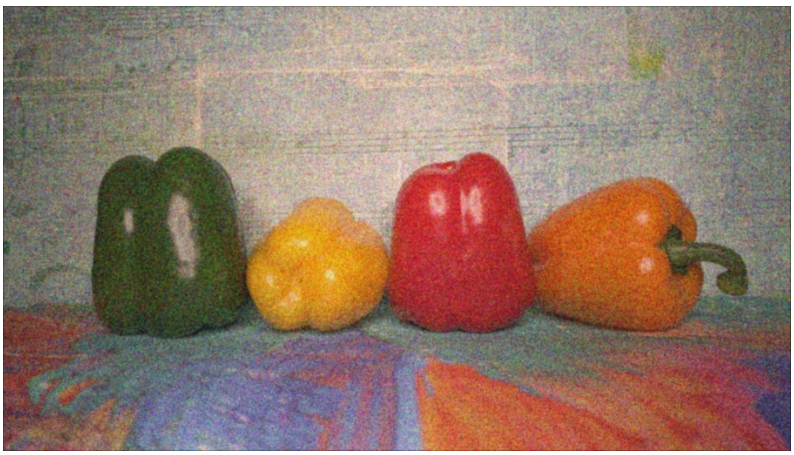
6. main:
   a. **gaussian:** I tried [3, 7, 11] kernel size and [1.0, 2.0, 3.0] sigma. Save each of imag e.
   b. **median:** I tried [3, 7, 11] kernel size and save each of image.
   c. **laplacian:** Initialize two kernels and convert them to numpy arrays. Save each of th em.

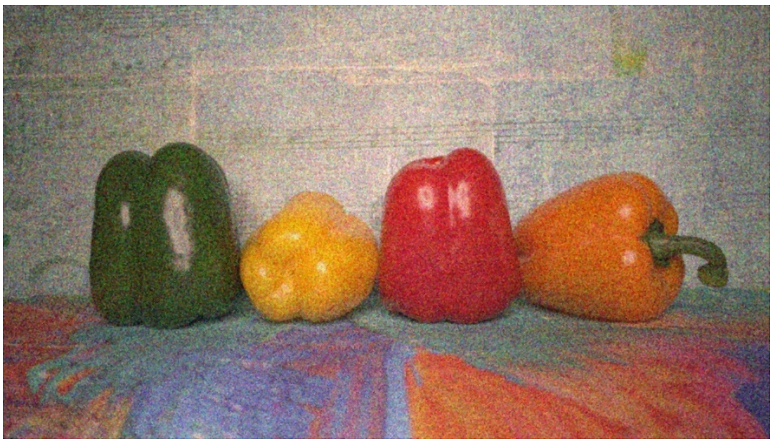
```python
if __name__ == "__main__":
    args = parse_args()

    if args.gaussian:
        input_img = cv2.imread("input_part1.jpg")
        kernel_sizes = [3, 7, 11]
        sigmas = [1.0, 2.0, 3.0]
        for kernel_size in kernel_sizes:
            for sigma in sigmas:
                output_img = gaussian_filter(input_img, kernel_size, sigma)
                cv2.imwrite(f"output_gaussian_{kernel_size}_{sigma}.jpg", output_img)
    elif args.median:
        input_img = cv2.imread("input_part1.jpg")
        kernel_sizes = [3, 7, 11]
        for kernel_size in kernel_sizes:
            output_img = median_filter(input_img, kernel_size)
            cv2.imwrite(f"output_median_{kernel_size}.jpg", output_img)
    elif args.laplacian:
        input_img = cv2.imread("input_part2.jpg")
        kernel_list = [
            [[0, -1, 0],
            [-1, 5, -1],
            [0, -1, 0]],

            [[-1, -1, -1],
            [-1,  9, -1],
            [-1, -1, -1]]
        ]
        kernel_np = np.array(kernel_list)
        for i, kernel in enumerate(kernel_np):
            output_img = laplacian_sharpening(input_img, kernel)
            cv2.imwrite(f"output_laplacian_{i+1}.jpg", output_img)
```

Part II. Results & Analysis (10%):

Please provide your observations and analysis for the following parts.

- Gaussian filter
  - **Sigma comparing:** The larger the sigma, the more blur the image.
  - kernel size: 7x7, σ :1.0:
  - 
  - kernel size: 7x7, σ :2.0:
  - 
  - kernel size: 7x7, σ :3.0:
  - 

- **Kernel comparing:** The larger the kernel size, the more blur the image. The difference between images is not clearly if the sigma is low.
- kernel size: 3x3, σ:1.0:
- 
- kernel size: 7x7, σ:1.0:
- 
- kernel size: 11x11, σ:1.0:
- 

- **Median filter:** The larger the kernel size, the more blur the image.
  - kernel size: 3x3
  - 
  - kernel size: 7x7
  - 
  - kernel size: 11x11
  - 

- Smoothing Spatial Filters
  - The sigma in gaussian filter should be set bigger to make the image more blur. By t hat, the noise can be more unclear.
  - If I set the sigma of gaussian filter to 3, the result of gaussian filter and median filt er are similiar.
  - Gaussian filter: kernel size: 11x11, sigma: 3.0
  - 
  - Median filter: kernel size: 11x11
  - 

- Laplacian filter
  - The result of filter 2 looks sharper.

0	-1	0
-1	5	-1
0	-1	0

Filter 1

-1	-1	-1
-1	9	-1
-1	-1	-1

Filter 2

Filter 1:

Filter 2:

Part III. Answer the questions (15%):

1. Please describe a problem you encountered and how you solved it.

   A: I was thinking filter gray image or color image. I tried both and finally filter color imag e as my result.

   I forgot to clip the pixel value and convert it to integer. I added it in return.

2. What padding method do you use, and does it have any disadvantages? If so, please sugges t possible solutions to address them.

   A: I use zero padding. The disadvantage is different in different filters.

   Gaussian filter: Cause the edge to be darker to it should be.

   Median filter: If the kernel size is small, it will skew the median calculation.

   Laplacian filter: The sharp transition between the original edge and zero-padding boarde r can be falsely detected an edge, creating a distracting, artificial border in the sharpened re sult.

   I think for most cases, **Reflect Padding** is the best choice. It is because it reflect the imag e edge and make it look like the extension of the image.

3. What problems do you encounter when using Gaussian filter and median filter to denoise i mages? Please suggest possible solutions to address them.

   A: **Gaussian filter:** It blurs everything, including the important edges that define objects in an image.

   Solution: Bilateral Filter. It avoids averaging pixels with very different brightness value s, which allows it to smooth surfaces while keeping the sharp edges intact.

   Median filter: It may remove fine details like thin lines or textures, making the image ap pear unnaturally flat.

   Solution: Use the smallest effective kernel. It can effective removes the noise.

4. What problems do you encounter when using Laplacian filters to sharpen images? Please s uggest possible solutions to address them.

   A: The main disadvantage of the Laplacian filter is its extreme sensitivity to noise, which it drastically amplifies. The best solution is to smooth the image first by applying a gentle Ga ussian blur. This technique, known as Laplacian of Gaussian (LoG), sharpens edges withou t amplifying the original noise.