Homework 1: Image Enhancement Using Spatial Filters

Report Template

Please keep the title of each section, and note that the questions listed in Part III should be retained.

Part I. Implementation (5%):

Please provide screenshots of your code snippets, and explain your implementation.

```python
import numpy as np
import cv2
import argparse
import math
def parse_args():
    parser = argparse.ArgumentParser()
    parser.add_argument('--gaussian', action='store_true',default=False)
    parser.add_argument('--median', action='store_false',default=False)
    parser.add_argument('--laplacian', action='store_false',default=True)
    parser.add_argument('--kernelSize',type=int, default=3)
    parser.add_argument('--sigma',default=3)
    parser.add_argument('--filter',type=bool ,default=True,help="For laplacian filter, choosing filter 1 or 2, True for filter 1, False for filter 2")
    args = parser.parse_args()
    return args

def padding(input_img, kernel_size):
    ############## YOUR CODE STARTS HERE ##############

    # Zero padding

    # the paddingSize is defined by how much 0 we need to pad around corner pixel,for example:
    # kernel size=3 , picture is 3*4
    #           oooooo
    #           oxxxxo
    #           oxxxxo
    #           oxxxxo
    #           oooooo
    # o is padding zero
    # after padding, the size become (3+2*1) * (4+2*1)
    paddingSize=kernel_size//2
    im_Height,im_Width=input_img.shape[0],input_img.shape[1]

    #create the return img
    output_img=np.zeros([im_Height+2*paddingSize,im_Width+2*paddingSize,3])

    for i in range(im_Height+2*paddingSize):
        for j in range (im_Width+2*paddingSize):
            for c in range (3):# RGB
                if i<paddingSize or j<paddingSize or i>=im_Height+paddingSize or j>=im_Width+paddingSize:
                    output_img[i,j,c]=0 # padding zero
                else :
                    output_img[i,j,c]= input_img[i-paddingSize,j-paddingSize,c] #keep input pixel value
    ############## YOUR CODE ENDS HERE ################
    return output_img

def convolution(input_img, kernel):
    ############## YOUR CODE STARTS HERE ##############
    kernel_size=kernel.shape[0]
    padImage=padding(input_img,kernel_size) # first, zero-padding for the convolution

    output_img=np.zeros_like(input_img) # output_img should be the same size with input_img

    im_Height,im_Width=input_img.shape[0],input_img.shape[1]

    for i in range(im_Height):
        for j in range (im_Width):
            for c in range (3): # color : RGB
                region=padImage[i:i+kernel_size,j:j+kernel_size,c] # define the convoltion region
                output_img[i,j,c]=abs(np.sum(region*kernel)) # do convolution

    ############## YOUR CODE ENDS HERE ################
    return output_img
```

```python
def gaussian_filter(input_img):
    ############### YOUR CODE STARTS HERE ###############
    args=parse_args()
    kernelSize=args.kernelSize

    kernel = np.zeros([kernelSize,kernelSize]) # create kernel
    sum_val=0
    mid=kernelSize//2
    sigma=args.sigma
    for i in range(kernelSize):
        for j in range (kernelSize):
            di=i-mid
            dj=j-mid
            kernel[i,j]=np.exp(-(di**2+dj**2)/2*(sigma**2))/(2*math.pi*(sigma**2)) # assign value into kernel using the gaussian function
            sum_val+=kernel[i,j]

    kernel=kernel/sum_val # do regularation so that the sum of kernel index is 1
    ############### YOUR CODE ENDS HERE #################
    return convolution(input_img, kernel)

def median_filter(input_img):
    ############### YOUR CODE STARTS HERE ###############

    # Applying median filter without padding:
    # Edge and corner pixels retain their original values as the kernel
    # cannot fully cover them. Only fully covered interior pixels are filtered,
    # preserving boundary information and avoiding padding artifacts.
    # for example : 5*6 picture, 3*3 kernel size
    # c for corner pixel, m for mid pixel(will be changed by median filter)
    #    cccccc
    #    cmmmmc
    #    cmmmmc
    #    cmmmmc
    #    cccccc

    args=parse_args()
    kernelSize=args.kernelSize
    mid=kernelSize//2

    output_img=input_img
    im_Height,im_Width=input_img.shape[0],input_img.shape[1]

    for i in range(mid,im_Height-mid): #ignore from 0 to mid
        for j in range (mid,im_Width-mid): # ignore from 0 to mid
            for c in range (3): # color : RGB
                region=input_img[i-mid:i+mid+1,j-mid:j+mid+1,c]
                output_img[i,j,c]=np.median(region) # assign the median of kernel
    ############### YOUR CODE ENDS HERE #################
    return output_img

def laplacian_sharpening(input_img):
    ############### YOUR CODE STARTS HERE ###############
    args=parse_args()

    filter1=np.array([[0,-1,0],
                      [-1,5,-1],
                      [0,-1,0]])
    filter2=np.array([[-1,-1,-1],
                      [-1,9,-1],
                      [-1,-1,-1]])
    # using argument to decide filter 1 or 2
    kernel=filter1 if args.filter else filter2
    ############### YOUR CODE ENDS HERE #################
    return convolution(input_img, kernel)
```

The comment of code include the detail of implementation.

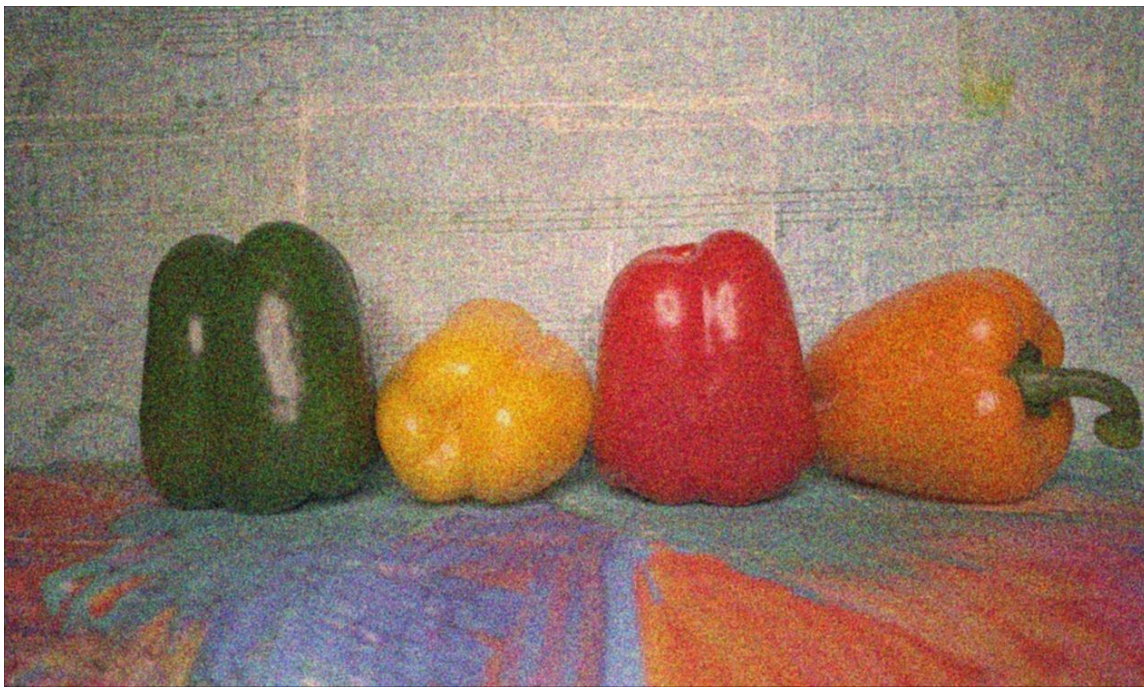
For the convolution and median filter, separately do the convolution (median) for three channel (RGB)

Part II. Results & Analysis (10%):

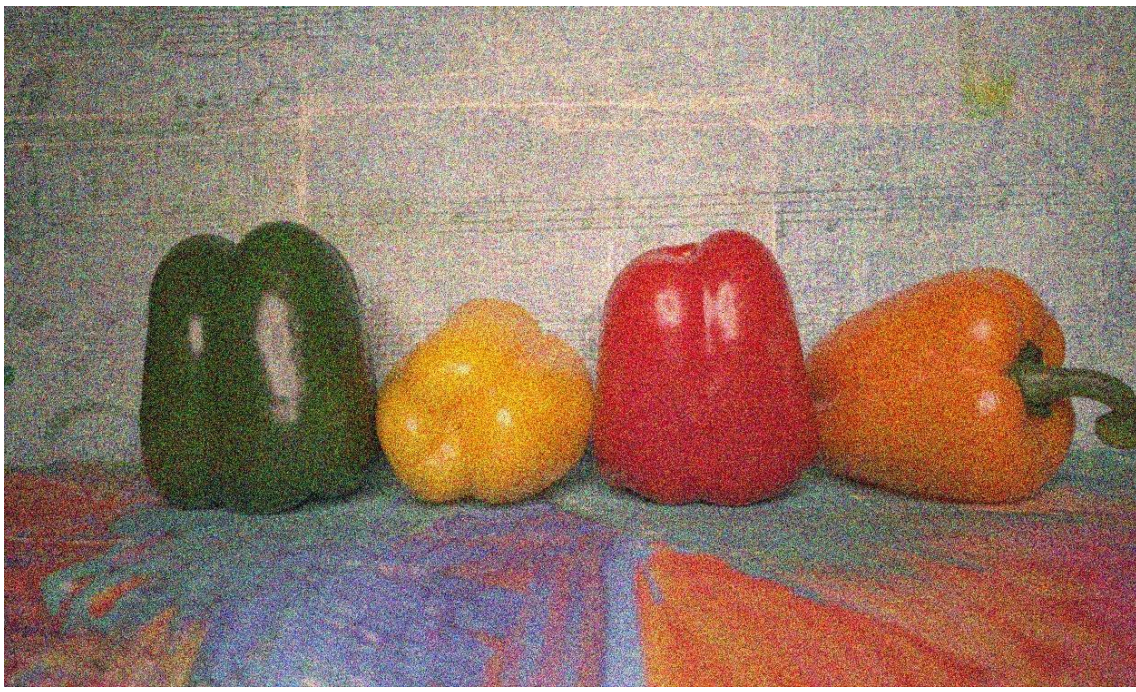
Please provide your observations and analysis for the following parts.

- **Gaussian filter**
  - **Try three different σ and compare the results**
    - **Fixed Kernel size=3, σ=**

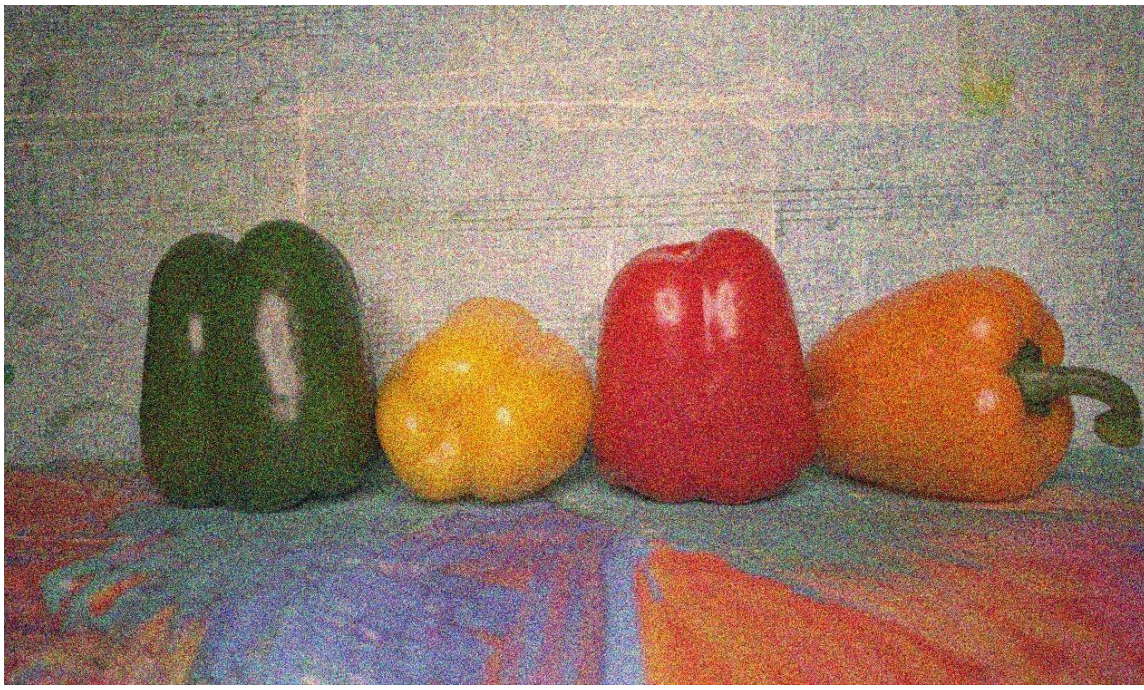
a. 1



b. 3



c. 10



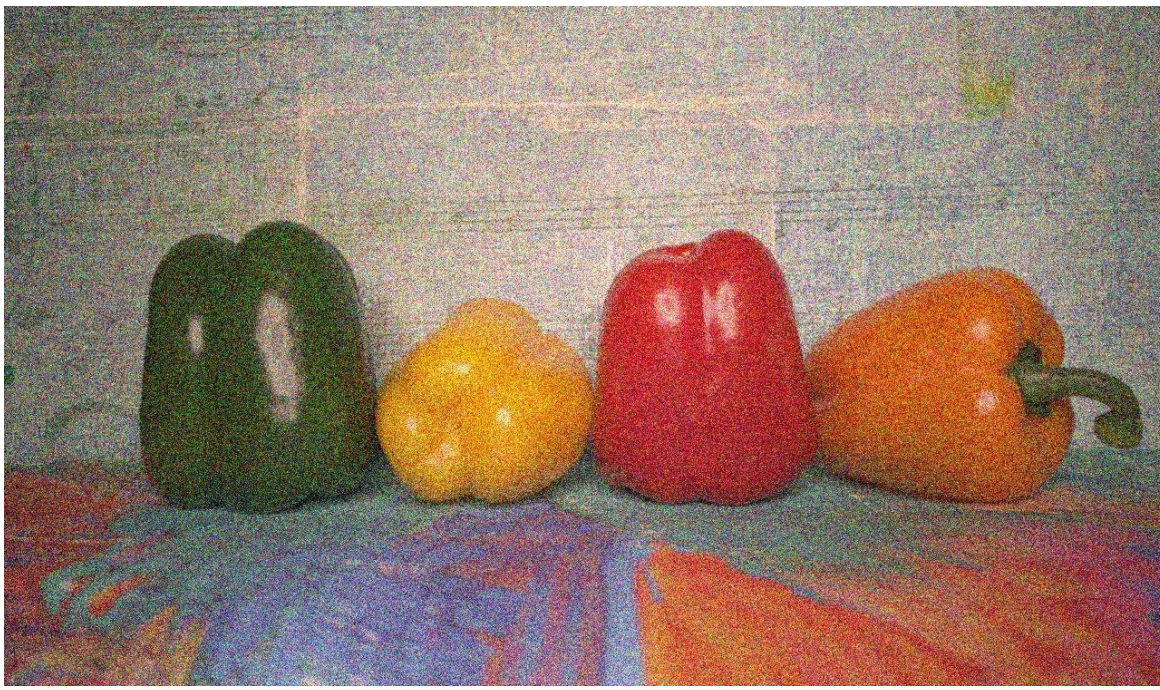
Low sigma (σ=1): The filter applies minimal smoothing, so noise remains prominent. The image retains more detail but displays visible graininess, especially in flat or uniform area

Moderate sigma (σ=3): The noise is partially reduced. The image appears slightly smoother, with fewer bright speckles, though some noise grains are still visible. This level of filtering provides a balance between smoothing and detail preservation.

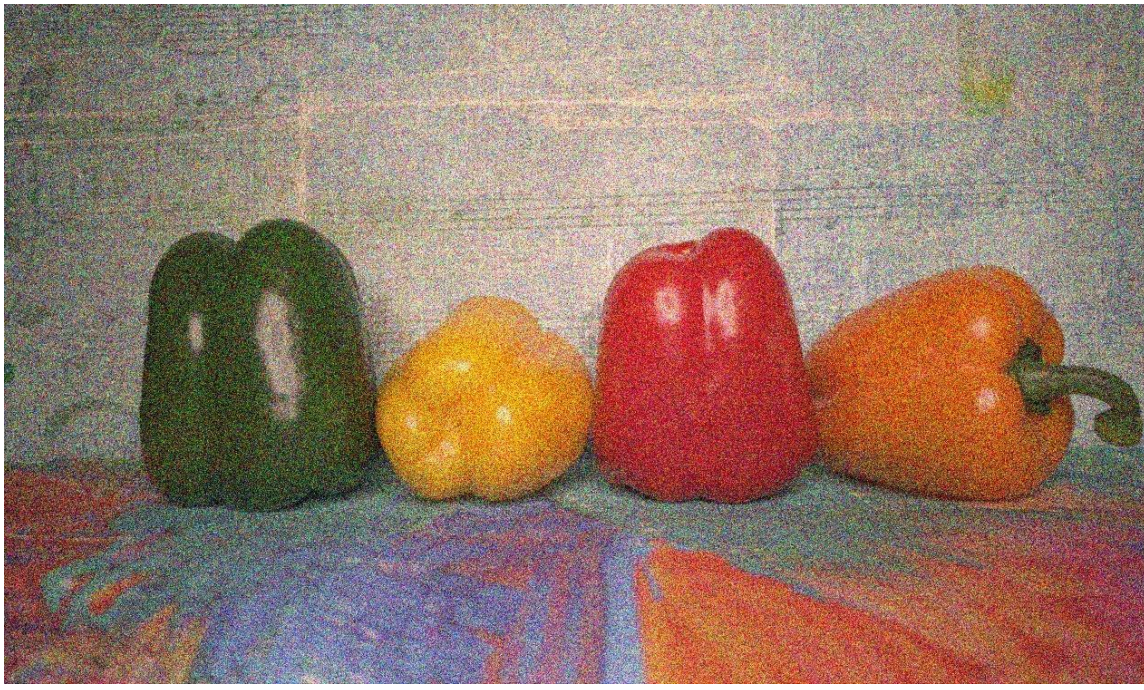
High Sigma (σ=10): A larger sigma value results in strong smoothing, significantly reducing noise. The image becomes very soft, with most random grains nearly invisible. However, fine details are also blurred, leading to a loss in sharpness.

- **Try three different filter sizes and compare the results.**
  - **Fixed σ=3,kernel size=**

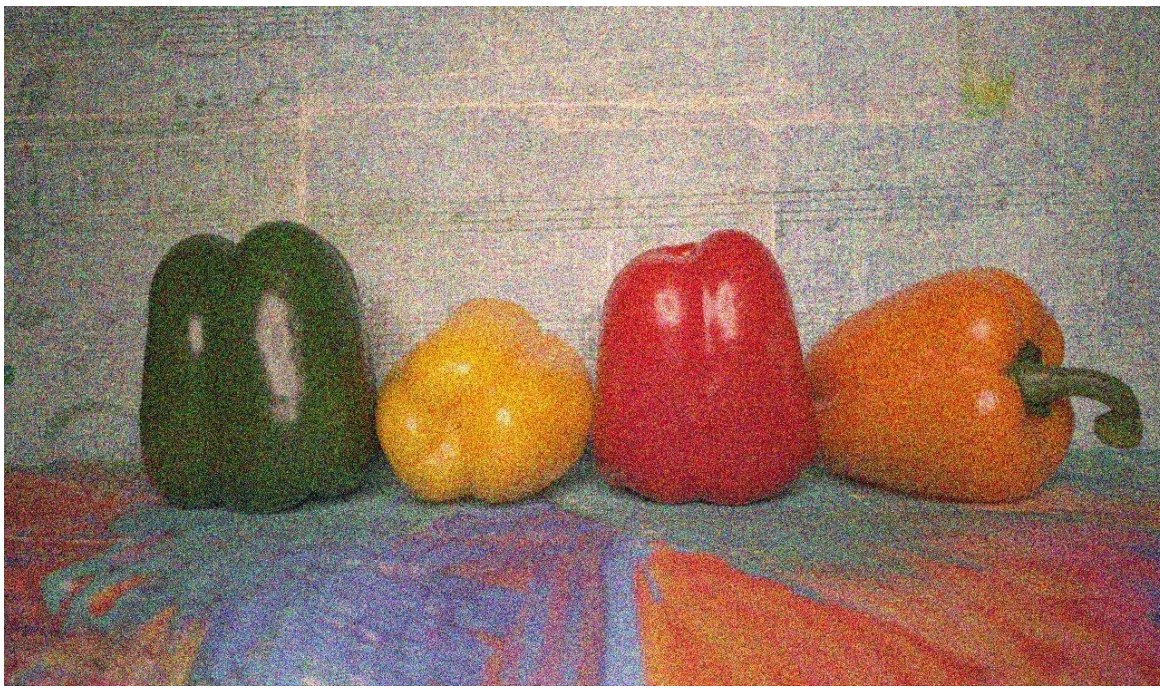
a. 3



b. 9



c. 15



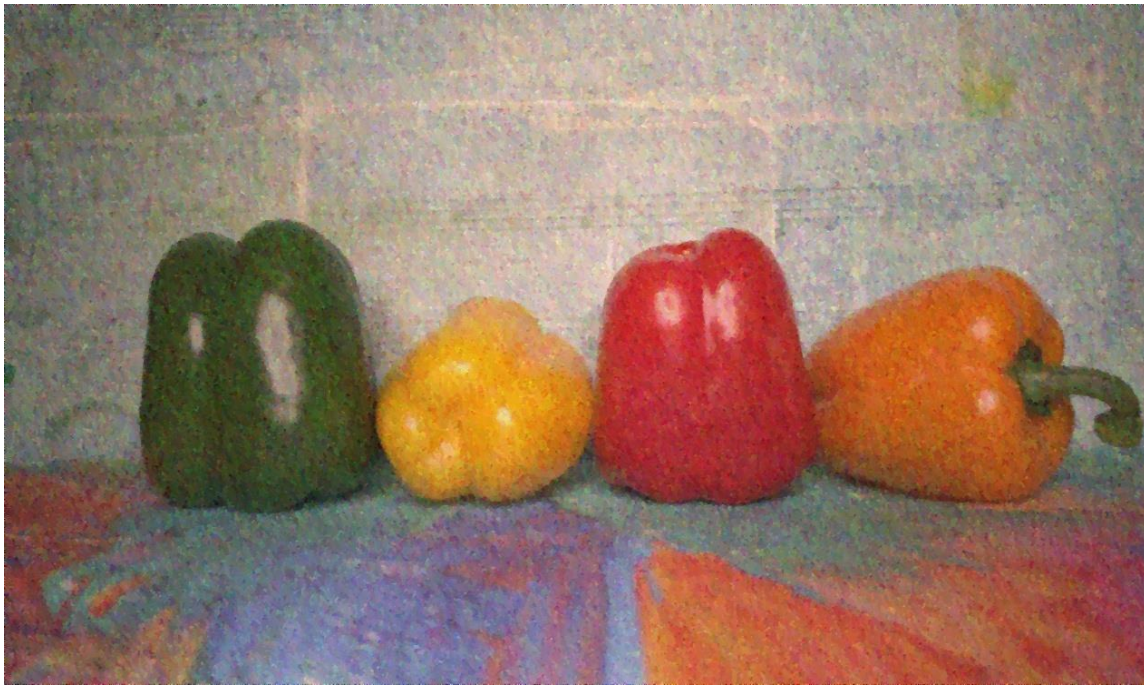
Kernel Size = 3: This image should exhibit the least amount of blurring. Only minor smoothing is applied, so the noise remains quite visible. With a small kernel size, the filter has a limited effect and mainly smooths out noise within a small area, while preserving most of the fine details.

Kernel Size = 9: In this image, the blurring effect is more noticeable, and the details start to soften slightly, with some of the noise being smoothed out. A medium-sized kernel covers a broader area, reducing noise to a moderate extent while still retaining some image detail.

Kernel Size = 15: This image should show the strongest blurring effect, with most noise almost completely smoothed away, but at the cost of losing finer details. A large kernel size results in significant smoothing, effectively removing a large amount of noise, but it also makes the image look much softer.

- **Median filter**
    - **Try three different filter sizes and compare the results.**

a. Kernel size = 3



b. Kernel size = 5

c. Kernel size = 9

Kernel size = 3: A small kernel size mainly targets isolated noise and effectively removes small specks without significantly affecting the details of the image. Edges and textures are well-preserved, with minimal blurring.

Kernel size = 5: With a medium kernel size, the filter applies more noticeable smoothing, reducing larger clusters of noise while beginning to blur some of the finer details. The edges become slightly softened, although they retain more definition compared to Gaussian blurring.

Kernel size = 9: A large kernel size produces strong smoothing, removing a substantial amount of noise but also sacrificing more details. The image starts to look blocky, with reduced edge sharpness and fewer fine details. This creates a “patchy” or “painterly” effect, where the image appears as if it’s made up of larger color patches.

- **Smoothing Spatial Filters**
  - **Compare the results of Gaussian filter and median filter.**

    Gaussian Filter: Uses a weighted average, creating a smooth, natural blur. Effective for reducing Gaussian noise but may blur edges and fine details.
    Median Filter: Replaces each pixel with the median value of its neighborhood. Ideal for removing salt-and-pepper noise, preserving edges better but resulting in a “blocky” look with larger kernels.

- **Laplacian filter**
  - **Compare the results of two laplacian filters.**

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

Filter 1

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

Filter 2

Filter1:

Filter 2:

Filter 1: This filter has a smaller central value (5) and lower values around it. It provides a mild sharpening effect, enhancing edges but without overly exaggerating contrast.

Filter 2 : This filter uses a larger central value (9) with all surrounding values set to -1. It creates a much stronger sharpening effect, making edges stand out more prominently.

Part III. Answer the questions (15%):

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

   Since this is my first time taking an image processing course, I initially struggled to understand the shape of images when they're converted into arrays. I spent some time looking up references to get a clearer understanding.

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

   I used zero padding. The disadvantage of this method is that it limits the effectiveness of the median filter at the edges and corners, as those areas cannot be fully processed. A potential solution is to use another padding method, such as reflect padding, which mirrors the edge pixels to better

preserve the image structure. However, implementing the correct logic for this can be challenging to me.

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

   Gaussian Filter: While it effectively smooths Gaussian noise, it also tends to blur edges and fine details, which can reduce image sharpness.

   Solution: Apply Gaussian filter selectively, focusing on noisy areas, or use an edge-preserving filter like the bilateral filter to reduce noise while retaining edges.

   Median Filter: This filter is very effective for salt-and-pepper noise but can produce a blocky or patchy appearance, especially with larger kernel sizes. It may also miss some Gaussian noise.

   Solution: Use a smaller kernel size to reduce blockiness or combine the median filter with other techniques (e.g., Gaussian filter) to handle different types of noise effectively.

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

   When using Laplacian filters to sharpen images, I encountered the following issues:

   Increased Noise: Laplacian sharpening tends to amplify noise along with edges, especially in images with high noise levels. Solution: Apply a denoising filter (like Gaussian) before using the Laplacian filter to reduce noise while sharpening edges.

   Over-sharpening Artifacts: Strong sharpening can create halo effects or unnatural contrast around edges, which may reduce image quality. Solution: Control the intensity of the Laplacian filter by blending the sharpened image with the original or using a lower-weighted kernel to achieve a subtler effect.