INT3412E 20 - Computer Vision: Homeworks 2

Lưu Văn Đức Thiệu

1 Image filtering

1.1 Prove linearity

Mean filtering is linear:
We will prove that: mean_filter(X + Y) = mean_filter(X) + mean_filter(Y) $\forall$ X, Y $\in R^{m,n}$
Set Z = X + Y so $\forall$ i $\in$ m, j $\in$ n we have: $Z_{i,j} = X_{i,j} + Y_{i,j}$
$Let X' = mean\_filter(X), \quad Y' = mean\_filter(Y) \quad \text{and} \quad Z' = mean\_filter(Z) \quad \Rightarrow \quad \forall i \in m, j \in n$ we have :

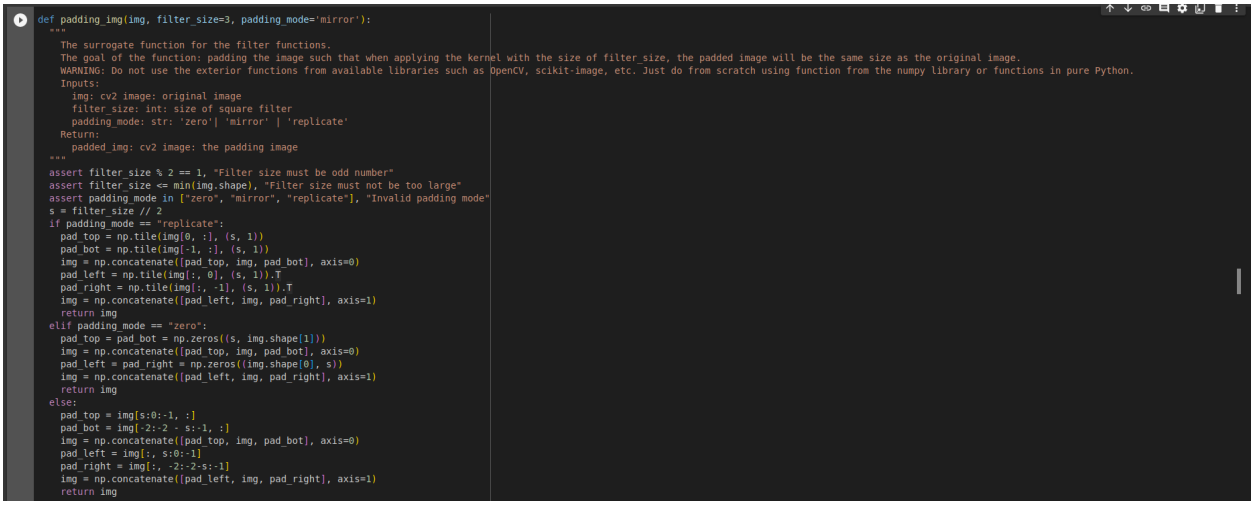
$$X'_{i,j} = \frac{1}{9} * \sum_{p=-1}^{1} \sum_{q=-1}^{1} X_{i+p,j+q} \quad \text{and} \quad Y'_{i,j} = \frac{1}{9} * \sum_{p=-1}^{1} \sum_{q=-1}^{1} Y_{i+p,j+q}$$

$$\begin{aligned} Z'_{i,j} &= \frac{1}{9} * \sum_{p=-1}^{1} \sum_{q=-1}^{1} Z_{i+p,j+q} \\ &= \frac{1}{9} * \sum_{p=-1}^{1} \sum_{q=-1}^{1} X_{i+p,j+q} + Y_{i+p,j+q} \\ &= \frac{1}{9} * \sum_{p=-1}^{1} \sum_{q=-1}^{1} X_{i+p,j+q} + \frac{1}{9} * \sum_{p=-1}^{1} \sum_{q=-1}^{1} Y_{i+p,j+q} \\ &= X'_{i,j} + Y'i, j \\ \Rightarrow \quad & Z' = X' + Y' \\ \Rightarrow \quad & mean\_filter(X + Y) = mean\_filter(X) + mean\_filter(Y) \end{aligned}$$

Median filtering is not linear:
I will prove it by counter-proof: Assume we have two matrix $X = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \end{bmatrix}$ and $Y = \begin{bmatrix} 4 & 15 & 13 & 2 & 10 \\ 11 & 7 & 8 & 9 & 1 \\ 3 & 5 & 12 & 14 & 6 \end{bmatrix} \Rightarrow median\_filter(X) = \begin{bmatrix} 7 & 8 & 9 \end{bmatrix}$ and $median\_filter(Y) = \begin{bmatrix} 8 & 9 & 9 \end{bmatrix}$, sum of them is $\begin{bmatrix} 15 & 17 & 18 \end{bmatrix}$ meanwhile median_filter(X + Y) is $\begin{bmatrix} 16 & 17 & 16 \end{bmatrix}$ so median filtering is not linear because it doesn't satisfy linear condition.

1.2 Implement padding and filter functions

- Implementation of padding function 1
- Implementation of mean filter function 2
- Implementation of median filter function 3



```python
def padding_img(img, filter_size=3, padding_mode='mirror'):
  """
    The surrogate function for the filter functions.
    The goal of the function: padding the image such that when applying the kernel with the size of filter_size, the padded image will be the same size as the original image.
    WARNING: Do not use the exterior functions from available libraries such as OpenCV, scikit-image, etc. Just do from scratch using function from the numpy library or functions in pure Python.
    Inputs:
      img: cv2 image: original image
      filter_size: int: size of square filter
      padding_mode: str: 'zero'| 'mirror' | 'replicate'
    Return:
      padded_img: cv2 image: the padding image
  """
  assert filter_size % 2 == 1, "Filter size must be odd number"
  assert filter_size <= min(img.shape), "Filter size must not be too large"
  assert padding_mode in ["zero", "mirror", "replicate"], "Invalid padding mode"
  s = filter_size // 2
  if padding_mode == "replicate":
    pad_top = np.tile(img[0, :], (s, 1))
    pad_bot = np.tile(img[-1, :], (s, 1))
    img = np.concatenate([pad_top, img, pad_bot], axis=0)
    pad_left = np.tile(img[:, 0], (s, 1)).T
    pad_right = np.tile(img[:, -1], (s, 1)).T
    img = np.concatenate([pad_left, img, pad_right], axis=1)
    return img
  elif padding_mode == "zero":
    pad_top = pad_bot = np.zeros((s, img.shape[1]))
    img = np.concatenate([pad_top, img, pad_bot], axis=0)
    pad_left = pad_right = np.zeros((img.shape[0], s))
    img = np.concatenate([pad_left, img, pad_right], axis=1)
    return img
  else:
    pad_top = img[s:0:-1, :]
    pad_bot = img[-2:-2 - s:-1, :]
    img = np.concatenate([pad_top, img, pad_bot], axis=0)
    pad_left = img[:, s:0:-1]
    pad_right = img[:, -2:-2-s:-1]
    img = np.concatenate([pad_left, img, pad_right], axis=1)
    return img
```

Figure 1: Padding Function

```python
def mean_filter(img, filter_size=3, padding_mode='mirror'):
  """
  Smoothing image with mean square filter with the size of filter_size.
  WARNING: Do not use the exterior functions from available libraries such as OpenCV, scikit-image, etc. Just do from scratch using function from the numpy library or functions in pure Python.
  Inputs:
    img: cv2 image: original image
    filter_size: int: size of square filter,
    padding_mode: str: 'zero'| 'mirror' | 'replicate'
  Return:
    smoothed_img: cv2 image: the smoothed image with mean filter.
  """

  convolve = np.zeros((img.shape))
  img = padding_img(img, filter_size, padding_mode)
  filter = np.ones((filter_size, filter_size)) / (filter_size**2)
  s = filter_size // 2
  x, y = img.shape
  for v in range(s, x - s):
    for h in range(s, y - s):  #
        area = img[(v - s):(v + s + 1), (h - s): (h + s + 1)]
        convolve[v - s, h - s] = np.sum(np.multiply(filter, area))
  return convolve
```

Figure 2: Mean Filter Function

```python
def median_filter(img, filter_size=3, padding_mode='mirror'):
  """
  Smoothing image with median square filter with the size of filter_size.
  WARNING: Do not use the exterior functions from available libraries such as OpenCV, scikit-image, etc. Just do from scratch using function from the numpy library or functions in pure Python.
  Inputs:
    img: cv2 image: original image
    filter_size: int: size of square filter
    padding_mode: str: 'zero'| 'mirror' | 'replicate'
  Return:
    smoothed_img: cv2 image: the smoothed image with median filter.
  """
  median = np.zeros((img.shape))
  img = padding_img(img, filter_size, padding_mode)
  s = filter_size // 2
  x, y = img.shape
  for v in range(s, x - s):
    for h in range(s, y - s):
        area = img[(v - s):(v + s + 1), (h - s): (h + s + 1)]
        median[v - s, h - s] = np.median(area)
  return median
```

Figure 3: Median Filter Function

```python
def mse(gt_img, smooth_img):
  """
  Calculate the Mean Square Error metric
  Inputs:
    gt_img: cv2 image: groundtruth image
    smooth_img: cv2 image: smoothed image
  Outputs:
    mse_score: MSE score
  """
  try:
    gt_img = np.array(gt_img)
    smooth_img = np.array(smooth_img)
  except Exception:
    raise ValueError("Input must be 2D array like format")

  return np.mean(np.power((gt_img - smooth_img), 2))
```

Figure 4: MSE Metric

```python
def psnr(gt_img, smooth_img):
  """
    Calculate the PSNR metric
    Inputs:
      gt_img: cv2 image: groundtruth image
      smooth_img: cv2 image: smoothed image
    Outputs:
      psnr_score: PSNR score
  """
  try:
    gt_img = np.array(gt_img)
    smooth_img = np.array(smooth_img)
  except Exception:
    raise ValueError("Input must be 2D array like format")

  return 20 * np.log10(np.max(gt_img)/np.sqrt(mse(gt_img, smooth_img)))
```

Figure 5: PSNR Metric

1.3 Metrics

- Implementation of Mean Square Error metric 4
- Implementation of Peak Signal-to-Noise Ratio metric 5

After experimented using three metrics (MSE, PNSR, SSIM metric) to assess similarity of Noise Image 6 and Clean Image 7; Noise Image after using mean filter 8 and Clean Image; Noise Image after using median filter 9 and Clean Image, respectively, I have relized that SSIM metric is the most suitable metric to use for this problem.
Based on the naked eye, we can see that the Noise Image after use median filter is most similar to the Clean Image, then the Noise Image after use mean filter and finally the Noise Image, so we need a metric can describe that relationship . According to the result of experiment 10 we can see that only SSIM metric can express this relation among 3 cases.

1.4 Optimize parameters

Implementation of optimize parameter function 11
After apply this function to optimize parameters for median_filter function and mean_filter function I get results that best filter size and best padding mode for mean_filter function is 5 and "replicate" respectively; best filter size and best padding mode for median_filter function is 3 and "replicate" respectively.12
According to the result, median filter is better than mean filter with SSIM metric for this case.

2 Unsharp mask filter

2.1 Implementation

- Implementation of gaussian_filter 13
- Implementation of unsharp_filter 14

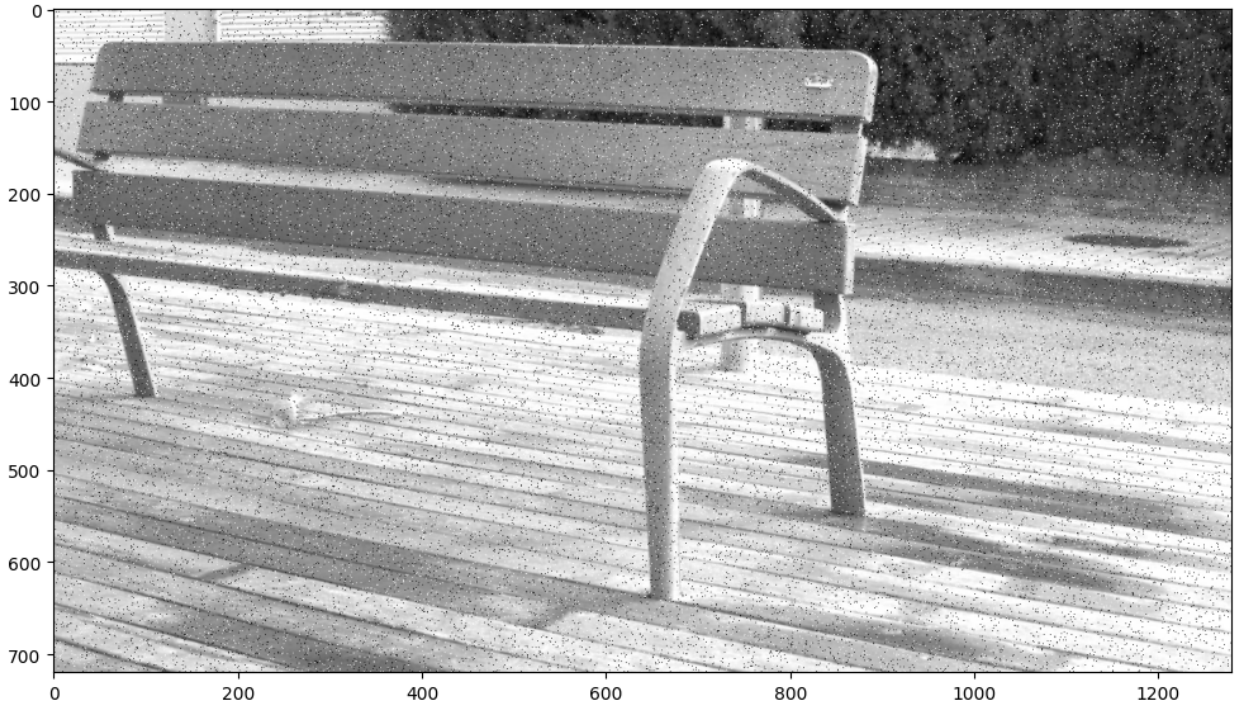


Figure 6: Noise Image

Figure 7: Clean Image

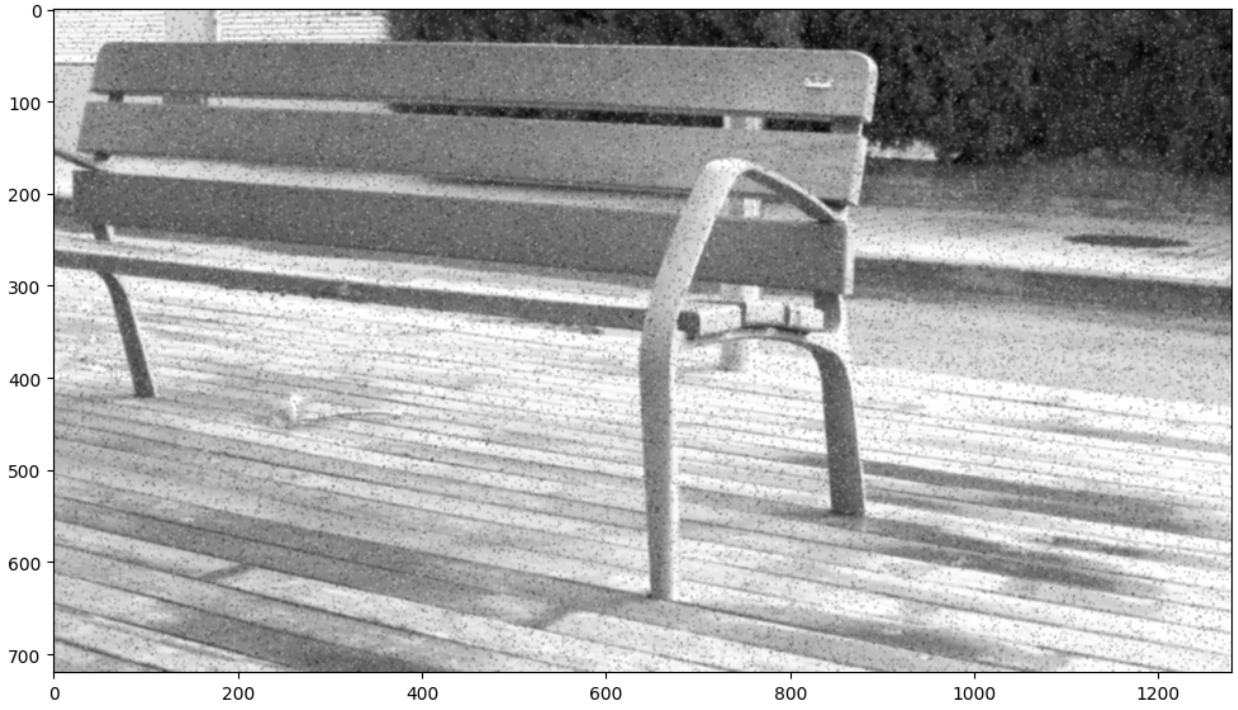


Figure 8: Mean Image

Figure 9: Median Image

```
print("PSNR between noise image and clean image:", psnr(clean_img, noise_img))
print("MSE between noise image and clean image:", mse(clean_img, noise_img))
print("SSIM between noise image and clean image:", compare_ssim(noise_img, clean_img, data_range=clean_img.max() - clean_img.min()))
print("--------------------------------------------------------------------------")
print("PSNR between mean filtered image and clean image:", psnr(clean_img, mean_blur))
print("MSE between mean filtered image and clean image:", mse(clean_img, mean_blur))
print("SSIM between mean filtered image and clean image:", compare_ssim(clean_img, mean_blur, data_range=clean_img.max() - clean_img.min()))
print("--------------------------------------------------------------------------")
print("PSNR between median filtered image and clean image:", psnr(clean_img, median_blur))
print("MSE between median filtered image and clean image:", mse(clean_img, median_blur))
print("SSIM between median filtered image and clean image:", compare_ssim(clean_img, median_blur, data_range=clean_img.max() - clean_img.min()))
```

```
PSNR between noise image and clean image: 41.137976602367075
MSE between noise image and clean image: 5.0036013454861115
SSIM between noise image and clean image: 0.362664911563029
--------------------------------------------------------------------------
PSNR between mean filtered image and clean image: 26.263183884698236
MSE between mean filtered image and clean image: 153.73118413226595
SSIM between mean filtered image and clean image: 0.6363219640842767
--------------------------------------------------------------------------
PSNR between median filtered image and clean image: 36.91360687919537
MSE between median filtered image and clean image: 13.234869791666666
SSIM between median filtered image and clean image: 0.9628636170478749
```

Figure 10: Experiment Result

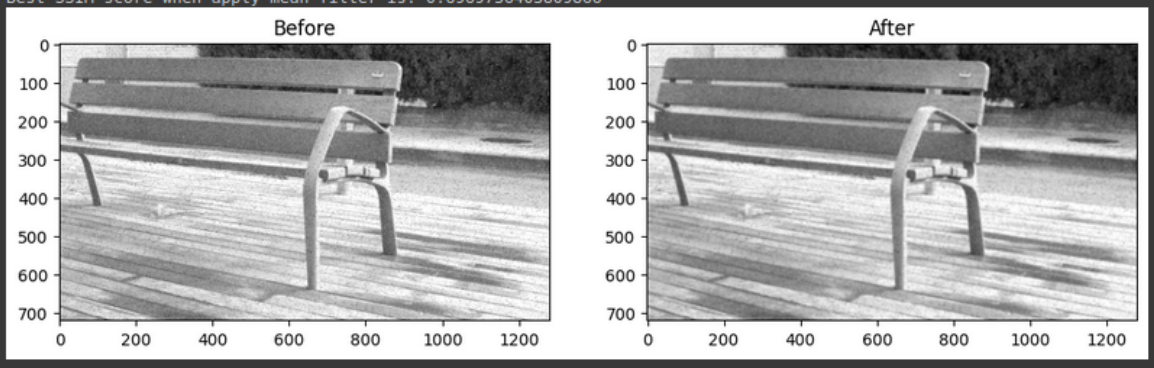
```
def optimize_params(noise_img, gt_img, smooth_fn):
  """
    Find the best value of parameters filter_size, padding_mode
    Inputs:
      noise_img: cv2 image: noise image
      gt_img: cv2 image: no noise image
      smooth_fn: the smooth function
    Outputs:
      best_filter_size
      best_padding_mode
  """
  try:
    noise_img = np.array(noise_img)
    gt_img = np.array(gt_img)
  except:
    print("Invalid input")
    return
  assert noise_img.shape[0] == gt_img.shape[0] and noise_img.shape[1] == gt_img.shape[1], "Noise image and Ground truth image must have same size"
  max_ssim = 0
  best_filter_size = 1
  best_padding_mode = "zero"
  for filter_size in range(1, 11, 2):
    for padding_mode in ['zero', 'mirror', 'replicate']:
      new_img = smooth_fn(noise_img, filter_size, padding_mode)
      score = compare_ssim(clean_img, new_img, data_range=clean_img.max() - clean_img.min())
      if max_ssim < score:
        max_ssim = score
        best_filter_size = filter_size
        best_padding_mode = padding_mode

  return best_filter_size, best_padding_mode
```

Figure 11: Optimize parameters function

```
fs, pm = optimize_params(noise_img, clean_img, mean_filter)
print("Best filter size:", fs, "\nBest padding mode:", pm)
score = compare_ssim(clean_img, mean_blur, data_range=clean_img.max() - clean_img.min())
print("Best SSIM score when apply mean filter is:", score)
mean_blur = mean_filter(noise_img, fs, pm)
show_res(noise_img, mean_blur)
```

```
Best filter size: 5
Best padding mode: replicate
Best SSIM score when apply mean filter is: 0.6969756403809866
```




```
[23] fs, pm = optimize_params(noise_img, clean_img, median_filter)
print("Best filter size:", fs, "\nBest padding mode:", pm)
score = compare_ssim(clean_img, median_blur, data_range=clean_img.max() - clean_img.min())
print("Best SSIM score when apply median filter is:", score)
median_blur = median_filter(noise_img, fs, pm)
show_res(noise_img, median_blur)
```

```
Best filter size: 3
Best padding mode: replicate
Best SSIM score when apply median filter is: 0.9629107353815392
```

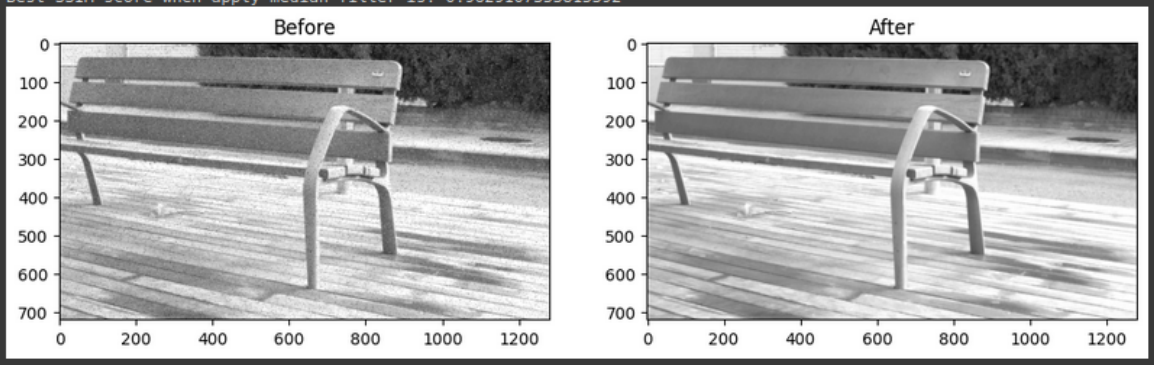
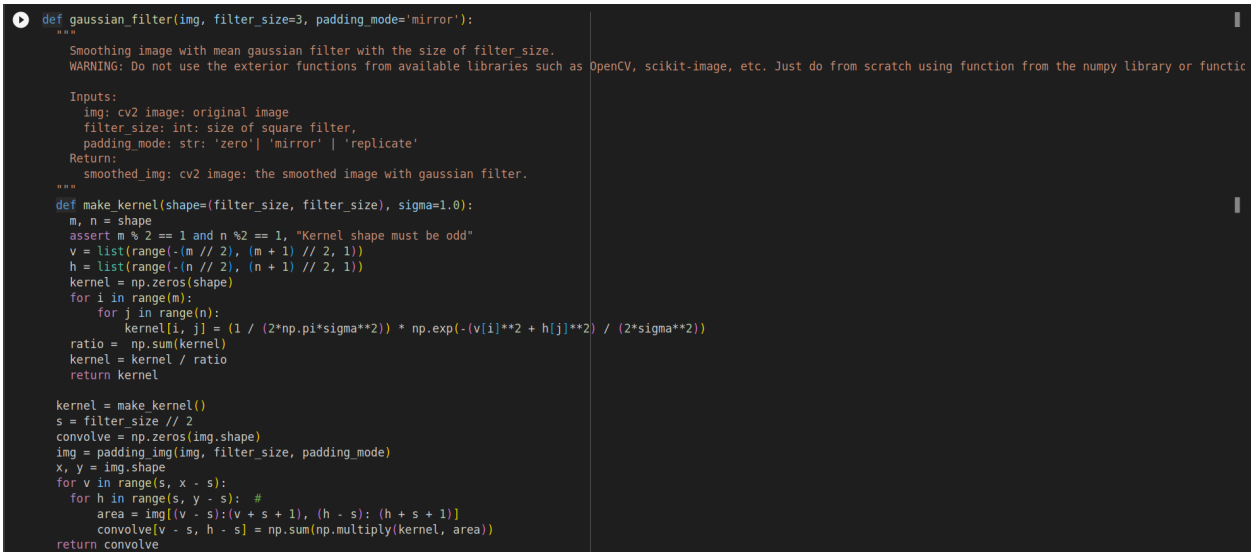


Figure 12: Optimize parameters result



```python
def gaussian_filter(img, filter_size=3, padding_mode='mirror'):
    """
      Smoothing image with mean gaussian filter with the size of filter_size.
      WARNING: Do not use the exterior functions from available libraries such as OpenCV, scikit-image, etc. Just do from scratch using function from the numpy library or functic

      Inputs:
        img: cv2 image: original image
        filter_size: int: size of square filter,
        padding_mode: str: 'zero'| 'mirror' | 'replicate'
      Return:
        smoothed_img: cv2 image: the smoothed image with gaussian filter.
    """
    def make_kernel(shape=(filter_size, filter_size), sigma=1.0):
      m, n = shape
      assert m % 2 == 1 and n %2 == 1, "Kernel shape must be odd"
      v = list(range(-(m // 2), (m + 1) // 2, 1))
      h = list(range(-(n // 2), (n + 1) // 2, 1))
      kernel = np.zeros(shape)
      for i in range(m):
          for j in range(n):
              kernel[i, j] = (1 / (2*np.pi*sigma**2)) * np.exp(-(v[i]**2 + h[j]**2) / (2*sigma**2))
      ratio =  np.sum(kernel)
      kernel = kernel / ratio
      return kernel

    kernel = make_kernel()
    s = filter_size // 2
    convolve = np.zeros(img.shape)
    img = padding_img(img, filter_size, padding_mode)
    x, y = img.shape
    for v in range(s, x - s):
      for h in range(s, y - s):  #
          area = img[(v - s):(v + s + 1), (h - s): (h + s + 1)]
          convolve[v - s, h - s] = np.sum(np.multiply(kernel, area))
    return convolve
```

Figure 13: Gaussian Filter Function

```python
def unsharp_filter(img, alpha, filter_type="gaussian", padding_mode="mirror", filter_size=3):
    """
      Unsharp mask filter
      Inputs:
        img: cv2 image: original image
        alpha: float: [0, 1]
        filter_type: str: 'gaussian'|'mean'|'median'
        filter_size: int: size of square filter,
        padding_mode: str: 'zero'| 'mirror' | 'replicate'
      Return:
        unsharped_img: cv2 image: the smoothed image with gaussian filter.
    """
    filter_func = gaussian_filter if filter_type=="gaussian" else median_filter if filter_type=="median" else mean_filter
    unsharped_img = img + alpha * (img - filter_func(img, filter_size, padding_mode))
    return unsharped_img
```

Figure 14: Unsharp Filter Function

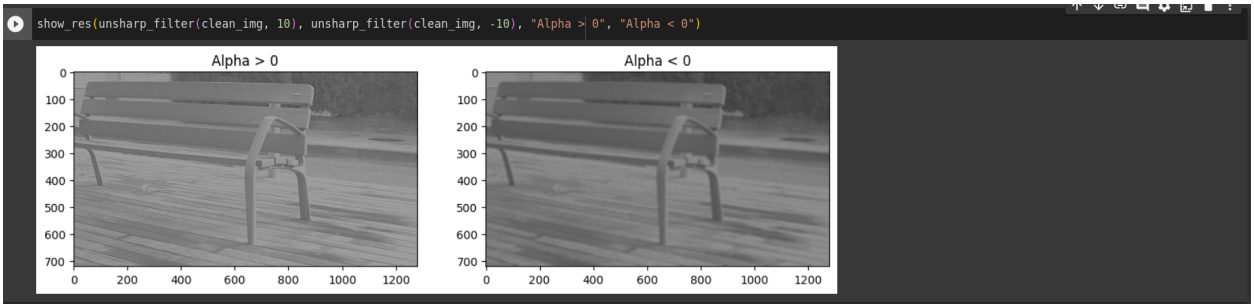


Figure 15: Illustration alpha parameter effect to unsharp filter

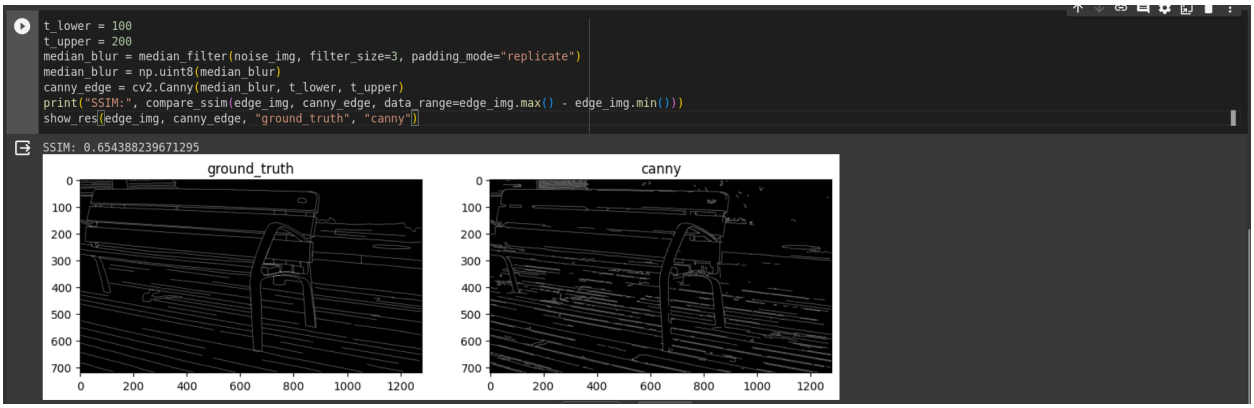


Figure 16: Default canny edge detection result

2.2 Explaination

Formula:

$$f\_new = f\_old + \alpha(f\_old - f\_old * g)$$

α is a sharpen parameter that if α is a positive number, the new image will be a sharpen image, if α is greater, new image will tend to sharpen. And vice versa, if α is a negative number, the new image will dimmer than original image. 15

3 Edge detection

3.1 Canny Edge detection

1. Firstly, I load noise image and edge image from google drive.
2. Use median filter with optimized parameters that have found in previous section to filter noise image and get median_blur image.
3. Apply canny edge detection function for median_blur image with lower threshold value is 100 and upper threshold value is 200, other parameters are default parameters and I have result is canny_edge image.
4. Use compare_ssim function to measure SSIM score between canny_edge image and edge image that have loaded before. Result is SSIM score ≈0.65. 16
5. I fix lower threshold value is 180 and adjust upper threshold value from 200 to 255 and step 5 to

```
median_blur = median_filter(noise_img, filter_size=3, padding_mode="replicate")
median_blur = np.uint8(median_blur)
for upper in range(200, 256, 5):
  canny_edge = cv2.Canny(median_blur, 180, upper)
  print(f"SSIM if upper threshold is {upper}:", compare_ssim(edge_img, canny_edge, data_range=edge_img.max() - edge_img.min()))
```

```
SSIM if upper threshold is 200: 0.6844251790696576
SSIM if upper threshold is 205: 0.6868679895697899
SSIM if upper threshold is 210: 0.6901674380286822
SSIM if upper threshold is 215: 0.6919117211416093
SSIM if upper threshold is 220: 0.6940049104000199
SSIM if upper threshold is 225: 0.6949816002275071
SSIM if upper threshold is 230: 0.6970390003464227
SSIM if upper threshold is 235: 0.6977536369654955
SSIM if upper threshold is 240: 0.6991816556549182
SSIM if upper threshold is 245: 0.7002639450130395
SSIM if upper threshold is 250: 0.7019324689508859
SSIM if upper threshold is 255: 0.7025377539147916
```

Figure 17: Adjust upper threshold parameters

observe SSIM score between created image and ground truth image for each case. According to result 17, if upper threshold value increase, the SSIM score tend to increase.

3.2 Threshold parameters

- High threshold: This is the threshold value for detecting strong edges. Pixels with a gradient magnitude above the high threshold are considered to be strong edges.
- Low threshold: This is the threshold value for detecting weak edges. Pixels with a gradient magnitude between the low and high thresholds are considered to be weak edges.
- Weak edges are only considered to be valid edges if they are connected to strong edges. This helps to suppress noise and produce a more accurate edge map.

3.3 Hough Lines Transform

1. I will use canny edge detection with parameters like above to have canny_edge image
2. Calculate lines in canny_edge image using HoughLines function in OpenCV package
3. I adjust threshold parameter of HoughLines function and see that if threshold is too small like 100, many edges are detected but is not real edge of image, if threshold to to high like 150, many edges in image are not detected, so I choose 120 to balance although leaving out some edges of image.
4. Draw lines in median_blur image (Image created by using median filter for noise image) 18

3.4 Optimization

In canny edge detection function, I will use gradient ascent that maximize SSIM score between canny_edge image and ground truth image to optimize low threshold value and high threshold value. 19
With this strategy I get optimized low threshold value is 162 and high threshold value is 283. 20

```python
median_blur_copy = median_blur.copy()
lines = cv2.HoughLines(canny_edge, 1, np.pi / 180, 120, None, 0, 0)
if lines is not None:
  for i in range(0, len(lines)):
      rho = lines[i][0][0]
      theta = lines[i][0][1]
      a = math.cos(theta)
      b = math.sin(theta)
      x0 = a * rho
      y0 = b * rho
      pt1 = (int(x0 + 2000*(-b)), int(y0 + 2000*(a)))
      pt2 = (int(x0 - 2000*(-b)), int(y0 - 2000*(a)))
      cv2.line(median_blur_copy, pt1, pt2, (0,0,255), 3, cv2.LINE_AA)
show_img(median_blur_copy)
```

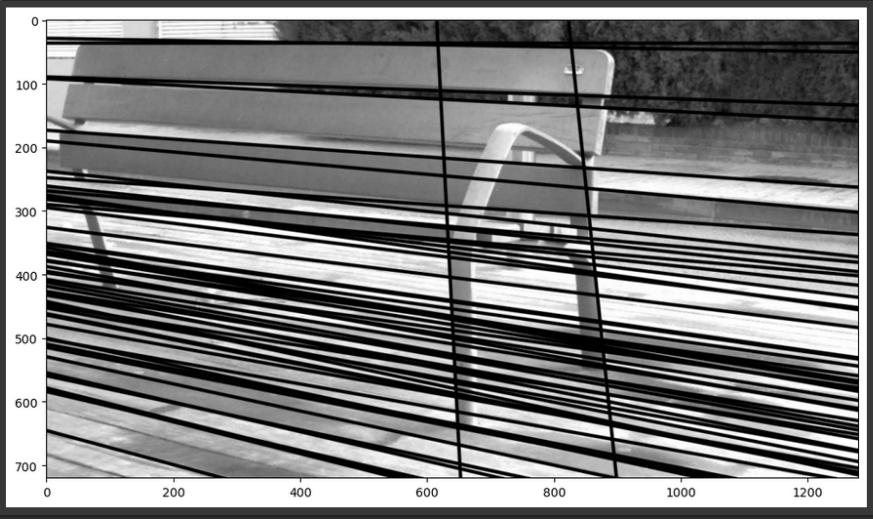


Figure 18: Hough Lines Transform

```python
median_blur = median_filter(noise_img, filter_size=3, padding_mode="replicate")
median_blur = np.uint8(median_blur)
lower = 100
upper = 200
delta = 5
alpha = 10000
while True:
  print(lower, upper)
  old_lower = lower
  old_upper = upper
  old_score = compare_ssim(edge_img, cv2.Canny(median_blur, lower, upper), data_range=edge_img.max() - edge_img.min())

  new_score_lower = compare_ssim(edge_img, cv2.Canny(median_blur, lower + delta, upper), data_range=edge_img.max() - edge_img.min())
  d_lower = (new_score_lower - old_score) / delta
  lower = lower + int(alpha * d_lower)

  new_score_upper = compare_ssim(edge_img, cv2.Canny(median_blur, lower, upper + delta), data_range=edge_img.max() - edge_img.min())
  d_upper = (new_score_upper - old_score) / delta
  upper = upper + int(alpha * d_upper)
  if old_lower == lower and old_upper == upper:
    print("Result", lower, upper)
    break
```

Figure 19: Gradient ascent

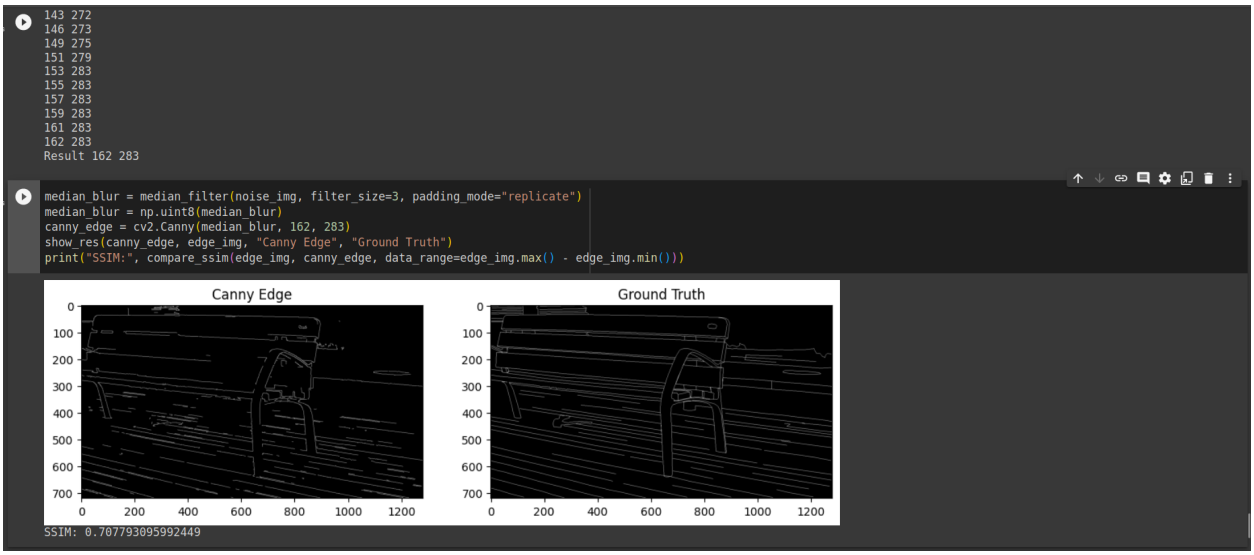


Figure 20: Optimized parameters