Computer Visual Homework 2

ID:113598045 Name:林翡

- Explain your code:

1. **Calculate PSNR (calculatePSNR、calculateMSE Functions):**

In the calculateMSE function, the goal is to calculate the MSE value of the original grayscale image and the filtered image. First, check whether the two images have the same dimensions[Lines 9-10] , and then obtain the height and width of the original grayscale image[Line 11]. Use nested loops to iterate over each pixel of the image, for each pixel (i, j), the function calculates the difference between the pixel values of img1 and img2. The pixel values are converted to floats to ensure that the calculation can handle potential decimal values accurately , and then squares the difference and adds it to squared_diff_sum , and the squared difference is accumulated for all pixels[Lines 14-17]. Calculates the total number of pixels in the image by multiplying the height by the width, and the MSE is calculated by dividing the total squared difference (squared_diff_sum) by the total number of pixels[Lines 18-21].

In the calculatePSNR function, the goal is to calculate the PSNR value of the original grayscale image and the filtered image. First call the calculateMSE function to calculate the MSE values of the two images[Line 26]. Then checks if the MSE is zero. If the MSE is zero, it indicates that the two images are identical, resulting in no error. In this case, the function returns infinity (float('inf')), representing the best possible PSNR value, as the ratio would be undefined (infinite quality)[Lines 27-28]. Calculate based on the PSNR formula, rounding the result to two decimal places[Lines 30-32].

2. **Median Filter (quicksort、padding、Median_filter Functions):**

In the quicksort function, use the quick sort algorithm to sort the pixel value list from small to large. First checks if the length of the input list (arr) is less than or equal to 1. If it is, the list is already sorted, so the function returns it as is[Lines 36-37]. Then select a pivot, representing the middle of the list[Line 38], put the numbers smaller than the pivot in the left list[Line 39], and put the numbers larger than the pivot in the right list[Line 41], and the numbers equal to the pivot in the middle list[Line 40].

In the padding function, the goal is to add padding around a given image. First, obtain the height and width of the input image. Then, create a

new image with padding, and copy the original image into the center of the padded image[Lines 46-50]. Add padding to the top, bottom, left, right, and corners of the image. The top edge of the padded image is filled with the first row of the original image, and the bottom edge is filled with the last row of the original image[Lines 53-54]. The left edge of the padded image is filled with the first column of the original image, and the right edge is filled with the last column of the original image[Lines 57-58]. Each of the four corners of the padded image is filled with the corresponding corner pixel from the original image[Lines 61-64].

In the Median_filter function, the goal is to implement a median filter. First, the height and width of the input image are obtained, and the padding size is calculated. The padding function is called to pad the image, and an empty array is created to store the filtered result [Lines 68-74]. Next, nested loops are used to iterate over each pixel in the image, considering the specified stride. For each pixel (i, j), a neighborhood (or window) of pixel values is extracted from the padded image[Lines 76-79]. This window is flattened and sorted using the quicksort function[Line 81]. The median value is determined as the middle element of the sorted list, and this median value is then assigned to the corresponding position in the filtered image[Lines 83-84].

3. **Gaussian Filter (Gaussian_kernel、Gaussian_filter Functions):**

In the Gaussian_kernel function, the goal is to generate a 2D Gaussian kernel. First, the center index of the kernel is calculated, and an empty kernel array is created [Lines 89-91]. For each pixel (i, j), the distances x = i - center and y = j - center from the center are calculated, and the Gaussian kernel equation is used to compute the corresponding value [Lines 94-100]. Finally, the kernel is normalized by dividing each value by the sum of all values, ensuring that the total sum equals 1 [Line 102].

In the Gaussian_filter function, uses a Gaussian filter to blur or smooth an image using a convolution with a Gaussian kernel. Same as Median_filter function, first the height and width of the input image are obtained, and the padding size is calculated. The padding function is called to pad the image, and an empty array is created to store the filtered result. The Gaussian_kernel function is called to generate a 2D Gaussian kernel of the specified size and with a fixed standard deviation of 1[Lines 106-113]. Uses nested loops to iterate over each pixel (i, j) in the image. For each pixel, a neighborhood (or window) of pixel values of the same size as the Gaussian kernel is extracted from the padded image. The pixel values in the

window are multiplied element-wise by the corresponding values in the Gaussian kernel. The sum of these products is then computed, resulting in the final value for the filtered image at position (i, j). The computed value is stored in the filtered_image at the corresponding position[Lines 115-120].

4. **Combine (combine_median_and_gaussian Function):**

   In the combine_median_and_gaussian function, the goal is to apply a two-step filtering process to a noisy image, using both a median filter and a Gaussian filter[Lines 125-130].

5. **Image Histogram (count_pixel_intensity、plot_histogram Functions):**

   In the count_pixel_intensity function, the goal is to count occurrences of each pixel value(0~255). The function uses a loop that iterates over each possible pixel value (from 0 to 255). For each pixel value i, it calculates how many pixels in the image are equal to i using np.sum(img == i), and stores that count in the corresponding index of pixel_counts[Lines 135-136].

   In the plot_histogram function, it visualizes the distribution of pixel intensities in the input image by plotting a histogram and also printing a summary table of the pixel counts. First, the count_pixel_intensity function is called to obtain the counts of each pixel intensity, and pixel_values is initialized as an array of pixel values from 0 to 255 [Lines 142-143]. Then it calculates the total number of pixels in the image and checks if the sum of the pixel_counts matches this total [Lines 146-147]. A pandas DataFrame is created to organize the pixel values and their corresponding counts, which is then printed along with the specified table_title [Lines 149-153]. Finally, it uses Matplotlib to create a bar chart histogram that displays the pixel value distribution [Lines 155-162].

6. **Main function:**

   First read noisy_image.png and gray_image.png, then call the Median_filter function, set the kernel size to 3, the stride size to 1, and the padding size to half of the kernel[Lines 166-169]. Then save the result of median_filter_img as output_q1.png. Call the calculatePSNR function to calculate the PSNR value after median filtering[Lines 171-174].

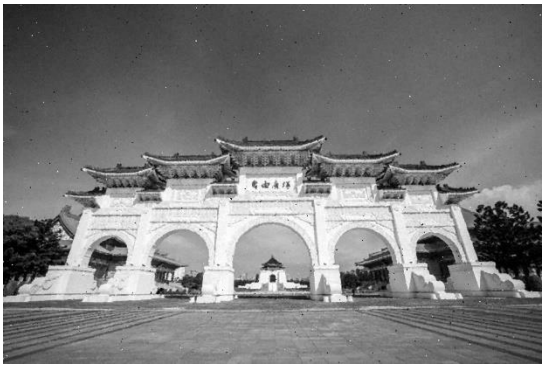
   Call the Gaussian_filter function to implement Gaussian filtering, set the kernel size to 5, the stride size to 1, and the padding size to half the kernel size. Save the result as output_q2.png, and calculate the PSNR value with gray_image.png[Lines 176-181].

   Call the combine_median_and_gaussian function to implement the combined filter. The parameter median kernel size is set to 3, the Gaussian kernel size is also set to 3, the padding size is also half of the kernel size,

and the PSNR value is calculated[Lines 184-187].

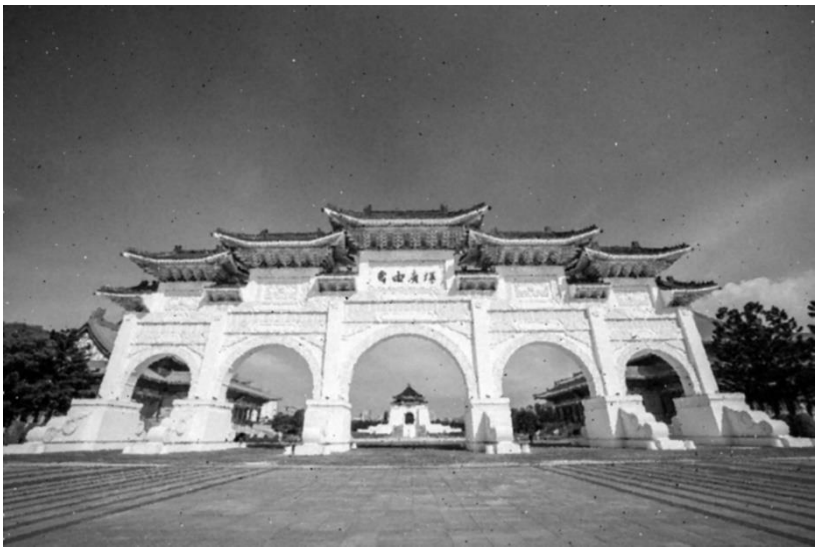
Draw the four images noisy_image.png, output_q1.png, output_q2.png, output_q3.png accumulate the number of each pixel value[Lines 190-193].

- Put the 3 result images that were generated by 3 different filters and their corresponding PSNR values, compare and describe what you observe.



output_q1.png

output_q2.png

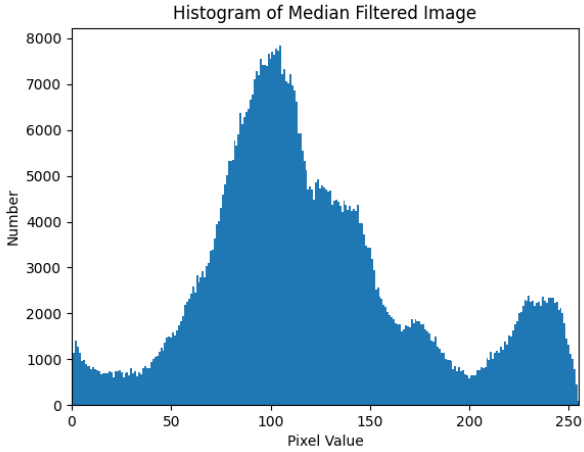


output_q3.png

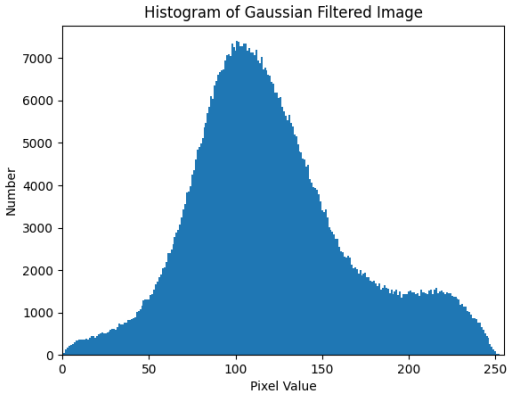
1. Compared with the original image, the PSNR of median filtering is 26.16 dB, while the PSNR after Gaussian filtering is 21.11 dB. The PSNR of combining median and Gaussian filtering is 26.4 dB (both to the second decimal place).
2. From the analysis of the above PSNR results, the image after median filtering can reduce the noise of the image more effectively than the image after Gaussian filtering, because the PSNR value is higher than that after Gaussian filtering, it can be seen that the median filtering is good at processing salt and pepper noise, while the PSNR value of combining the median and Gaussian filters is the highest, which shows that the

combination of the two filters achieves a good balance between noise reduction and detail preservation.

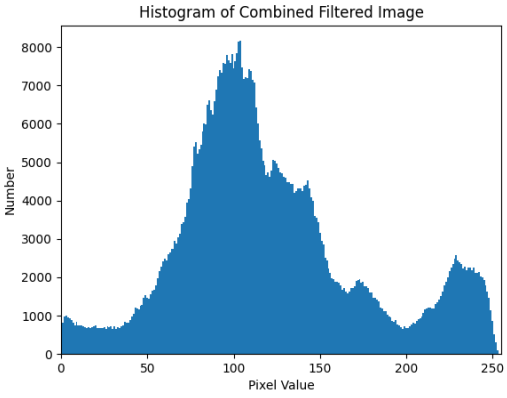
- Put the 3 histograms results, analyze and describe the difference between them, and explain the reason.



output_q1_his.png



output_q2_his.png



output_q3_his.png

- In the histogram of the median-filtered image, two distinct peaks can be observed, located at pixel values around 100 and between 200-255. I assume this is because the median filter is effective at removing salt-and-pepper noise (the noise present in the input image), while preserving the contrast between high-intensity (bright areas) and low-intensity (dark areas). It efficiently removes noise in the mid-range intensities, allowing the edges and contrast in the image to be well-preserved.

  In contrast, the histogram of the Gaussian-filtered image shows a single peak, primarily concentrated around a pixel value of 100 in the mid-range. This is

because the Gaussian filter mainly smooths the image, blurring extreme bright and dark areas, which results in a more uniform distribution of pixel intensities and reduces the number of high-contrast regions.

As for the combined filter, its histogram is similar to that of the median filter, showing a dual-peak distribution. This is likely because the combined filter retains the noise removal characteristics of the median filter while also applying some smoothing. However, the effect on image contrast is not as pronounced as with the Gaussian filter alone, which is why the distribution resembles that of the median filter.