Computer Visual Homework 1

ID:113598045 Name:林翡

- Explain your program and method:
  1. **Convert to grayscale image(convertGray Function):**

     First get the height, width, and channel of the original image[Line 8], create a new blank grayscale image with a shape of (height, width), and initialize it with zero[Line 9]. The data type of this grayscale image is uint8 (unsigned 8-bit integer), with each pixel having a value ranging from 0 to 255. Use a nested for loop to iterate the rows and columns of each pixel, obtain the channel values of the three channels (RGB), and then calculate the gray value according to the formula: (0.3 * R) + (0.59 * G) + (0.11 * B),and converted to an integer[Lines 11-18]. Finally, the calculated grayscale value is stored in the new grayscale image[Line 19].
  2. **Convolution operation with Laplacian edge detection kernel using zero padding and stride** 1**(ZeroPadding、edge_detection、ConvolutionOperation Functions):**

     In the ZeroPadding function, the goal is to apply zero padding to the input image. The input parameter is an nparray, which is a 2D array representing a grayscale image (i.e., the grayscale image obtained from the first task). First, create a new array filled with zeros, where the size is 2 rows and 2 columns larger than the original array[Line 32]. This new array will save the padded image. Use a nested for loop to iterate through each pixel of the original array and copy it to the new array at the position (i+1, j+1), leaving a border of zeros around the image[Lines 33-36]. Finally, use Matplotlib to display the padded image, setting the color mapping to grayscale[Lines 37-39].

     In the edge_detection function, the goal is to use Laplacian convolution kernel for edge detection. It uses a specific 3x3 convolution kernel [Lines 42-44] to detect the edges in the image and generate a new image. First, an empty pixel list is created to save the pixel values from the specified region [Line 47]. A nested loop, for i in range(row, row + 3) and for j in range(col, col + 3), is used to extract all the pixels within a 3x3 region of the 2D image matrix img, starting from row, col, and these pixels are added to the pixels list [Lines 48-50]. Using zip(pixels, kernel), the extracted pixel values are multiplied element-wise with the corresponding positions in the convolution kernel, and the results are summed using sum() to get the

result of the convolution operation for this 3x3 region [Line 52].

In the ConvolutionOperation function, the goal is to perform convolution on the entire image. First, the grayscale image's data type is converted to int32 to avoid data overflow during the subsequent operations (where pixel values may exceed 255 or be less than 0) [Line 57]. Zero padding is applied to the grayscale image to enable convolution at the image boundaries [Line 59]. After padding, the width and height of the padded image are obtained [Line 61]. An empty list is created, which will eventually save the image data after edge detection [Line 63]. A nested for loop is used to iterate over the padded image (excluding the boundaries), calling the edge_detection function to perform convolution, and the result is saved in the image_row list. After completing the convolution for each row, the row is added to the edge_detect_image list [Line 67-72]. Then, the edge_detect_image list is converted into a NumPy array [Line 75]. Using np.clip(arr, 0, 255), the convolution results are clipped to the valid range of 0 to 255, and the array is converted to the uint8 type to conform to standard image format [Line 78].

Next, the ConvolutionOperation function is called to perform convolution on the grayscale image, generating the edge-detected image edge_detected_img[Line 81]. Since edge_detected_img is a NumPy array, it needs to be converted into an image [Line 83].

3. **Pooling operation with each of Average pooling and Max pooling.Using the SAME 3x3 kernel and stride 2(Pooling Function):**

In the Pooling function, the parameter pool_size=3 represents a 3x3 kernel size, and stride=2 indicates that the kernel moves by 2 pixels at a time. First, the height and width of the image are obtained, ignoring the channels [Line 89]. Next, the dimensions of the pooled image are calculated [Lines 92-93]. The formula (height - pool_size) // stride + 1 ensures that the calculated dimensions fit the image boundaries and prevent the pooling region from exceeding the image range. Two empty NumPy arrays, max_pool_img and average_pool_img, are created to save the results of max pooling and average pooling, respectively [Lines 96-97]. A nested for loop is used to iterate over each pixel position of the pooled image [Lines 100-101]. The image is sliced to extract the pooling region, which is always a 3x3 area (as determined by pool_size), and the region's position shifts based on the loop indices i and j, moving by the size of the stride [Line 103]. The np.max(region) and np.mean(region) functions are used to compute the maximum and average values of the region, and these

values are assigned to max_pool_img[i, j] and average_pool_img[i, j], representing the value at position (i, j) in the pooled image [Lines 105-107].

4. **Binarization operation for each of the result of pooling operation(Binarization Function):**

   In the Binarization function, the goal is to perform binarization on two images, which is a process that converts a grayscale image into only two colors (black and white). The parameter img is the input image, and threshold is the binarization threshold used to determine whether a pixel should be black or white. First, the input image is converted to a grayscale image (by calling the convertGray function written in the first question) [Line 115]. The height and width of the input image are obtained [Line 116], and a blank binarized image, binar_img, of the same size as the input image is created [Line 117]. A nested for loop is used to iterate over each pixel, and for each pixel, the following condition is checked: if the pixel's value is less than the set threshold, the pixel's value in binar_img is set to 0 (black). Otherwise, the pixel's value in binar_img is set to 255 (white) [Lines 118-123]. This converts pixels in the original image that are below the threshold to black and those that are above or equal to the threshold to white, completing the binarization process.

   Regarding the threshold settings, I set the threshold to 50 for the Average Pooling image and Max Pooling image[Lines 129-130], as this yields more noticeable results.

- Put the result images (1 input images, 6 outputs images):

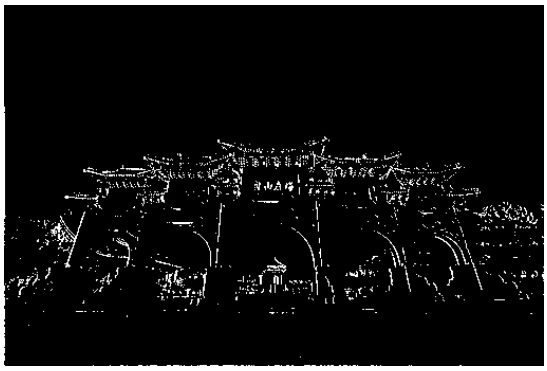
Input image

CKS _Q1.jpg

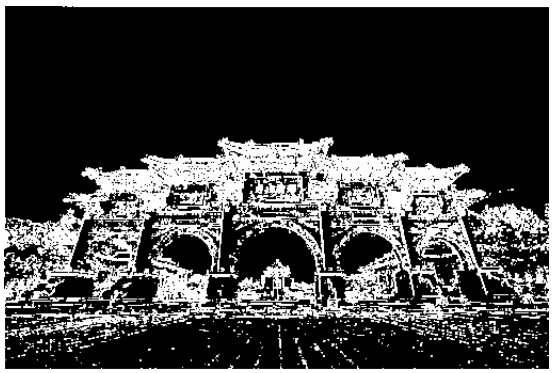
CKS _Q2.jpg

CKS _Q3a.jpg

CKS _Q3b.jpg



CKS _Q4a.jpg



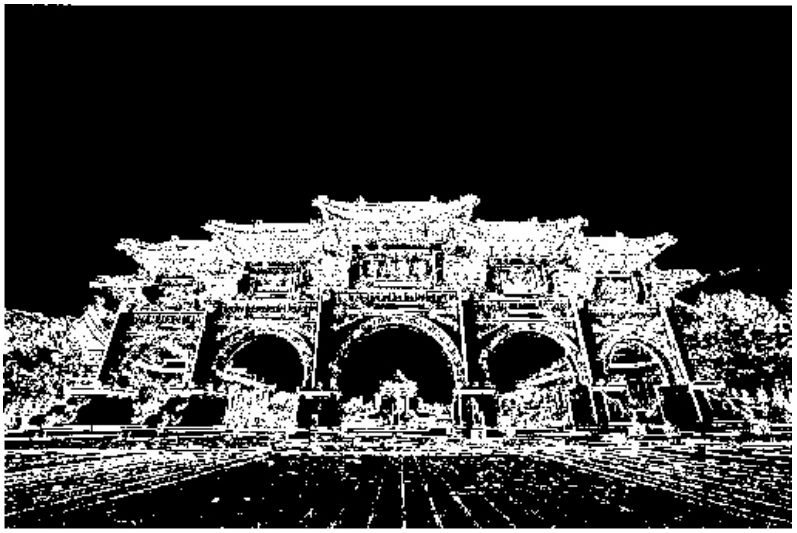
CKS _Q4b.jpg

- Compare and discuss the difference between the output produced by mean pooling + binarization (Output4a) and max pooling + binarization (Output4b):

  From the generated images, it can be observed that the result after average pooling appears darker than the image after max pooling. Therefore, when both thresholds are set to 50, the white areas depicted in the image resulting from mean pooling and binarization are significantly fewer compared to those from max pooling and binarization.

  Additionally, regarding the threshold settings, when I set the threshold to 128,

the image after average pooling resulted in a completely black image, and the effect after max pooling was also not very satisfactory. This is why I lowered the threshold to 50. However, I found that using a threshold of 50 for mean pooling and binarization was still not ideal, so I later adjusted it to 15, which made the results more comparable to those from max pooling and binarization(As shown in the figure below)



The result of adjusting the mean pooling threshold to 15