Computer Vision HW03 旁聽生 陳又蕾

Brief Description: This homework used Python, OpenCV, numpy to transform the iconic Lena image to

(a) an original image and its histogram

(b) an image with intensity divided by 3 and its histogram

(c) an image after applying histogram equalization to the reduced-intensity image (b) and its histogram

Please use python ./main.py to run the program.

NOTE: The input image is first binarized before passing into the functions.

The kernel used for the first four homework problems is a 3-5-5-5-3 octagonal. The tuples indicate the directions/change in rows and columns of all white points neighboring the origin pixel:

```
kernel = [(-2, -1), (-2, 0), (-2, 1), (-1, -2),(-1, -1),(-1, 0),(-1, 1),
(-1, 2), (0, -2), (0, -1), (0, 0), (0, 1), (0, 2), (1, -2), (1, -1), (1, 0),
(1, 1), (1, 2), (2, -1), (2, 0), (2, 1)]
```

The same idea is used for the two L-shaped kernels in the hit-and-miss pronlem:

```
kernel_j = [(0, 0), (0, -1), (1, 0)]
kernel_k = [(-1, 1), (0, 1), (-1, 0)]
```

Dilation

```
def dilation(img, kernel):
    res = np.zeros((img.shape[0], img.shape[1]), np.int)
    for row in range(img.shape[0]):
        for col in range(img.shape[1]):
            if(img[row][col] == 255):
                for (px, py) in kernel:
                    new_row = row +  px
                    new_col = col + py
                    if(new_row >= 0 and new_row < img.shape[0] and new_col >= 0
                        and new_col < img.shape[1]):
                        res[new_row][new_col] = 255
```

```
    return res
```

I first created an array of black pixels with the size of the input image. I later loop through the input image to check if it is a white point. If it is, the kernel will be looped through to generate the rows and columns of all the neighboring pixels that is supposed to be transformed to a white point. If the row and column of that neighboring pixel does not exceed the range of the image size, the value of its respective pixel in the array will be set to white. The resulting array of pixels is at last returned from this function.

Erosion

```
def erosion(img, kernel):
    res = np.zeros((img.shape[0], img.shape[1]), np.int)
    is_eroded = True
    for row in range(img.shape[0]):
        for col in range(img.shape[1]):
            for (px, py) in kernel:
                new_row = row +  px
                new_col = col + py
                if(new_row < 0 or new_row >= img.shape[0] or new_col < 0
                    or new_col >= img.shape[1] or
                    img[new_row][new_col] != 255):
                    is_eroded = False
            if(is_eroded):
                res[row][col] = 255
            is_eroded = True
    return res
```

I first created an array of black pixels with the size of the input image. For each pixel of the input image, I loop through the kernel to examine whether each points of the kernel can be found around the pixel and is a white point. If the condition is met, the value of its respective pixel in the array will be set to white. The resulting array of pixels is at last returned from this function.

Opening

```
def opening(img, kernel):
    return dilation(erosion(img, kernel), kernel
```

The input image is first eroded then dilated using the dilation and erosion I wrote previously.

Closing

```
def closing(img, kernel):
    return erosion(dilation(img, kernel), kernel)
```

The input image is first dilated then eroded using the dilation and erosion I wrote previously.

Hit-and-miss

```
def complement(img):
    res = np.zeros((img.shape[0], img.shape[1]), np.int)
    for row in range(img.shape[0]):
        for col in range(img.shape[1]):
            if(img[row][col] == 0):
                res[row][col] = 255
            else:
                res[row][col] = 0
    return res


def hit_and_miss(img, kernel_j, kernel_k):
    complement_img = complement(img)
    img1 = erosion(img, kernel_j)
    img2 = erosion(complement_img, kernel_k)
    res = np.zeros((img.shape[0], img.shape[1]), np.int)
    for row in range(img.shape[0]):
        for col in range(img.shape[1]):
            if(img1[row][col] == 255 and img2[row][col] == 255):
                res[row][col] = 255
    return res
```

I first created a complement of the input image by transforming all white pixels to black and vice versa. I then erode the input image and complement image with its respective kernel given from the problem. Lastly, for each pixel of the image, I check to see whether both eroded images have a white point on that specific pixel. If so, the value of that respective pixel in an array of black pixels will be set to white. The resulting array of pixels is at last returned from this function.