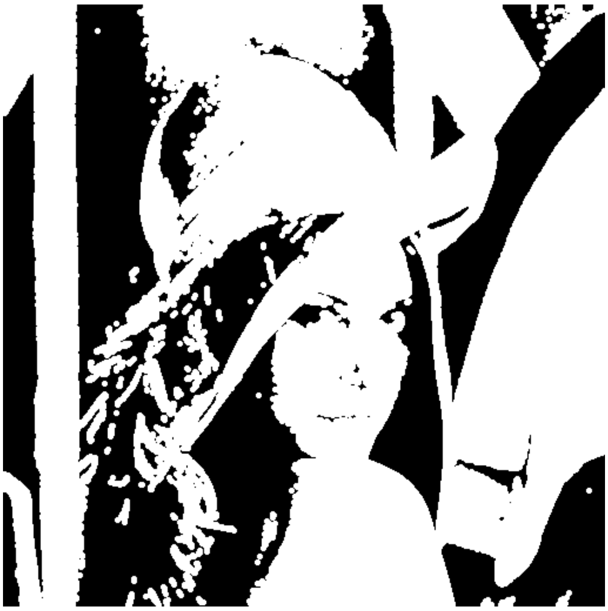
Computer Vision Homework #4

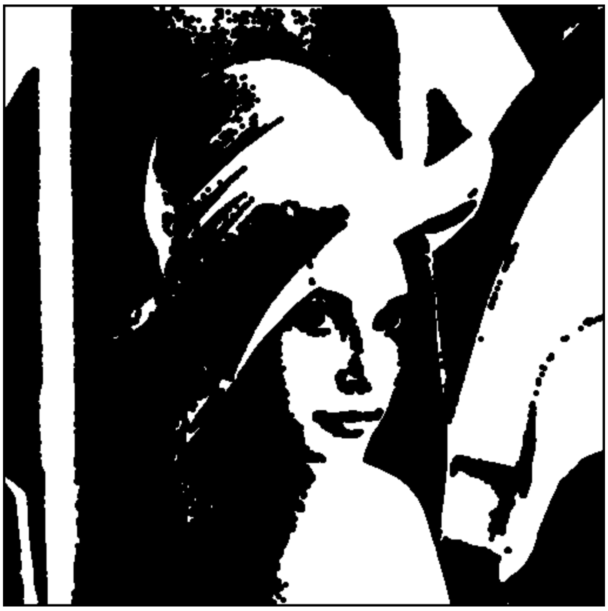
Student-ID: r10944020; Name: 林顥倫; Department: GINM

[Results]

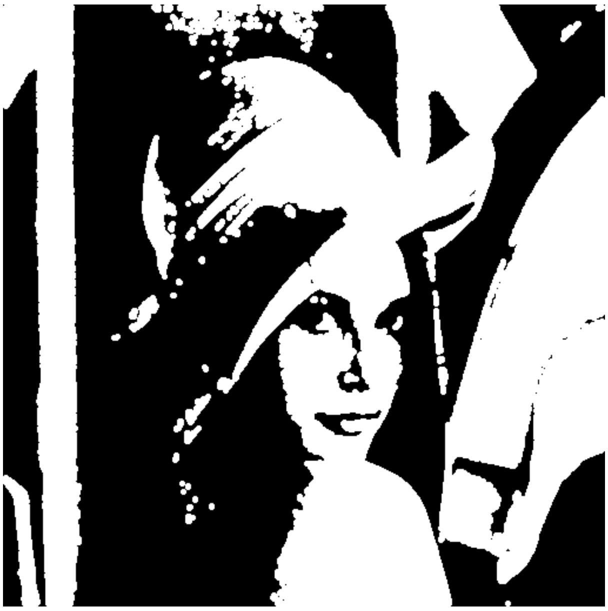
(a) Dilation



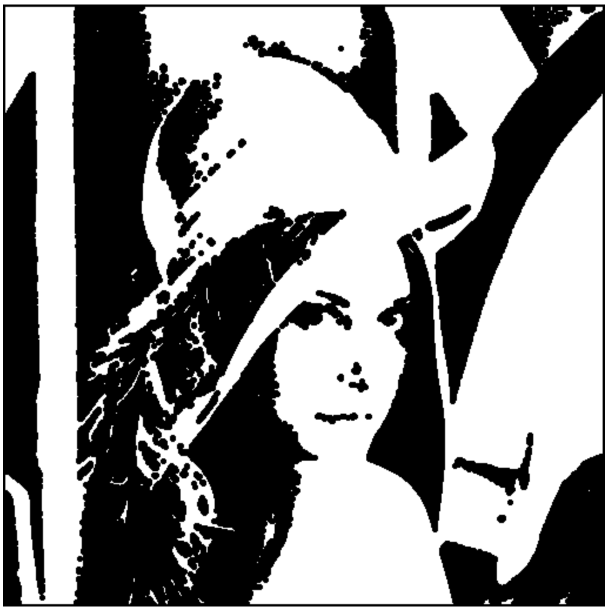
(b) Erosion



(c) Opening



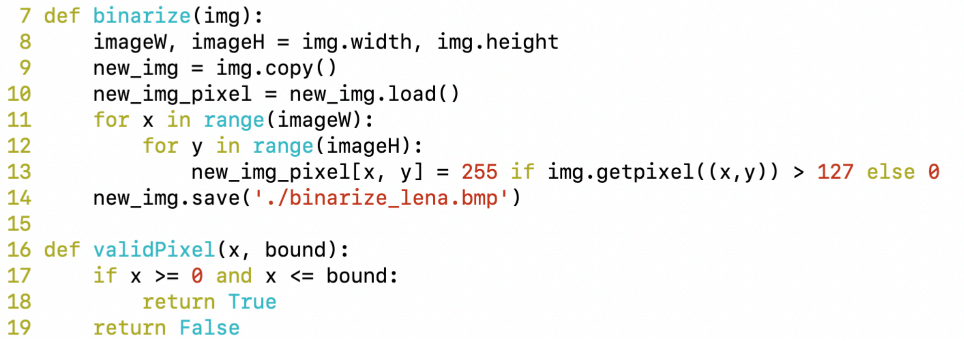
(d) Closing



(e) Hit-and-miss transform

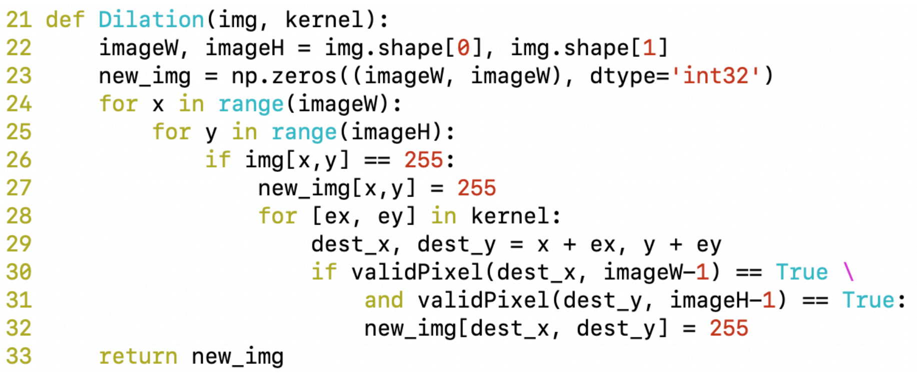
[Code Fragment & Explanation]

First of all, I would like to introduce 2 auxiliary functions I used in this assignment. Function “binarize” helps us to binarize original source image. Function “validPixel” helps us to check if the current (x,y) pair is in the valid pixel region.



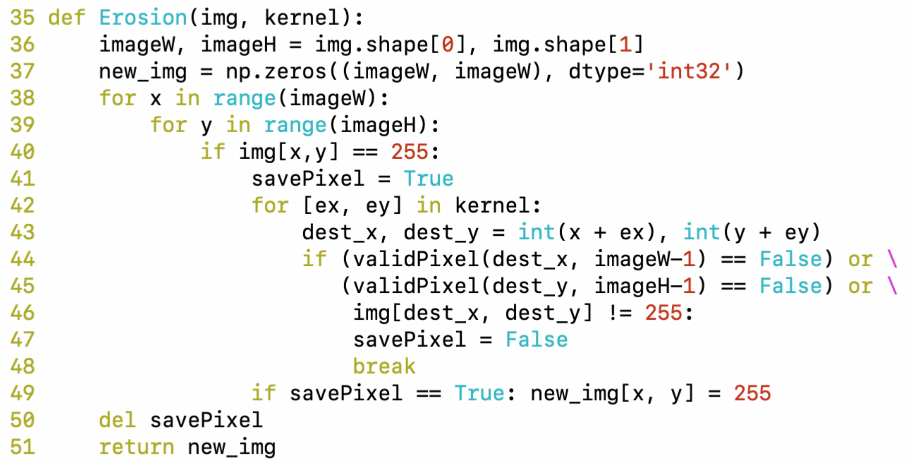
```
def binarize(img):
    imageW, imageH = img.width, img.height
    new_img = img.copy()
    new_img_pixel = new_img.load()
    for x in range(imageW):
        for y in range(imageH):
            new_img_pixel[x, y] = 255 if img.getpixel((x,y)) > 127 else 0
    new_img.save('./binarize_lena.bmp')

def validPixel(x, bound):
    if x >= 0 and x <= bound:
        return True
    return False
```

Part-1 Dilation: If the pixel value in the binarized image is 255, paste the kernel pattern.



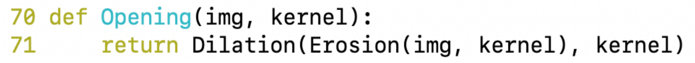
```
def Dilation(img, kernel):
    imageW, imageH = img.shape[0], img.shape[1]
    new_img = np.zeros((imageW, imageW), dtype='int32')
    for x in range(imageW):
        for y in range(imageH):
            if img[x,y] == 255:
                new_img[x,y] = 255
                for [ex, ey] in kernel:
                    dest_x, dest_y = x + ex, y + ey
                    if validPixel(dest_x, imageW-1) == True \
                        and validPixel(dest_y, imageH-1) == True:
                        new_img[dest_x, dest_y] = 255
    return new_img
```

Part-2 Erosion: If the kernel pattern is fit in the binarized image, set the origin of the kernel to 255.



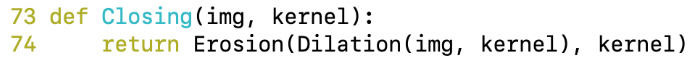
```
def Erosion(img, kernel):
    imageW, imageH = img.shape[0], img.shape[1]
    new_img = np.zeros((imageW, imageW), dtype='int32')
    for x in range(imageW):
        for y in range(imageH):
            if img[x,y] == 255:
                savePixel = True
                for [ex, ey] in kernel:
                    dest_x, dest_y = int(x + ex), int(y + ey)
                    if (validPixel(dest_x, imageW-1) == False) or \
                       (validPixel(dest_y, imageH-1) == False) or \
                        img[dest_x, dest_y] != 255:
                        savePixel = False
                        break
                if savePixel == True: new_img[x, y] = 255
    del savePixel
    return new_img
```

Part-3 Opening: By definition. Do erosion first and do dilation.



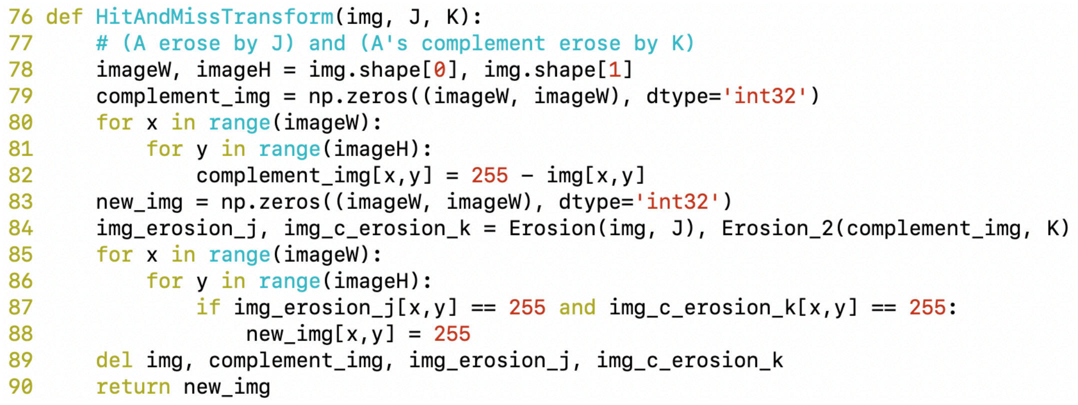
```
def Opening(img, kernel):
    return Dilation(Erosion(img, kernel), kernel)
```

Part-4 Closing: By definition. Do dilation first and do erosion.



```
def Closing(img, kernel):
    return Erosion(Dilation(img, kernel), kernel)
```

Part-5 Hit-And-Miss Transform: First, let binarized image erode by J kernel. Second, let binarized image after taking complement erode by K Kernel. Last but not least, take their intersection.



```
def HitAndMissTransform(img, J, K):
    # (A erose by J) and (A's complement erose by K)
    imageW, imageH = img.shape[0], img.shape[1]
    complement_img = np.zeros((imageW, imageW), dtype='int32')
    for x in range(imageW):
        for y in range(imageH):
            complement_img[x,y] = 255 - img[x,y]
    new_img = np.zeros((imageW, imageW), dtype='int32')
    img_erosion_j, img_c_erosion_k = Erosion(img, J), Erosion_2(complement_img, K)
    for x in range(imageW):
        for y in range(imageH):
            if img_erosion_j[x,y] == 255 and img_c_erosion_k[x,y] == 255:
                new_img[x,y] = 255
    del img, complement_img, img_erosion_j, img_c_erosion_k
    return new_img
```