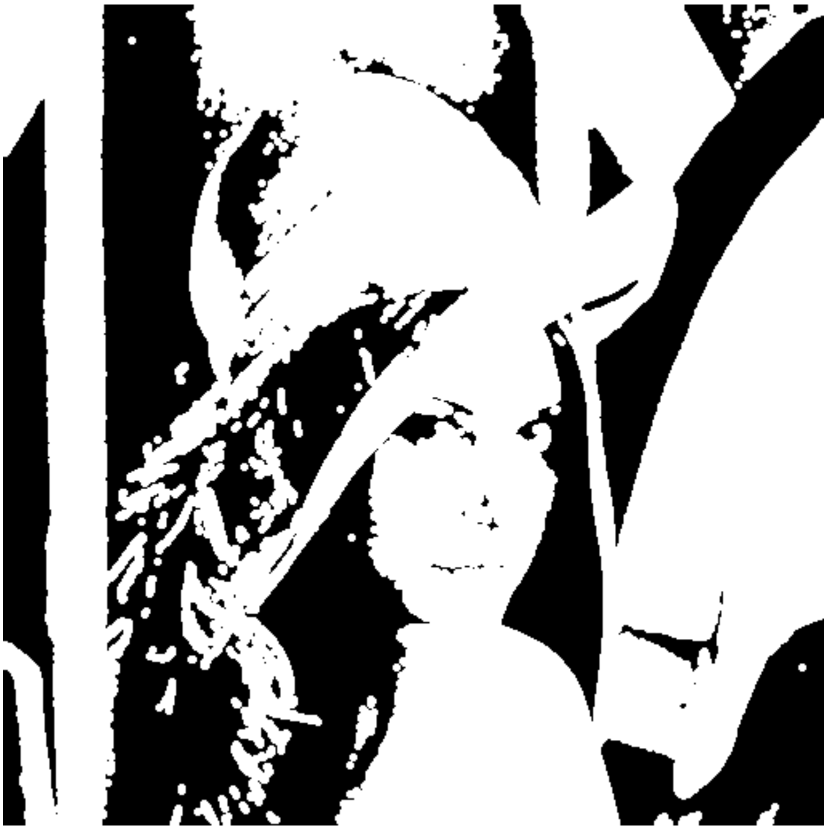
# Computer Vision Homework #4

資工四 b05902115 陳建丞

## 1. Dilation

- **Result**



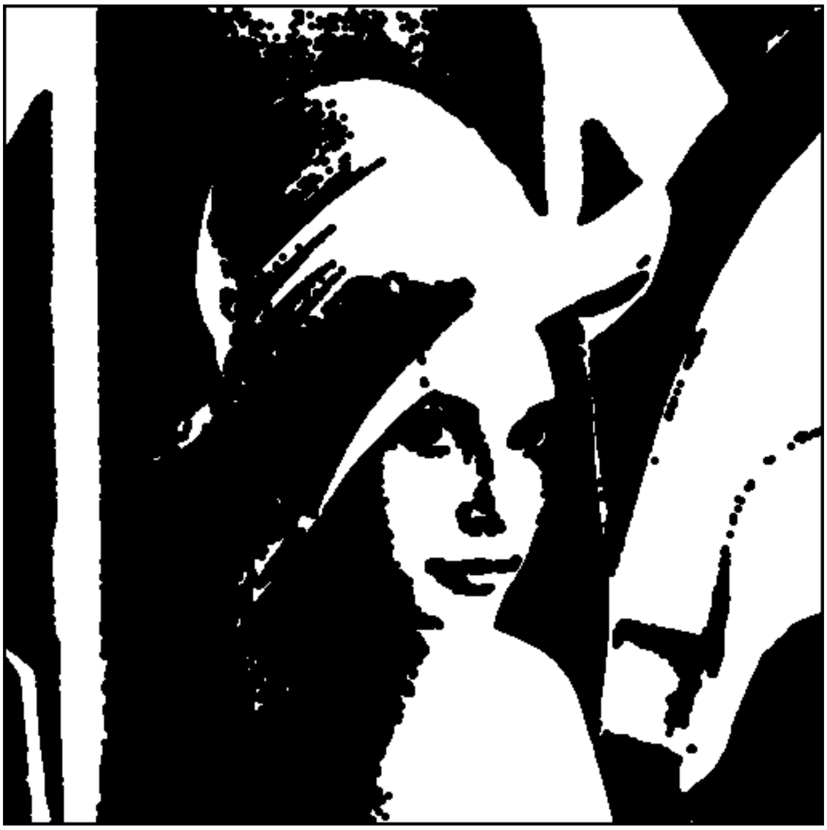
- **Implementation**
  - $A \oplus B = \{c \in E^N | c = a + b \ for \ some \ a \in A \ and \ b \in B\}$
  - **Kernel: Octagonal 3-5-5-5-3 kernel**

  Traverse every pixel of the input image. If the pixel's color is white, change the color of the neighboring pixel according to the shape of the kernel applied.

```
def Dilation(img, kernel):

  kernel = kernel.copy() * 255
  r = int((kernel.shape[0]-1)/2)
  new_img = img.copy()
  height, width = img.shape[:2]

  for i in range(height):
    for j in range(width):
      if img[i][j] == 0:
        continue

      for h in range(-r, r+1):
        for w in range(-r, r+1):
          if (i+h) < 0 or (i+h) >= height or (j+w) < 0 or (j+w) >= width:
            continue

          if new_img[i+h][j+w] == 0:
            new_img[i+h][j+w] = kernel[h+r][w+r]

  return new_img
```

## 2. Erosion

- **Result**



- **Implementation**
  - $A \ominus B = \{x \in E^N | x + b \in A\ for\ every\ b \in B\}$
  - **Kernel: Octagonal 3-5-5-5-3 kernel**

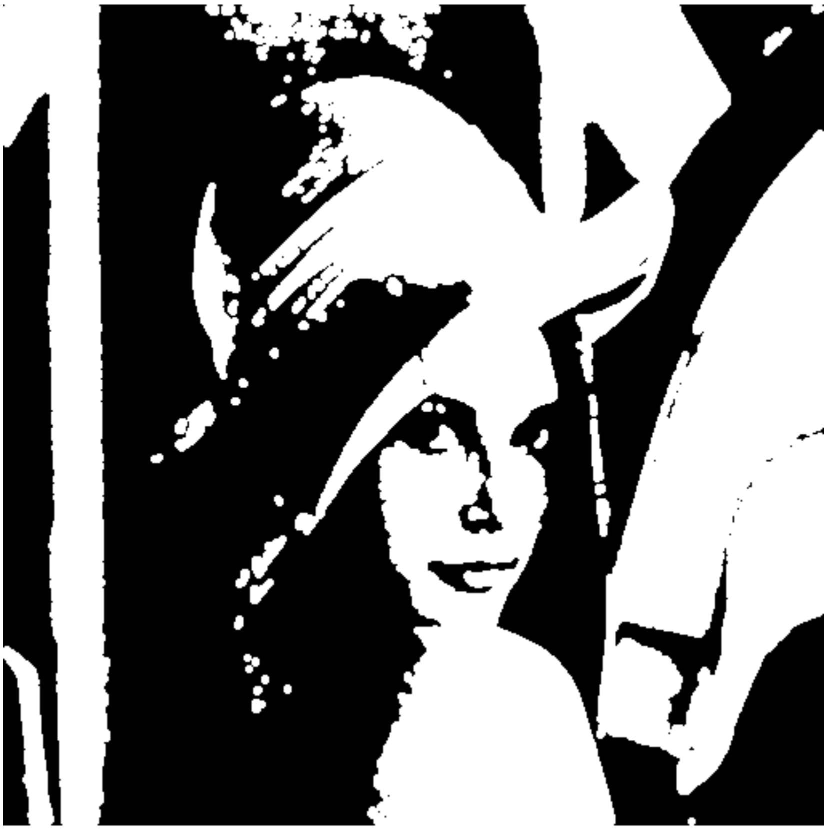
Traverse every pixel of the input image. For every pixel, check if the pixel itself and its neighboring pixels are the same as the kernel. If the shape is the same, set the central pixel as white in the output image. I use the [:] slice syntax to cut off the desired submatrix pattern from the original image to compare with the kernel.

```
def Erosion(img, kernel):
```

```
cover = kernel.copy()
kernel = kernel.copy() * 255
r = int((kernel.shape[0]-1)/2)

height, width = img.shape[:2]
new_img = np.zeros((height, width), dtype='uint8')

for i in range(height):
  for j in range(width):

    if i-r < 0 or i+r >= height or j-r < 0 or j+r >= width:
      continue

    if (np.multiply(img[i-r:i+r+1, j-r:j+r+1], cover) == kernel).all():
      new_img[i][j] = 255

return new_img
```

## 3. Opening

- **Result**



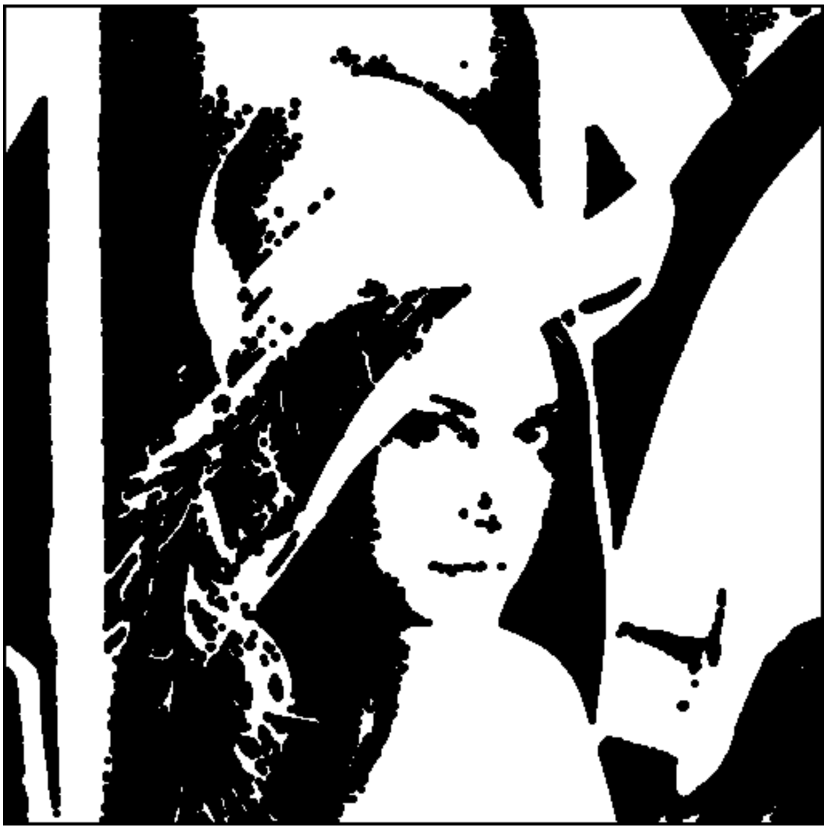
- **Implementation**
  - $B \circ K = (B \ominus K) \oplus K$

Simply apply the formula with the erosion and dilation function above.

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

## 4. Closing

- **Result**



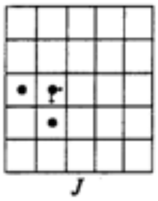
- **Implementation**
  - $B \bullet K = (B \oplus K) \ominus K$

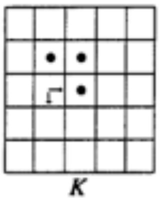
  Simply apply the formula with the erosion and dilation function above.

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

## 5. Hit and miss transformation

- **Result**

- **Implementation**
  - $A \otimes (J, K) = (A \ominus J) \cap (A^c \ominus K)$
  - **Kernel:** 



To implement the formula, just simply run `Erosion(img, J)` and `Erosion(255-img, K)` and then iterate through the two return images to find intersection.

```python
def Hit_and_miss_transform(img, J, K):

  A_J = Erosion(img, J)
  Ac_K = Erosion(255 - img , K)
  new_img = np.zeros((img.shape), dtype='uint8')

  for i in range(img.shape[0]):
    for j in range(img.shape[1]):
      if (A_J[i][j] == Ac_K[i][j]) and (A_J[i][j] == 255):
        new_img[i][j] = 255

  return new_img
```

## Note

- **Python package**
  - **skimage** : read and write image
  - **numpy** : array manipulation
- Before processing the operations above, I run `Binarize(img, 128)` function to convert **lena.bmp** into binarized image. The function is revised from previous homework.