# Computer Vision Homework #7

資工四 b05902115 陳建丞

## 1. Thinning

- **Implementation**
  - **Part1 : Yokoi Operator**

$$h(b,c,d,e) = \begin{cases} q & \text{if b} = \text{c and } (\text{d} \neq \text{b} \vee \text{e} \neq \text{b}) \\ r & \text{if b} = \text{c and } (\text{d} = \text{b} \wedge \text{e} = \text{b}) \\ s & \text{if b} \neq \text{s} \end{cases}$$

$$f(a_1,a_2,a_3,a_4) = \begin{cases} 5 & \text{if } a_1 = a_2 = a_3 = a_4 = \text{r} \\ n & \text{where n} = \text{number of } \{a_k | a_k = q\} \end{cases}$$

I call the `Yokoi(img)` function from hw6 to return the corresponding yokoi value array.

```
def Yokoi(img):

  height, width = img.shape[:2]
  yokoi = np.zeros((height, width), dtype = int)
  bin_img = img.copy()
  bin_img[bin_img == 255] = 1
  connectivity = 4
  neighbors = np.array([[(0, 0), (0, 1), (-1, 1), (-1, 0)],
            [(0, 0), (-1, 0), (-1, -1), (0, -1)],
            [(0, 0), (0, -1), (1, -1), (1, 0)],
            [(0, 0), (1, 0), (1, 1), (0, 1)]])

  h_table = np.zeros((2, 2, 2, 2))

  for b in range(2):
    for c in range(2):
      for d in range(2):
        for e in range(2):
          h_table[b, c, d, e] = h(b, c, d, e)


  # Compute h(b, c, d, e)
  for i in range(height):
    for j in range(width):

      if bin_img[i, j] == 0:
        continue

      f_input = []
```

```
      for k in neighbors:

        idx = (i, j) + k
        bin_value = []
        for m, n in idx:
          if m < 0 or n < 0 or n >= height or m >= width:
            bin_value.append(0)
          else:
            bin_value.append(bin_img[m, n])

        f_input.append(h_table[bin_value[0], bin_value[1],
bin_value[2], bin_value[3]])

      yokoi[i][j] = f(f_input[0], f_input[1], f_input[2], f_input[3])

  return yokoi
```

- **Part2 : Pair Relationship Operator**

$$h(a,m) = \begin{cases} 1 & \text{if a} = \text{m} \\ 0 & \text{otherwise} \end{cases}$$

$$y = \begin{cases} q & \text{if } \Sigma h(x_n, m) < 1 \text{ or } x_o \neq \text{m} \\ p & \text{if } \Sigma h(x_n, m) \leq 1 \text{ and } x_o = \text{m} \end{cases}$$

Follow the formula from slides, I iterate through the images and find the corresponding marked image.

```
def Pair_relationship(yokoi):

  marked_img = np.zeros((yokoi.shape[0], yokoi.shape[1]), dtype = int)
  height, width = yokoi.shape[:2]

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

      if yokoi[i][j] == 0:
        continue
      if yokoi[i][j] != 1:
        marked_img[i][j] = 1
        continue

      is_p = False
      for m, n in [(0, 1), (-1, 0), (0, -1), (1, 0)]:

        if i+m < 0 or j+n < 0 or i+m >= height or j+n >= width:
          continue

        if yokoi[i+m][j+n] == 1:
```

```
            # Mark as p
            marked_img[i][j] = 2
            is_p = True
            break
        if not is_p:
          # Mark as q
          yokoi[i][j] = 1

  return marked_img
```

- **Part3：Connected Shrink Operator**

$$h(b,c,d,e) = \begin{cases} 1 & \text{if b} = \text{c and } (\text{d} \neq \text{b} \vee \text{e} \neq \text{b}) \\ 0 & \text{otherwise} \end{cases}$$

$$f(a_1,a_2,a_3,a_4,x) = \begin{cases} g & \text{if exactly one of } a_n = 1 \\ x & \text{otherwise} \end{cases}$$

Given the result of `Yokoi()` and `Pair_relationship()`, I apply the formula above. If a pixel is $q$ from marked image, then apply $h()$ function on its 4 neighbors and then pass the result to $f()$ function to see if this pixel should be removed.Note that the $h()$ and $f()$ functions should be applied on the updated images. After 7 iterations, we can get the result.

```
def Thinning(img):

  height, width = img.shape[:2]
  bin_img = img.copy()
  bin_img = (bin_img/255).astype(int)
  neighbors = np.array([[(0, 0), (0, 1), (-1, 1), (-1, 0)],
            [(0, 0), (-1, 0), (-1, -1), (0, -1)],
            [(0, 0), (0, -1), (1, -1), (1, 0)],
            [(0, 0), (1, 0), (1, 1), (0, 1)]])


  yokoi = Yokoi(img)
  marked_img = Pair_relationship(yokoi)

  # Connected Shrink


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

      if marked_img[i][j] != 2:
        continue

      cnt = 0
```

```
      for k in neighbors:

        idx = (i, j) + k
        bin_value = []
        for m, n in idx:
          if m < 0 or n < 0 or n >= height or m >= width:
            bin_value.append(0)
          else:
            bin_value.append(bin_img[m, n])

        if h(bin_value[0], bin_value[1], bin_value[2], bin_value[3])
== 1:
          cnt += 1

      if cnt == 1:
        img[i][j] = 0
        bin_img[i][j] = 0

  return img
```

- **Result(400%)**

- **Python package**
  - **skimage** : read and write image
  - **numpy** : array manipulation
- **Other function**

`Binarize(img, threshold)` : To generate a binary image (from previous homework)

`Downsampleing(img, size)` : To downsample the given image by size.

`h(b,c,d,e)` : Return the corresponding h value.

`f(a_1, a_2, a_3, a_4)` : Return the corresponding f value.