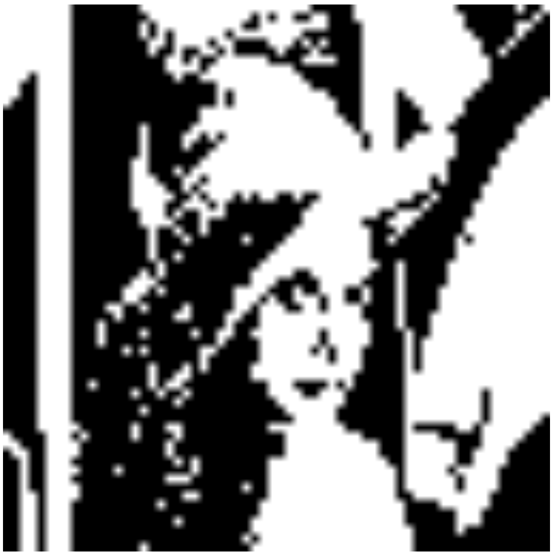
Computer Vision Homework #6

資工四 b05902115 陳建丞

1. Downsampling

- **Result (400%)**



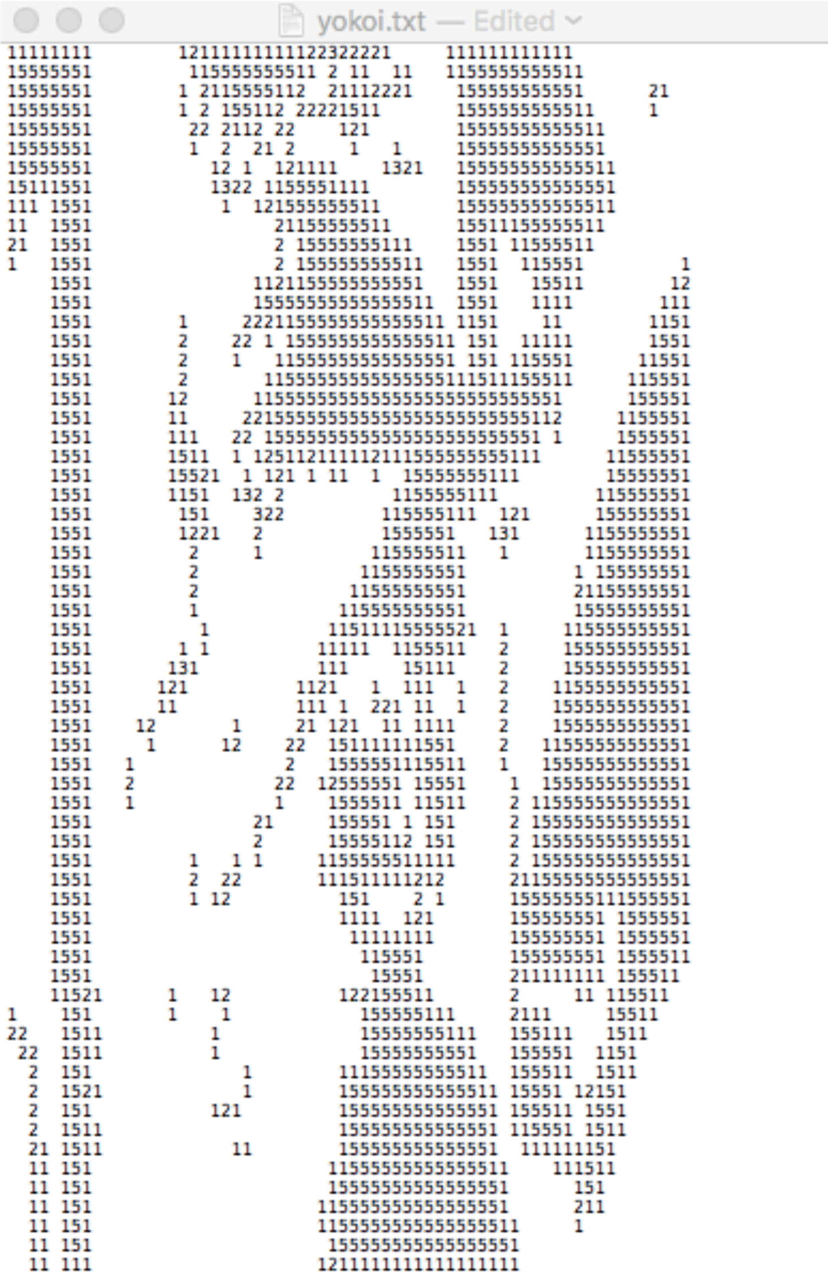
- **Implementation**

  Create a new image with size 64x64. Then use the 8x8 blocks of original image to take the topmost-left pixel as the downsampled new image.

```
def Downsampling(img, size):

  img_down = np.zeros((int(img.shape[0]/size), int(img.shape[1]/size)), dtype = int)

  for i in range(img_down.shape[0]):
    for j in range(img_down.shape[1]):
      img_down[i][j] = img[i*size][j*size]

  return img_down
```

2. Yokoi

- **Result**



```
11111111        1211111111111122322221   111111111111
15555551         115555555511 2 11  11   1155555555511
15555551        1 2115555112  21112221    155555555551      21
15555551        1 2 155112 22221511       1555555555511     1
15555551         22 2112 22    121        15555555555511
15555551         1  2  21 2     1   1     15555555555551
15555551           12 1  121111    1321   155555555555511
15111551           1322 1155551111        155555555555551
111 1551            1  121555555511       155555555555511
11  1551                 21155555511      15511555555511
21  1551                 2 15555555111    1551 11555511
1   1551                 2 155555555511   1551  115551         1
    1551               1121155555555551   1551   15511        12
    1551               15555555555555511  1551   1111        111
    1551        1     2221555555555555511 1151    11        1151
    1551        2    22 1 1555555555555511 151  11111       1551
    1551        2    1   11555555555555551 151 115551      11551
    1551        2       11555555555555555111511155511     115551
    1551       12      11555555555555555555555555551      155551
    1551       11     221555555555555555555555555112     1155551
    1551       111   22 15555555555555555555555551 1     1555551
    1551       1511  1 125112111112111555555555111      11555551
    1551       15521  1 121 1 11  1  15555555111        15555551
    1551       1151  132 2          1155555111         115555551
    1551        151    322         115555111 121       155555551
    1551        1221   2           1555551   131      1155555551
    1551         2     1          115555511   1       1155555551
    1551         2               1155555551          1 155555551
    1551         2              11555555551          21155555551
    1551         1             115555555551          15555555551
    1551          1           11511115555521  1     115555555551
    1551        1 1          11111  1155511   2     155555555551
    1551       131           111     15111    2     155555555551
    1551      121          1121   1  111  1   2    1155555555551
    1551      11           111 1  221 11  1   2    1555555555551
    1551    12       1     21 121  11 1111    2    1555555555551
    1551     1      12    22  151111111551    2   11555555555551
    1551   1              2   1555551115511   1   15555555555551
    1551   2             22  12555551 15551    1  15555555555551
    1551   1             1    1555511 11511    2 115555555555551
    1551               21     155551 1 151     2 155555555555551
    1551               2      15555112 151     2 155555555555551
    1551         1   1 1     1155555511111     2 155555555555551
    1551         2  22       111511111212      21155555555555551
    1551         1 12          151    2 1      15555555111555551
    1551                       1111  121       155555551 1555551
    1551                        11111111       155555551 1555551
    1551                         115551        155555551 1555511
    1551                          15551        211111111 155511
    11521      1   12          122155511       2     11 115511
1    151       1    1            155555111     2111     15511
22   1511          1             15555555111   155111   1511
 22  1511          1             15555555551   155551  1151
  2  151              1        11155555555511  155511  1511
  2  1521             1        155555555555511 15551 12151
  2  151           121         155555555555551 155511 1551
  2  1511                      155555555555551 115551 1511
  21 1511            11        155555555555551  111111151
  11 151                      11555555555555511    111511
  11 151                      15555555555555551      151
  11 151                     115555555555555551      211
  11 151                     1155555555555555511     1
  11 151                      155555555555555551
  11 111                     1211111111111111111
```

- **Implementation**

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

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

I follow the formula above to iterate through downsampled image. I write function `h(b,c,d,e)` and `f(a_1,a_2,a_3,a_4)`. Since it takes time to calculate $h(b, c, d, e)$ every time, I create a h table to store the h values all 16 possible (b, c, d, e).

```
def Yokoi(img):

  yokoi = np.zeros((img.shape[0], img.shape[1]), 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(img.shape[0]):
    for j in range(img.shape[1]):

      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 >= img.shape[0] or m >= img.shape[1]:
            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
```

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

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

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

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