d11922022 資工所 楊寓鈞

Environment

- python3.9
- Pillow 10.0.0, numpy 1.25.2, pandas 2.0.3
- opencv-python 4.8.0.74

I/O

```
from PIL import Image
import numpy as np
import copy

img = Image.open('./lena.bmp') # load lena.bmp
img_array = np.array(img) # pixel content saved in np.array
width, height = img_array.shape # get `width` and `height`
img_list = img_array.tolist() # transform pixel content into list
```

Part 1

a. binarize

binarize at 128

```
result = copy.deepcopy(img_list)

for y in range(height):
        for x in range(width):
                result[y][x] = 255 if result[y][x] >= 128 else 0

img_ = Image.fromarray(np.array(result, dtype='uint8'), mode='L')
img_.save('./lena_binarize.bmp')
```

1. remove mutability of nested list
2. loop through every row
3. in each even row `y`, loop through every column of the row
4. for each column `x` (of the row `y`),
   + set 255 if pixel value >= 128

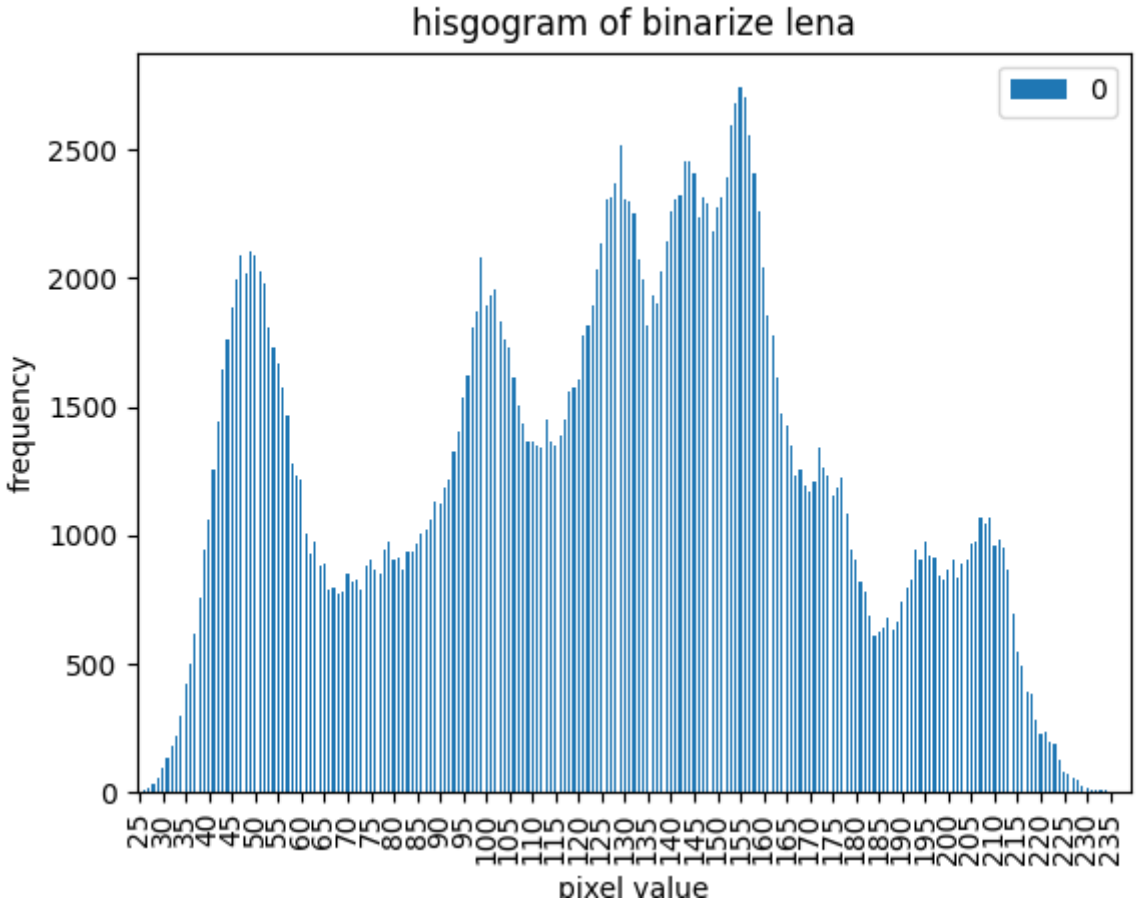
+ set 0 if pixel value < 128

b. histogram

```
import matplotlib.ticker as ticker
import matplotlib.pyplot as plt
import pandas as pd
%matplotlib inline

result = img_array.copy()
histogram = dict()



for y in range(height):
        for x in range(width):
                bin = histogram.get(result[y][x], 0)
                histogram[result[y][x]] = bin + 1

histogram = dict(sorted(histogram.items(), key=lambda x: x[0]))
df = pd.DataFrame({k:[v] for k,v in histogram.items()}).T
```

```
ax = df.plot.bar(title='hisgogram of binarize lena', xlabel='pixel value',
ylabel='frequency')
ax.xaxis.set_major_locator(ticker.MultipleLocator(base=5))
plt.savefig('histogram.png')
```

1. traverse through all pixels of the binarized image
2. create an dict to record gray level and it's corresponding pixel amounts
3. take pixel's gray level as key, accumulate the counts of the pixels at that gray level as key's value
4. plot histogram via pandas api



c. connected components

In the beginning, I tried the classical algorithm to do label propagation and get transitive closure. But I found the complexity of transitive closure is $O(n^3)$ where n is the number of relation pair, which was unacceptable to execute.

So I found a data structure called 'union find' and with 'path compression' algorithm can get this task done under acceptable execution time. My reference is: https://www.youtube.com/watch?v=ibjEGG7ylHk.

```
# make binarized pixels to get a label 1
for y in range(height):
        for x in range(width):
                result[y][x] = 1 if result[y][x] == 255 else 0


parent = [i for i in range(width*height)]

def union_find(x):
        origin_x =x
        while parent[x] != x:
                x = parent[x]
                parent[origin_x] = x
        return x

# connected components + path compressiion
label = 2
for y in range(height):
        for x in range(width):
                left_set, up_set = False, False
                if result[y][x] == 1:
                        if x > 0 and result[y][x-1] > 1:
                                # if left pixel is set before
                                # set current pixel to letf pixel's root
label
                                result[y][x] = union_find(result[y][x-1])
                                left_set = True

                        if y > 0 and result[y-1][x] > 1:
                                # if up pixel is set before
                                if left_set:
                                        # if current pixel's left pixel is
set before, set current pixel's parent to up pixel's root label
                                        parent[result[y][x]] =
union_find(result[y-1][x])
                                else:
                                        # if current pixel's left not set,
set current pixel to letf pixel's root label
                                        result[y][x] = union_find(result[y-
1][x]) #result[y-1][x]
                                up_set = True

                        if not left_set and not up_set:
                                # for the left most pixel
                                result[y][x] = label
                                label += 1
```

```
# merge components
# record bounding box and center
def default_dict():
        return {'count':0, 'center x':0, 'center y':0, 'left':float('inf'),
'right':float('-inf'), 'up':float('inf'), 'down':float('-inf'),
'pixels':list()}


box = defaultdict(default_dict)
        for y in range(height):
                for x in range(width):
                        if result[y][x] > 1:
                                result[y][x] = union_find(result[y][x])

                                cc = result[y][x]

                                box[cc]['left'], box[cc]['right'] =
min(box[cc]['left'], x), max(box[cc]['right'], x)
                                box[cc]['up'], box[cc]['down'] = min(box[cc]
['up'], y), max(box[cc]['down'], y)
                                box[cc]['center x'] += x
                                box[cc]['center y'] += y
                                box[cc]['count'] += 1
                                box[cc]['pixels'].append((y, x))


# get number of components
# get bounding box and center
num_of_cc = max(box)
for cc in box:
        box[cc]['center x'] = round(box[cc]['center x']/box[cc]['count'])
        box[cc]['center y'] = round(box[cc]['center y']/box[cc]['count'])
```

1. for every pixel set to 1, if they don't have any preceding neighbor pixel (left or up), then set the pixel to a new label
2. for every pixel set to 1, if they have a preceding left neighbor pixel, set their label to its predecessor's root parent. Also do path compression while finding the root of the predecessor.
3. for every pixel set to 1, if they have a preceding up neighbor pixel and the preceding left neighbor pixel is set, update their parent to the predecessor's root. If the preceding left neighbor pixel is not set, then set the pixel to its up predecessor's root.

   Also do path compression while fine the root parent of the predecessor.

4. merge connected component
5. calculate bounding boxes and the centers for connected components
6. draw bounding boxes and the centers for connected components

References

https://www.youtube.com/watch?v=ibjEGG7ylHk
https://www.cnblogs.com/hfc-xx/p/4666223.html
https://www.geeksforgeeks.org/union-by-rank-and-path-compression-in-union-find-algorithm/