Computer Vision HW2: Basic Image Manipulation (2)

R10741015 鄭傑鴻

Sept. 14, 2022

Original Picture:

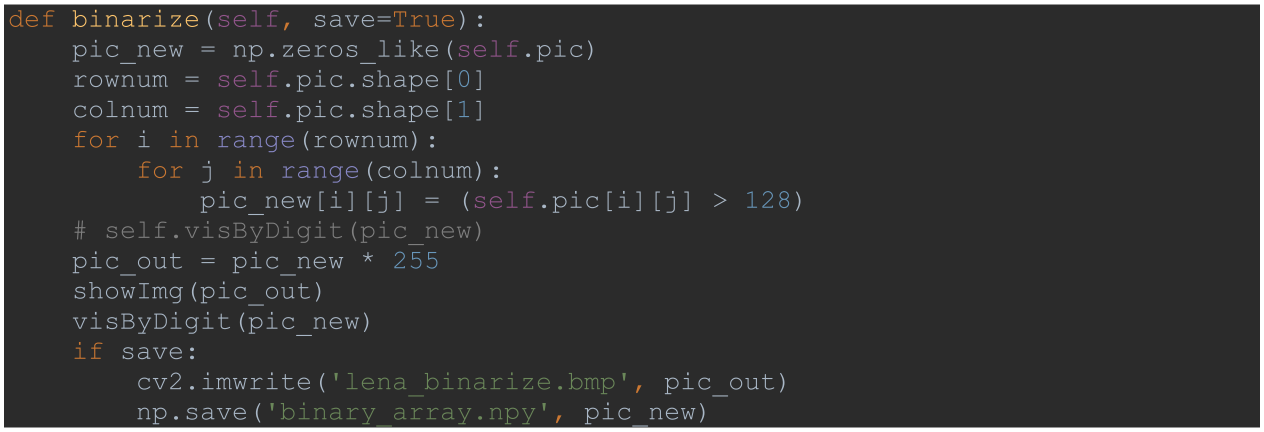
(With shape (512, 512))

(a) Generate a binary image, threshold at 128

- Description:

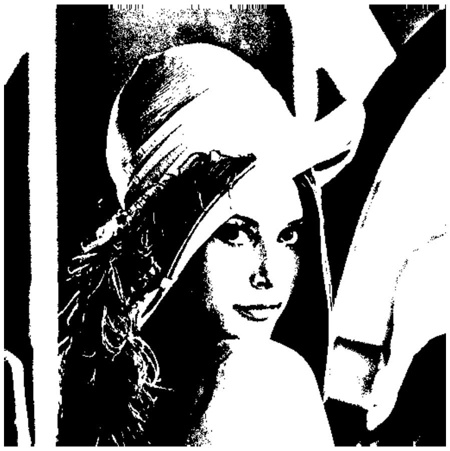
1. Initiate a new array pic_new, with shape equals to the original picture
2. Iterate over the array from left to right, from top to down
3. For every element (pixel) in the new picture, denoted as $pic\_new[i][j]$, if the correspondent pixel in the original picture $pic[i][j]$ is greater than 128, assign value 1 to $pic\_new[i][j]$, else assign 0 to $pic\_new[i][j]$. (i & j from 0 to 511)
4. Visualize by multiply the pic_new array by 255, such that pixels above threshold (128) result in white pixel, and pixels below threshold (128) result in black pixel.

- Code:



```python
def binarize(self, save=True):
    pic_new = np.zeros_like(self.pic)
    rownum = self.pic.shape[0]
    colnum = self.pic.shape[1]
    for i in range(rownum):
        for j in range(colnum):
            pic_new[i][j] = (self.pic[i][j] > 128)
    # self.visByDigit(pic_new)
    pic_out = pic_new * 255
    showImg(pic_out)
    visByDigit(pic_new)
    if save:
        cv2.imwrite('lena_binarize.bmp', pic_out)
        np.save('binary_array.npy', pic_new)
```

- Resulting Image:

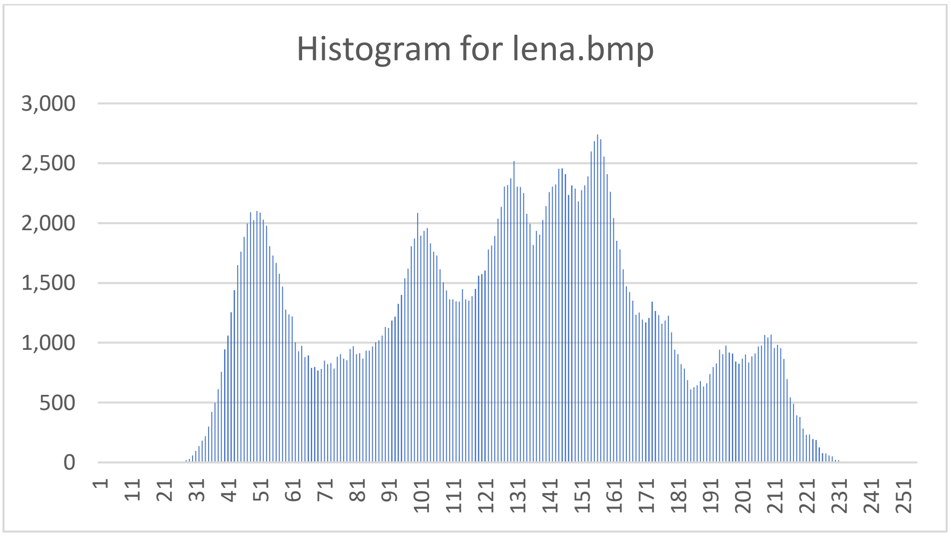


(b) Generate a histogram

- Description
  1. Initiate a new zero array $count\_array$, with shape (255,), each element denotes the count of each intensity (from 0 to 255)
  2. Iterate over the original picture from left to right, top to bottom
  3. For every pixel, add the count for its intensity by 1 (i.e. $count\_array[pic[i][j]] += 1$)
  4. Output the $count\_array$, and use excel to draw the corresponding histogram
- Code

```
def histoize_hardcore(self):
    count_array = np.zeros((255,), dtype=np.uint32)
    for i in range(512):
        for j in range(512):
            count_array[self.pic[i][j]] += 1
    np.savetxt("histogram.csv", count_array, delimiter=",")
```

- Result



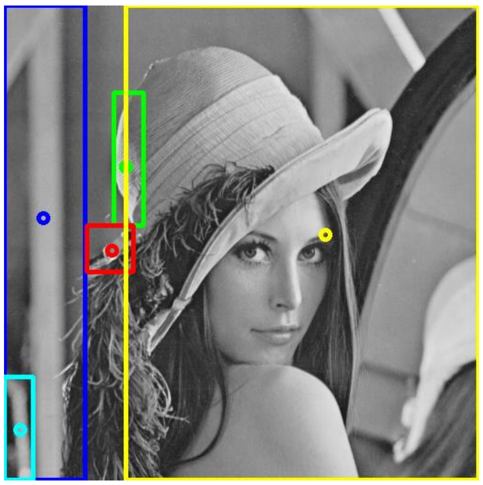
(c) Generate connected components with visualization

- Description / Algorithm

  Using “The Classical Algorithm”:

  1. Set up a new zero array cc_array, with shape (512, 512), to store information about connected components for each pixel
  2. Initiate the cc_array, making each pixel over threshold (128) a connected component containing only itself, label from *1, 2, …n*. For pixels under threshold, label them as *0*.
  3. Iterate over the cc_array from left to right, from top to bottom, and propagate the labels
     i. If the current element $cc\_array[i][j]$ is not background, try to find labels of connect components above and leftwards of it
     ii. If the above and leftward pixels both are not backgrounds, change the label $cc\_array[i][j]$ to *min(above label, leftward label)*, and record a pair of *(above label, leftward label)* into a list named *trans*
     iii. If either one (but not both) is non-background, change the label $cc\_array[i][j]$ to the label of that non-background
  4. Create multiple trees based on the connected pairs in *trans*, such that labels connected to one another share the same root
  5. Iterate over the cc_array again, translating each label into the root label
  6. Filter out connected component with less than 500 pixels
  7. Visualization: Draw bounding boxes along with its centroids, and use different color to distinguish

- Result



- Code

  Since it’s a bit lengthy, pleas kindly refer to the source code file.