CS270 HW2 Report

Tianyuan Wu, 63305667

Problem 1

1. Truncated Huffman coding
   In this homework, I implement truncated Huffman coding algorithm in following steps:
   (1) The input is a sequence contains integers, and most of them are 0.
   (2) Use a `Counter()` to count the frequency of each number.
   (3) Find all numbers which has the frequency less than the `Threshold` (0.0001 in this homework), merge them together as a union, and use binary encoding to encode them (i.e. 0, 1, 10, 11, …)
   (4) After merging the nodes with small frequency, we construct the huffman tree. First, for each number, we construct a Huffman tree node, which contains the number and the frequency of the number.
   (5) Then we choose 2 nodes with smallest frequency, and add a new node to be the parent of them, and assign the frequency of the new node to be the sum of the two nodes we choosed.
   (6) Repeat step (5) until all nodes are merged to one node.
   (7) Then we give each leaf node a unique huffman code. First, we init the huffman code of the node to be the empty sequence (i.e. “”), then find a path from the root to the leaf node, each time you go left, add a ‘0’ after the huffman code, and for each time going right, add a ‘1’ after the huffman code.
   (8) Run step (7) for all leaf nodes, and we will get the all huffman codes.
   (9) For the truncated numbers, the code of them will be `“HuffmanCode + BinaryCode”`. For example, if the Huffman code is “101”, and the binary code is “11”, then the encoding result of this node will finally be “10111”.
2. Original Y channel of “Lena.tiff”. Due to my result is based on *OpenCV-Python*, the result of this image is slightly different with the MATLAB result.

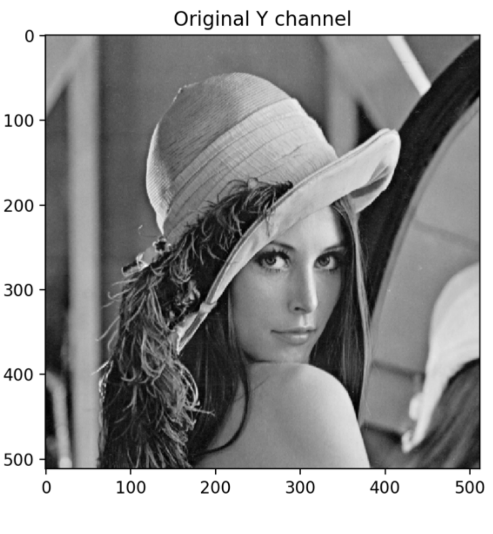


Fig.1. Original Y channel

3. The Y channel after compression

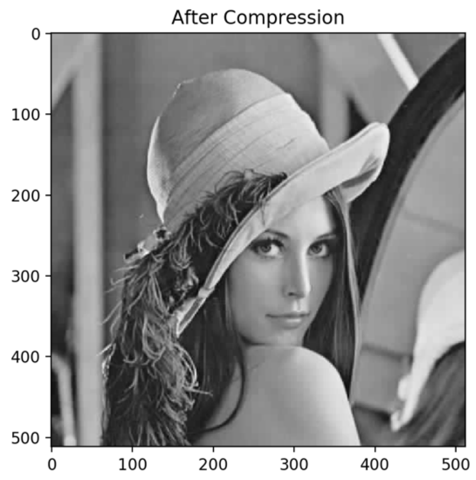


Fig.2. After compression

(We can find some slight difference between this 2 images).

4. Encoding results of the given block. For the OpenCV is different from MATLAB, the given 8*8 block is different from the right-top corner of the original image. So I just type the block and treat it as a 8*8 image, do DCT and Zig-Zag scan of it, and them give the truncated huffman code. Here are the truncated huffman encoding results:

```
      ===============Huffman Code for given block===============
Huffman encoding result:
['01100', '0011', '01101', '01110', '0010', '0000', '1', '00011',
'0101', '0100', '1', '1', '0101', '1', '01111', '1', '1', '1', '1',
'1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1',
'1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1',
'1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1',
'1', '1', '1']
Huffman code dictionary:
{66.0: '01100', 6.0: '0011', 15.0: '01101', -7.0: '01110', -4.0:
'0010', -6.0: '0000', -2.0: '00011', 2.0: '0100', -1.0: '01111', 1.0:
'0101', 0.0: '1'}
```

5. Parameters:
   a) `Root mean square error: 5.085659`
   b) `Mean-square signal-to-noise ratio: 683.506484`
   c) `Compressed ratio: 5.872796110849744`

Problem2

1. Algorithm:
   (1) The input is 2 images you want to blend together.
   (2) Calculate the Gaussian Pyramids of these 2 images, named as I_{1i} for left one, and

I_{2i} for the right one.

(3) Calculate the Laplace Pyramids of 2 images, using

$$L_i = I_i - (G * I_i) \downarrow 2$$

(4) Merge each layer of laplace pyramid. That is, if you want to merge 2 images with size 128*128, we know L_0 has size 128*128. So, we take left 128*64 part of L_{10}, and right 128*64 part of L_{20}, then take left 64*32 part of L_{11}, and right 64*32 part of L_{21}. Repeat this step until merging the top level of Laplace pyramid.

(5) Upsampling each layer, and get some layers which has the same size as the orignal image, and sum them together to get the result.

2. Fisrt, we apply linear registration on the "man.jpg", and the result is shown in Fig.3. Then, apply TPS transformation on the linear transform result, and clip it to be the same shape as "girl.jpg" (Fig.4).

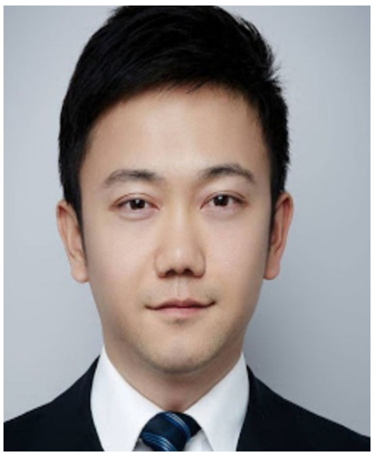


Fig.3. Linear transformation

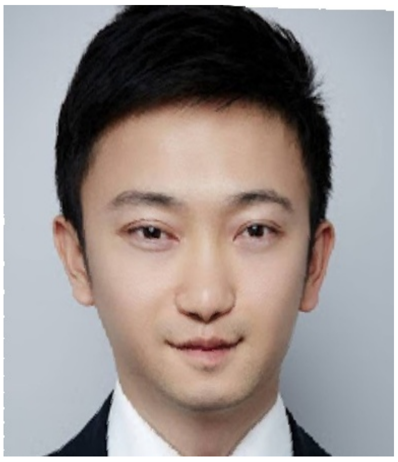


Fig.4. TPS transformation

3. The results of Pyramid blending. I've tried different levels of the pyramid, and get different results:

a) Use 4-level Laplace Pyramid

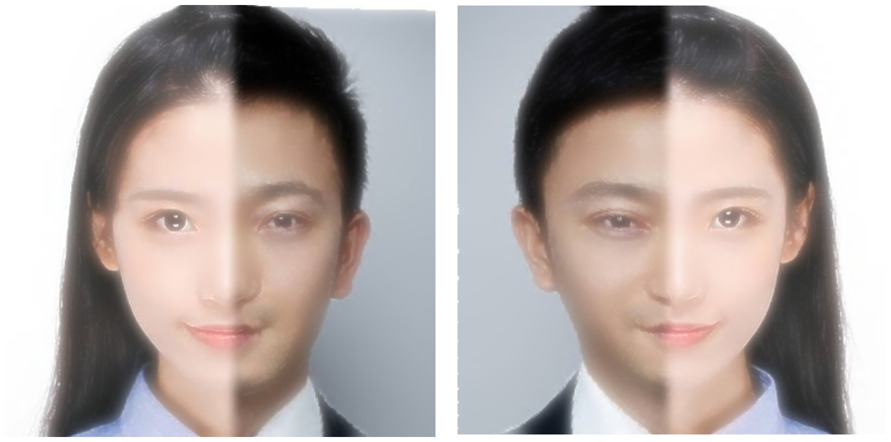


Fig.5. 4-level Pyramid

b) Use 5-level Laplace Pyramid

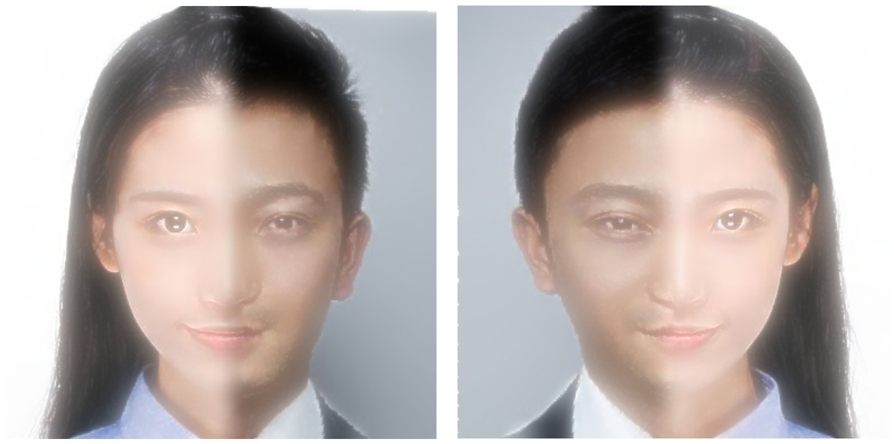


Fig.6. 5-level Pyramid

c) Use 6-level Laplace Pyramid (The performance is not good)

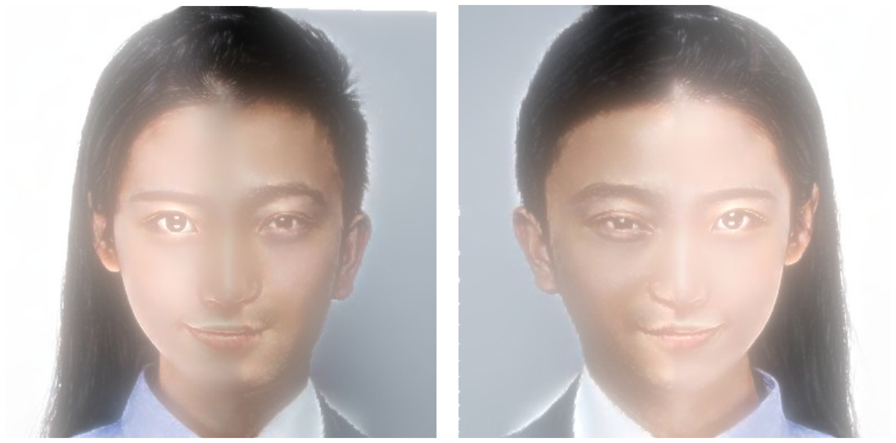


Fig.7. 6-level Pyramid

4. Hard blending

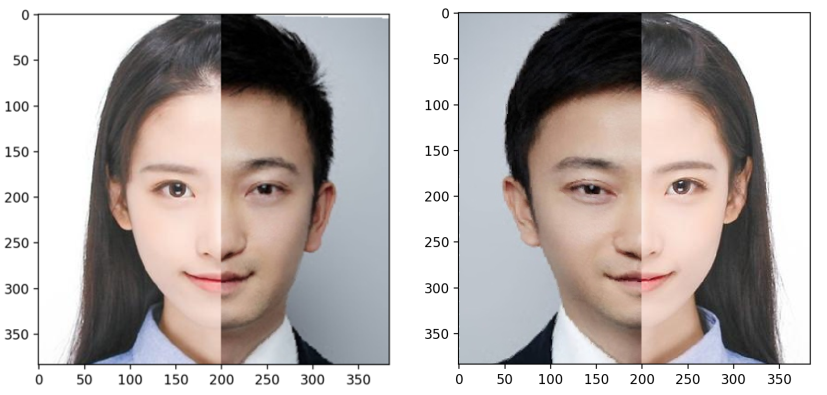


Fig.8. Hard blending

Problem 3

1. Algorithm:
   a) Algorithm - embedding:
      (1) The input is an original image and a logo to be watermarked.

(2) Do DCT for the whole image

(3) Construct a map, which `key=DCT[i, j]` and `val=(i, j)`, sort it by the keys (i.e. the magnitudes), then we can find K largest coefficients (not include DC), called c_1, c_2, … c_K.

(4) The logo image can be represented as w_1, w_2, … w_K. and we embed this sequence by:

$$c_i' = c_i(1 + \alpha w_i)$$

(5) Replace c_i by c_i', and take the inverse DCT

b) Algorithm – Extract logo:

(1) The input is, an image with embedded logo, and the original iamge (or known locations)

(2) Do DCT for the embedded image, extract K coefficients at known locations.

(3) Extract w_i by:

$$w_i = (\frac{c_i'}{c_i} - 1)/\alpha$$

(4) Find all w_i, and make it up to be the watermark

2. The new image (after add the logo watermark), which is similar to the original one.

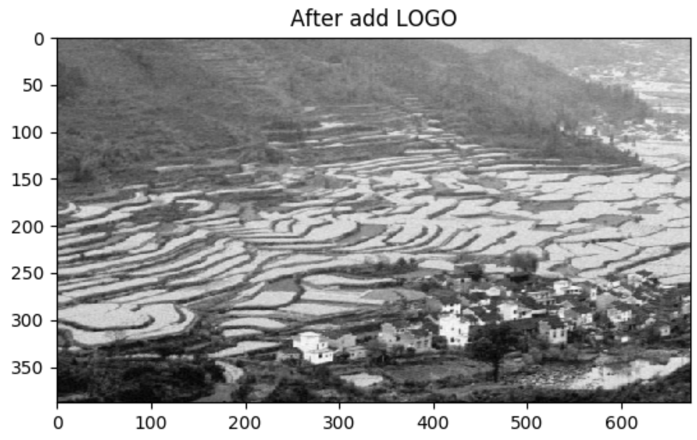


Fig.9. Shangrao (After add logo)

3. Extracted LOGO from the image shown above.

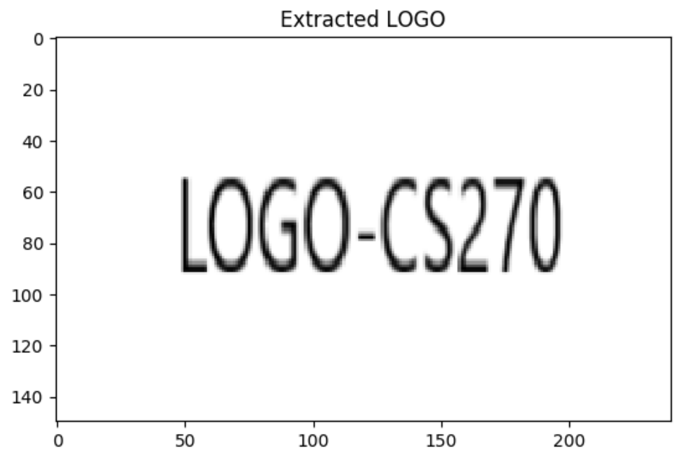


Fig.10. Extracted LOGO

4. Add random gaussian noise. After adding noise, the image is shown below (Fig.11).

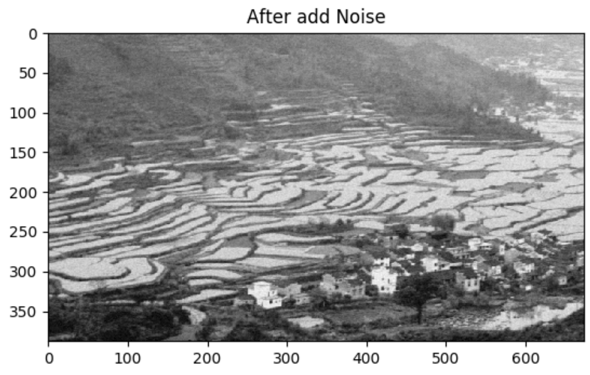
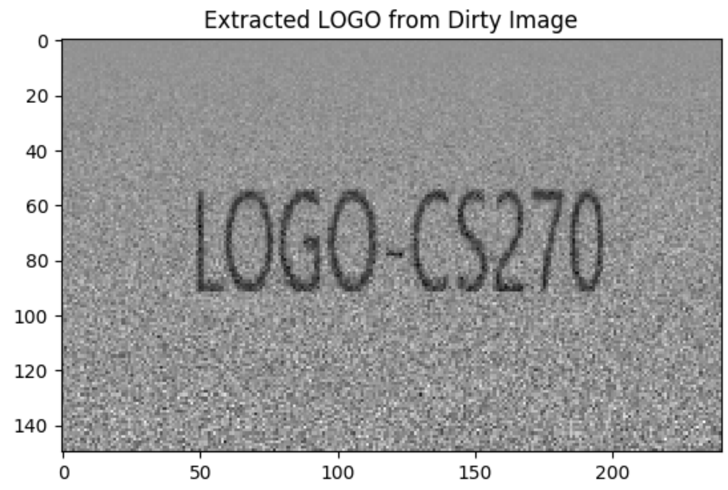


Fig.12. SR_Noise

5. Extracted LOGO from the image which was added noise (SR_Noise.png):


Extracted LOGO from Dirty Image
LOGO-CS270
0
20
40
60
80
100
120
140
0
50
100
150
200