

# Image Processing Programming

- Assignment3 report ( due to 2022/01/07 )
- 310553015 彭琍菱

## 實作環境與使用套件

使用語言：python3

使用套件：

- **numpy**：矩陣相關運算操作
- **scipy - fftpack**：編碼壓縮過程中會使用
- **PIL - Image**：圖像相關操作（例如讀寫、轉換色彩等等）

## 影像處理

以下會先介紹我實作 encoder & decoder 時的流程，以及說明嘗試的 experimentation 是什麼，與其程式碼不同的部分，我總共嘗試了 11 張圖片，後續會一一介紹跟分析。另外寫在 function 跟 import 等不在 main 裡的程式碼會放在 Rest of the code 目錄底下（最後面）

以下是我 Encoder & Decoder 的輸入輸出格式

格式	Encoder ( <code>encoder.py</code> )	Decoder ( <code>decoder.py</code> )
Input	JPEG	txt (from the output of encoder)
Output	txt	JPEG

Then compare the input of encoder and the output of decoder.

## Encoder

**RGB > YCbCr > DCT > Quantization > Block to zigzag > Encoding > 0101000101...from JPEG**

0101000101...from JPEG 這個部分我是存成 .txt 檔，不過因為這裡的 report 實在是沒辦法看出什麼東西，就是一堆 01，所以就不放了

## Decoder

**0101000101...from JPEG > Decoding > Zigzag to block > Dequantization > inverse DCT > YCbCr > RGB**

就是 encoder 反過來

# Experimentation

我做的主要實驗是使用 different block size 去嘗試觀察影像差的差異，皆會先 encode 再 decode 後進行觀察

主要變因：2 x 2，4 x 4，8 x 8 (standard) 這三個不同 Block size，所以後面我也會分別列出差異

## Quantization Table For Different Block Size

說明一下，JPEG Standard Quantization Table Coefficients 是如下這樣（跟老師提供的一樣）

Quantization Table: Luminance								Quantization Table: Chrominance							
16	11	10	16	24	40	51	61	17	18	24	47	99	99	99	99
12	12	14	19	26	58	60	55	18	21	26	66	99	99	99	99
14	13	16	24	40	57	69	56	24	26	56	99	99	99	99	99
14	17	22	29	51	87	80	62	47	66	99	99	99	99	99	99
18	22	37	56	68	109	103	77	99	99	99	99	99	99	99	99
24	35	55	64	81	104	113	92	99	99	99	99	99	99	99	99
49	64	78	87	103	121	120	101	99	99	99	99	99	99	99	99
72	92	95	98	112	100	103	99	99	99	99	99	99	99	99	99

但是我使用的是這個👉 Quantization Table for: Photoshop - (Save For Web 080) [ref](#)

Which the following matrix can be used to evaluate the JPEG compression quality of Photoshop's .

Note :  
Photoshop CS2 - The JPEG quality range for Photoshop CS2 runs from 1-12. It is expected that this range was used to give the user an idea that 10 is really the most anyone should use, and that 11 and 12 are there purely for experimental reasons (not much gain in quality for a large increase in file size).

Quantization Table	2 x 2	4 x 4	8 x 8
Luminance	[ 2, 2 ], [ 3, 4 ]	[ 2, 2, 4, 5 ], [ 3, 4, 5, 8 ], [ 4, 5, 7, 10 ], [ 5, 7, 9, 12 ]	[ 2, 2, 2, 2, 3, 4, 5, 6 ], [ 2, 2, 2, 2, 3, 4, 5, 6 ], [ 2, 2, 2, 2, 4, 5, 7, 9 ], [ 2, 2, 2, 4, 5, 7, 9, 12 ], [ 3, 3, 4, 5, 8, 10, 12, 12 ], [ 4, 4, 5, 7, 10, 12, 12, 12 ], [ 5, 5, 7, 9, 12, 12, 12, 12 ], [ 6, 6, 9, 12, 12, 12, 12, 12 ]
Chrominance	[ 11, 14 ], [ 14, 12 ]	[ 11, 14, 12, 12 ], [ 14, 12, 12, 12 ], [ 12, 12, 12, 12 ], [ 12, 12, 12, 12 ]	[ 3, 3, 5, 9, 13, 15, 15, 15 ], [ 3, 4, 6, 11, 14, 12, 12, 12 ], [ 5, 6, 9, 14, 12, 12, 12, 12 ], [ 9, 11, 14, 12, 12, 12, 12, 12 ], [ 13, 14, 12, 12, 12, 12, 12, 12 ], [ 15, 12, 12, 12, 12, 12, 12, 12 ], [ 15, 12, 12, 12, 12, 12, 12, 12 ], [ 15, 12, 12, 12, 12, 12, 12, 12 ]

所以程式碼的部分， $2 \times 2$ 、 $4 \times 4$ 、 $8 \times 8$  寫法只差在主程式中 block\_size\_num 這個變數與這兩個 quantization table，下面以  $2 \times 2$  為例

```
block_size_num = 2
```

```
def load_quantization_table2(component):  
  
    # for block_size 2x2  
  
    # Quantization Table for: Photoshop - (Save For Web 080)  
  
    #亮度通道的量化表  
    if component == 'lum':  
        q = np.array([[2, 2],  
                       [3, 4]])  
    #色度通道的量化表  
    elif component == 'chrom':  
        q = np.array([[11, 14],  
                       [14, 12]])  
    else:  
        raise ValueError((  
            "component should be either 'lum' or 'chrom', "  
            "but '{comp}' was found").format(comp=component))  
  
    return q
```

接下來請看結果與分析，我可以分成有明顯區別以及無明顯區別

有明顯失真

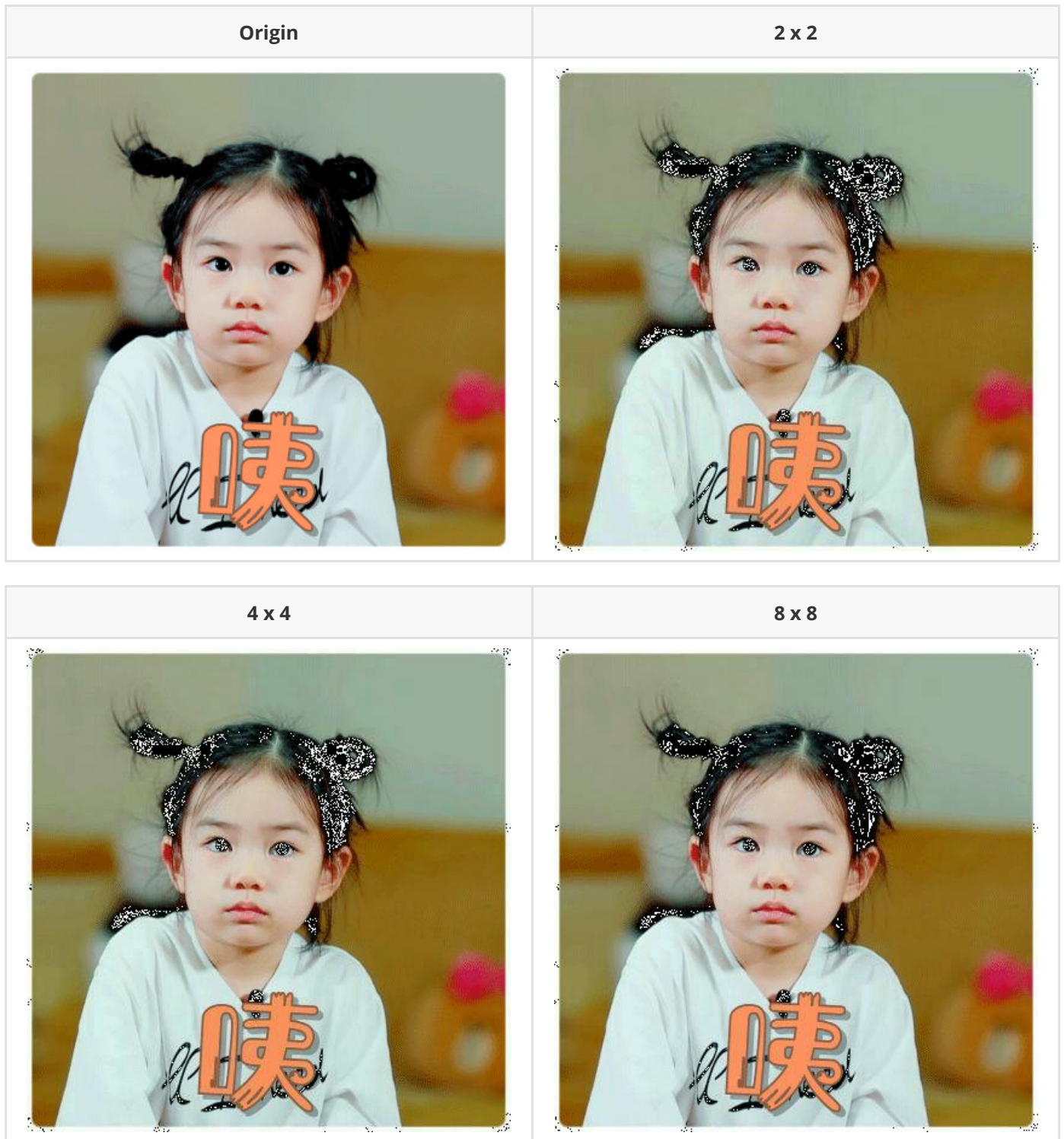
這邊列一些相對明顯有失真的影像

👉 如果放大圖片，可以看到左眼下緣、左手臂、左馬尾這三處失真明顯，在這張圖的表現上，4 x 4 比 2 x 2 差，2 x 2 又比 8 x 8 ( standard ) 差





Origin	2 x 2
	
4 x 4	8 x 8
	



📌 可以明顯看到頭髮、眼睛、麥克風這三處失真明顯，在這張圖的表現上，4 x 4 比 2 x 2 差，2 x 2 又比 8 x 8 ( standard ) 差



📌 放大圖片，可以看到貓米的眼睛、鬍鬚、頭頂黑色地方這三處有失真，其中眼睛以及鬍鬚更為明顯，在這張圖的表現上，4 x 4 跟 8 x 8 ( standard ) 差不多，2 x 2 反而略好

Origin	2 x 2
	
4 x 4	8 x 8
	

有些微失真



這邊列一些不那麼明顯的失真影像

👉 放大圖片，可以看到衣服左上角的英文字母 he 有些微白點，表現上，4 x 4 跟 8 x 8 ( standard ) 差不多，但是失真在不同地方，而 2 x 2 略好

Origin	2 x 2
	
4 x 4	8 x 8
	



📌 放大圖片，可以底下中文字“別惹我”的“別”字上有黑點，在這張圖的表現上，2 x 2、4 x 4、8 x 8 (standard) 無明顯差異

Origin	2 x 2
	
4 x 4	8 x 8
	

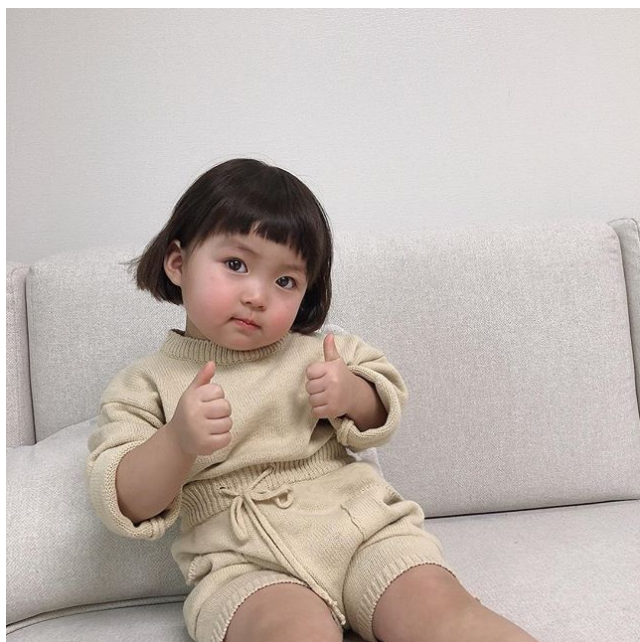


📌 放大圖片，可以看到小女孩的衣領黑色地方有依稀但是還算明顯的失真，在這張圖的表現上，4 x 4 跟 2 x 2 差不多，8 x 8 比較好

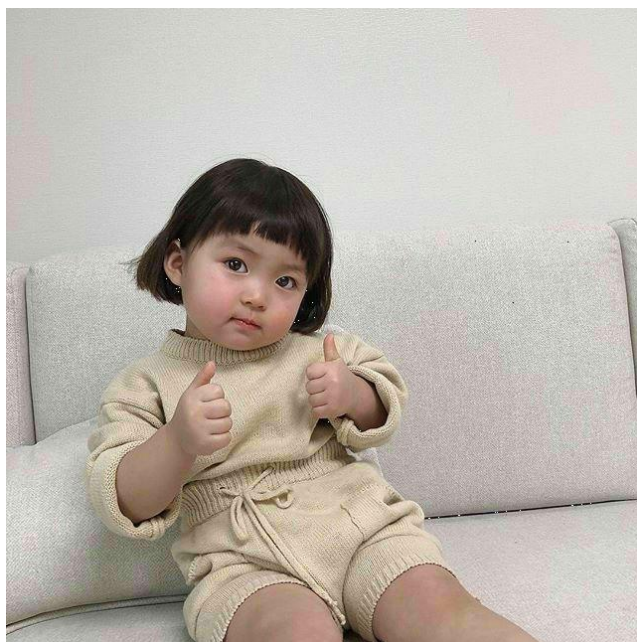
Origin	2 x 2
	
4 x 4	8 x 8
	

📌 放大圖片，可以看到女孩頭髮耳朵附近、沙發縫隙有失真狀況，在這張圖的表現上，4 x 4 跟 2 x 2 差不多，8 x 8 比較好

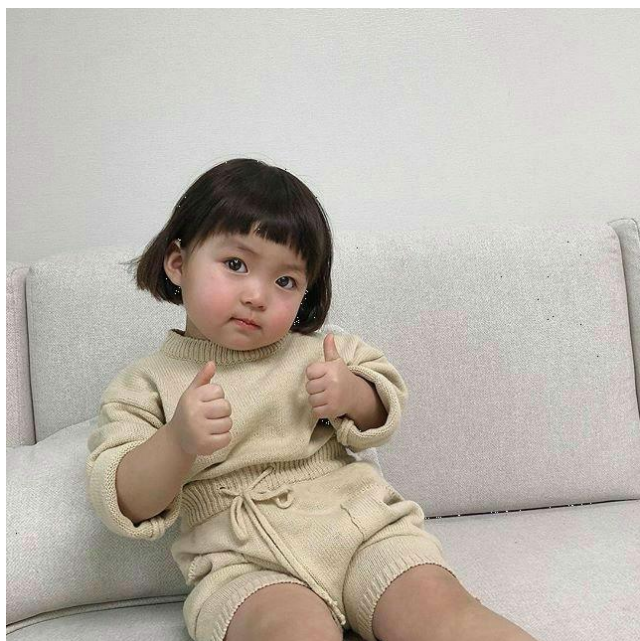
Origin



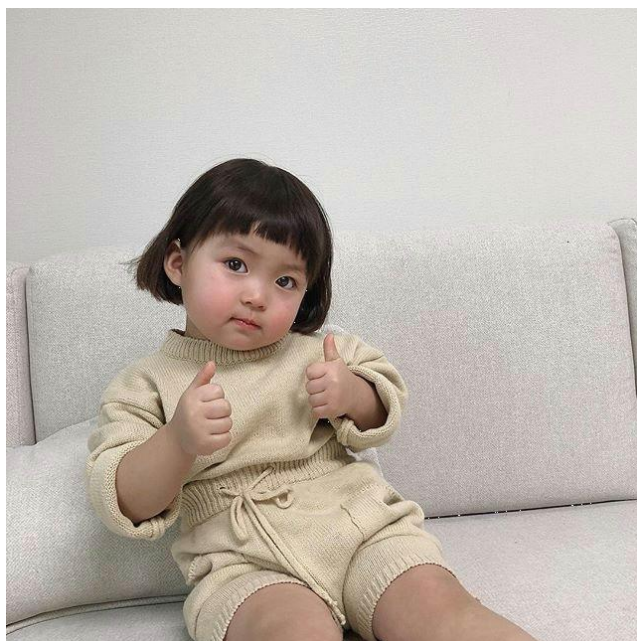
2 x 2



4 x 4




















8 x 8





無明顯失真

這邊列一些看不太出來有失真的影像（實際上仍有可能有失真的）

Origin	2 x 2	4 x 4	8 x 8
			
			
			
			
			

以上，總共 11 張圖片進行 encoding 後 decoding 的結果與原圖片比較

謝謝 老師 與 助教 這學期的用心教學，祝大家新年快樂！

## Rest of the code

參考自 [ref](#) 有修改一些讀檔寫檔以及上述有提到的內容，這邊以 2 x 2 為例子放程式碼

### Encoding `encoder.py`

- import

```
import argparse
import os
import math
import numpy as np
from utils import *
from scipy import fftpack
from PIL import Image
from huffman import HuffmanTree
```

- Customize load\_quantization\_table function for 2 x 2

```
def load_quantization_table2(component):

    # for block_size 2x2

    # Quantization Table for: Photoshop - (Save For Web 080)

    # 亮度通道的量化表
    if component == 'lum':
        q = np.array([[2, 2],
                      [3, 4]])
    # 色度通道的量化表
    elif component == 'chrom':
        q = np.array([[11, 14],
                      [14, 12]])
    else:
        raise ValueError((
            "component should be either 'lum' or 'chrom', "
            "but '{comp}' was found").format(comp=component))
    return q
```

- Quantization

```
def quantize(block, component):
    q = load_quantization_table2(component)
    return (block / q).round().astype(np.int32)
```

- Block to Zigzag

```
def block_to_zigzag(block):
    return np.array([block[point] for point in zigzag_points(*block.shape)])
```

- DCT ( Discrete Cosine Transform )



```
def dct_2d(image):
    return fftpack.dct(fftpack.dct(image.T, norm='ortho').T, norm='ortho')
```

- Encoding function

```
def run_length_encode(arr):
    # determine where the sequence is ending prematurely
    last_nonzero = -1
    for i, elem in enumerate(arr):
        if elem != 0:
            last_nonzero = i

    # each symbol is a (RUNLENGTH, SIZE) tuple
    symbols = []

    # values are binary representations of array elements using SIZE bits
    values = []

    run_length = 0

    for i, elem in enumerate(arr):
        if i > last_nonzero:
            symbols.append((0, 0))
            values.append(int_to_binstr(0))
            break
        elif elem == 0 and run_length < 15:
            run_length += 1
        else:
            size = bits_required(elem)
            symbols.append((run_length, size))
            values.append(int_to_binstr(elem))
            run_length = 0
    return symbols, values
```

- function for writing file

```
def write_to_file(filepath, dc, ac, blocks_count, tables):
    try:
        f = open(filepath, 'w')
    except FileNotFoundError as e:
        raise FileNotFoundError(
            "No such directory: {}".format(
                os.path.dirname(filepath))) from e

    for table_name in ['dc_y', 'ac_y', 'dc_c', 'ac_c']:

        # 16 bits for 'table_size'
        f.write(uint_to_binstr(len(tables[table_name]), 16))

        for key, value in tables[table_name].items():
            if table_name in {'dc_y', 'dc_c'}:
                # 4 bits for the 'category'
                # 4 bits for 'code_length'
```

```

        # 'code_length' bits for 'huffman_code'
        f.write(uint_to_binstr(key, 4))
        f.write(uint_to_binstr(len(value), 4))
        f.write(value)
    else:
        # 4 bits for 'run_length'
        # 4 bits for 'size'
        # 8 bits for 'code_length'
        # 'code_length' bits for 'huffman_code'
        f.write(uint_to_binstr(key[0], 4))
        f.write(uint_to_binstr(key[1], 4))
        f.write(uint_to_binstr(len(value), 8))
        f.write(value)

# 32 bits for 'blocks_count'
f.write(uint_to_binstr(blocks_count, 32))

for b in range(blocks_count):
    for c in range(3):
        category = bits_required(dc[b, c])
        symbols, values = run_length_encode(ac[b, :, c])

        dc_table = tables['dc_y'] if c == 0 else tables['dc_c']
        ac_table = tables['ac_y'] if c == 0 else tables['ac_c']

        f.write(dc_table[category])
        f.write(int_to_binstr(dc[b, c]))

        for i in range(len(symbols)):
            f.write(ac_table[tuple(symbols[i])])
            f.write(values[i])
f.close()

```

- main function

```

def main():
    block_size_num = 2

    parser = argparse.ArgumentParser()
    parser.add_argument("input", help="path to the input image file")
    args = parser.parse_args()

    input_file = "./origin_img/" + args.input + ".jpeg"
    output_file = "./after_encoding/" + args.input + ".txt"

    image = Image.open(input_file)
    ycbcr = image.convert('YCbCr')

    npmat = np.array(ycbcr, dtype=np.uint8)

    rows, cols = npmat.shape[0], npmat.shape[1]

    # block size: block_size_num x block_size_num
    if rows % block_size_num == cols % block_size_num == 0:

```

```

        blocks_count = rows // block_size_num * cols // block_size_num
    else:
        raise ValueError(("the width and height of the image "
                           "should both be multiples of 8"))

    # dc is the top-left cell of the block, ac are all the other cells
    dc = np.empty((blocks_count, 3), dtype=np.int32)
    ac = np.empty((blocks_count, block_size_num*block_size_num-1, 3), dtype=np.int32)

    for i in range(0, rows, block_size_num):
        for j in range(0, cols, block_size_num):
            try:
                block_index += 1
            except NameError:
                block_index = 0

            for k in range(3):
                # split block_size_num x block_size_num block and center the data range on zero
                # [0, 255] --> [-128, 127]
                block = npmat[i:i+block_size_num, j:j+block_size_num, k] - 128

                dct_matrix = dct_2d(block)
                quant_matrix = quantize(dct_matrix,
                                         'lum' if k == 0 else 'chrom')
                zz = block_to_zigzag(quant_matrix)

                dc[block_index, k] = zz[0]
                ac[block_index, :, k] = zz[1:]

    H_DC_Y = HuffmanTree(np.vectorize(bits_required)(dc[:, 0]))
    H_DC_C = HuffmanTree(np.vectorize(bits_required)(dc[:, 1:].flat))
    H_AC_Y = HuffmanTree(
        flatten(run_length_encode(ac[i, :, 0])[0]
                for i in range(blocks_count)))
    H_AC_C = HuffmanTree(
        flatten(run_length_encode(ac[i, :, j])[0]
                for i in range(blocks_count) for j in [1, 2]))

    tables = {'dc_y': H_DC_Y.value_to_bitstring_table(),
              'ac_y': H_AC_Y.value_to_bitstring_table(),
              'dc_c': H_DC_C.value_to_bitstring_table(),
              'ac_c': H_AC_C.value_to_bitstring_table()}

    write_to_file(output_file, dc, ac, blocks_count, tables)

if __name__ == "__main__":
    main()

```

## Decoding `decoder.py`

- Include 與一個全域變數 `block_size_num` (這邊是 2)

```
import argparse
import math
import os
import numpy as np
from utils import *
from scipy import fftpack
from PIL import Image

block_size_num = 2
```

- 跟 encoder 一樣的 customize `load_quantization_table` function for 2 x 2 block size

```
def load_quantization_table2(component):

    # for block_size 2x2
    # Quantization Table for: Photoshop - (Save For Web 080)

    #亮度通道的量化表
    if component == 'lum':
        q = np.array([[2, 2],
                      [3, 4]])
    #色度通道的量化表
    elif component == 'chrom':
        q = np.array([[11, 14],
                      [14, 12]])
    else:
        raise ValueError((
            "component should be either 'lum' or 'chrom', "
            "but '{comp}' was found").format(comp=component))
    return q
```

- A class for JPEG file reader

```
class JPEGFileReader:
    TABLE_SIZE_BITS = 16
    BLOCKS_COUNT_BITS = 32

    DC_CODE_LENGTH_BITS = 4
    CATEGORY_BITS = 4

    AC_CODE_LENGTH_BITS = 8
    RUN_LENGTH_BITS = 4
    SIZE_BITS = 4

    def __init__(self, filepath):
        self.__file = open(filepath, 'r')

    def read_int(self, size):
        if size == 0:
```



```

        return 0

    # the most significant bit indicates the sign of the number
    bin_num = self.__read_str(size)
    if bin_num[0] == '1':
        return self.__int2(bin_num)
    else:
        return self.__int2(binstr_flip(bin_num)) * -1

def read_dc_table(self):
    table = dict()

    table_size = self.__read_uint(self.TABLE_SIZE_BITS)
    for _ in range(table_size):
        category = self.__read_uint(self.CATEGORY_BITS)
        code_length = self.__read_uint(self.DC_CODE_LENGTH_BITS)
        code = self.__read_str(code_length)
        table[code] = category
    return table

def read_ac_table(self):
    table = dict()

    table_size = self.__read_uint(self.TABLE_SIZE_BITS)
    for _ in range(table_size):
        run_length = self.__read_uint(self.RUN_LENGTH_BITS)
        size = self.__read_uint(self.SIZE_BITS)
        code_length = self.__read_uint(self.AC_CODE_LENGTH_BITS)
        code = self.__read_str(code_length)
        table[code] = (run_length, size)
    return table

def read_blocks_count(self):
    return self.__read_uint(self.BLOCKS_COUNT_BITS)

def read_huffman_code(self, table):
    prefix = ''
    # TODO: break the loop if __read_char is not returning new char
    while prefix not in table:
        prefix += self.__read_char()
    return table[prefix]

def __read_uint(self, size):
    if size <= 0:
        raise ValueError("size of unsigned int should be greater than 0")
    return self.__int2(self.__read_str(size))

def __read_str(self, length):
    return self.__file.read(length)

def __read_char(self):
    return self.__read_str(1)

def __int2(self, bin_num):
    return int(bin_num, 2)

```

- A function to read image

```
def read_image_file(filepath):
    reader = JPEGFileReader(filepath)

    tables = dict()
    for table_name in ['dc_y', 'ac_y', 'dc_c', 'ac_c']:
        if 'dc' in table_name:
            tables[table_name] = reader.read_dc_table()
        else:
            tables[table_name] = reader.read_ac_table()

    blocks_count = reader.read_blocks_count()

    dc = np.empty((blocks_count, 3), dtype=np.int32)
    ac = np.empty((blocks_count, block_size_num*block_size_num-1, 3), dtype=np.int32)

    for block_index in range(blocks_count):
        for component in range(3):
            dc_table = tables['dc_y'] if component == 0 else tables['dc_c']
            ac_table = tables['ac_y'] if component == 0 else tables['ac_c']

            category = reader.read_huffman_code(dc_table)
            dc[block_index, component] = reader.read_int(category)

            cells_count = 0

            # TODO: try to make reading AC coefficients better
            while cells_count < block_size_num*block_size_num-1:
                run_length, size = reader.read_huffman_code(ac_table)

                if (run_length, size) == (0, 0):
                    while cells_count < block_size_num*block_size_num-1:
                        ac[block_index, cells_count, component] = 0
                        cells_count += 1
                else:
                    for i in range(run_length):
                        ac[block_index, cells_count, component] = 0
                        cells_count += 1
                    if size == 0:
                        ac[block_index, cells_count, component] = 0
                    else:
                        value = reader.read_int(size)
                        ac[block_index, cells_count, component] = value
                        cells_count += 1

    return dc, ac, tables, blocks_count
```

- Zigzag to block

```
def zigzag_to_block(zigzag):
    # assuming that the width and the height of the block are equal
    rows = cols = int(math.sqrt(len(zigzag)))

    if rows * cols != len(zigzag):
        raise ValueError("length of zigzag should be a perfect square")

    block = np.empty((rows, cols), np.int32)

    for i, point in enumerate(zigzag_points(rows, cols)):
        block[point] = zigzag[i]

    return block
```

- Dequantize

```
def dequantize(block, component):
    q = load_quantization_table2(component)
    return block * q
```

- Inverse DCT

```
def idct_2d(image):
    return fftpack.idct(fftpack.idct(image.T, norm='ortho').T, norm='ortho')
```

- main function

```
def main():

    inputPath = "./after_encoding/"
    outputPath = "./after_decoding/"

    parser = argparse.ArgumentParser()
    parser.add_argument("input", help="path to the input image file")
    args = parser.parse_args()

    input_file = inputPath + args.input + ".txt"
    output_file = outputPath + args.input + ".jpeg"

    dc, ac, tables, blocks_count = read_image_file(input_file)

    # assuming that the block is a block_size_num x block_size_num square
    block_side = block_size_num

    # assuming that the image height and width are equal
    image_side = int(math.sqrt(blocks_count)) * block_side

    blocks_per_line = image_side // block_side

    npmat = np.empty((image_side, image_side, 3), dtype=np.uint8)
```

```

for block_index in range(blocks_count):
    i = block_index // blocks_per_line * block_side
    j = block_index % blocks_per_line * block_side

    for c in range(3):
        zigzag = [dc[block_index, c]] + list(ac[block_index, :, c])
        quant_matrix = zigzag_to_block(zigzag)
        dct_matrix = dequantize(quant_matrix, 'lum' if c == 0 else 'chrom')
        block = idct_2d(dct_matrix)
        npmat[i:i+block_size_num, j:j+block_size_num, c] = block + 128

image = Image.fromarray(npmat, 'YCbCr')
image = image.convert('RGB')
image.save(output_file)
#image.show()

if __name__ == "__main__":
    main()

```

## References

---

- <https://www.impulseadventure.com/photo/jpeg-quantization.html>
- <https://www.impulseadventure.com/photo/jpeg-quantization-lookup.html?src1=10334>
- <https://github.com/ghallak/jpeg-python>
- <https://yasoob.me/posts/understanding-and-writing-jpeg-decoder-in-python/>
- <https://www.wenyanet.com/opensource/zh/5fe9e2bbf290af082a592576.html>