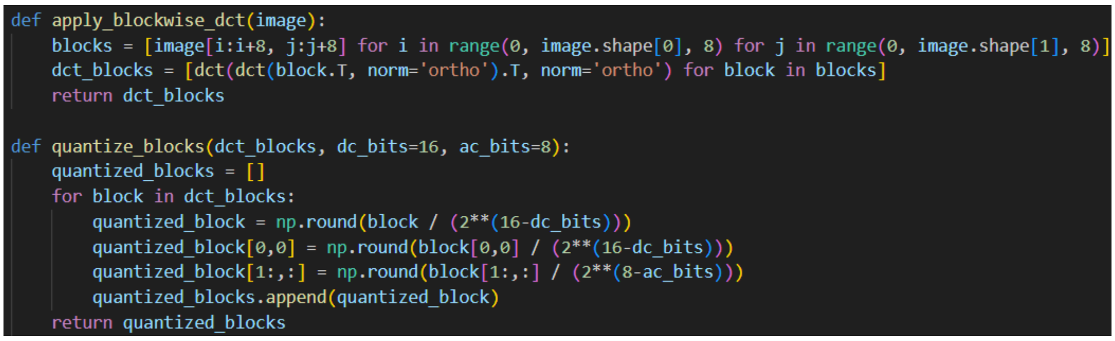
VC HW4 Entropy Coding

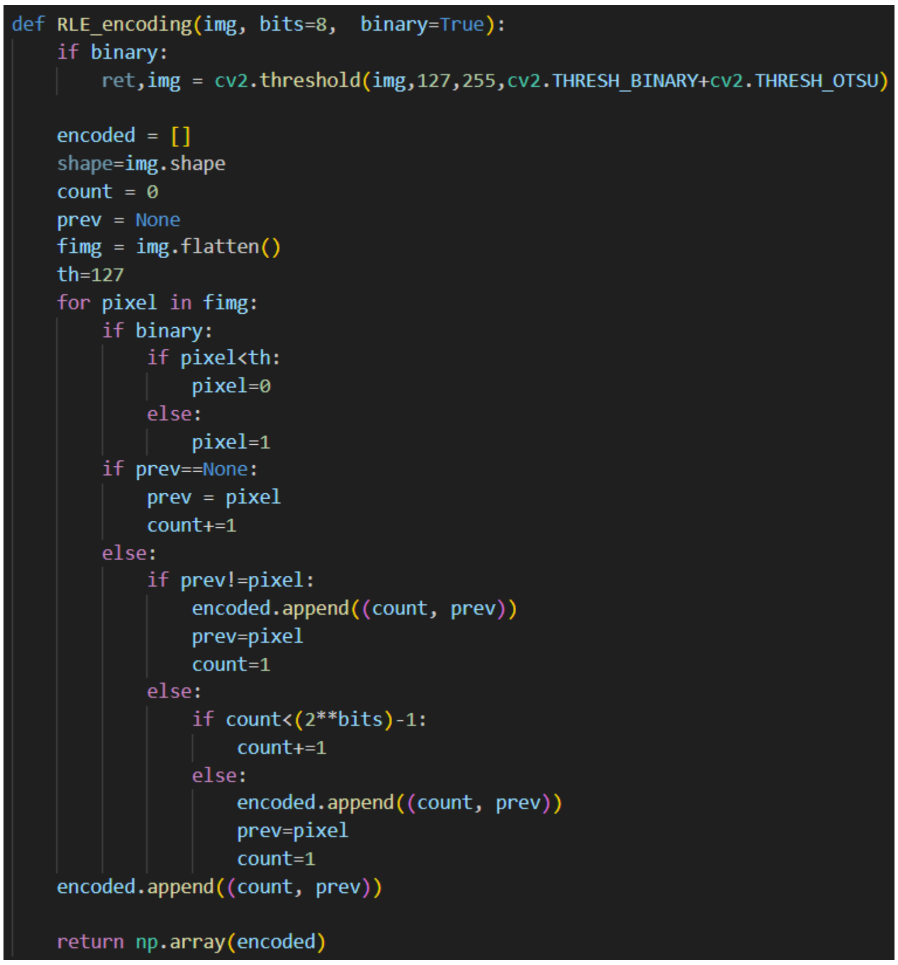
1. Introduction

- Run length encoding and run length decoding
  - 8x8 block-based DCT coefficients of "lena.png."
  - Quantize the coefficients with 16-bit for DC and 8-bit for AC.
  - Use a raster scan to visit all 8x8 blocks in these images.
  - Do the run length encoding by using a zigzag scan to visit all pixels in one block.
  - Do the run length decoding and IDCT to recover the image.

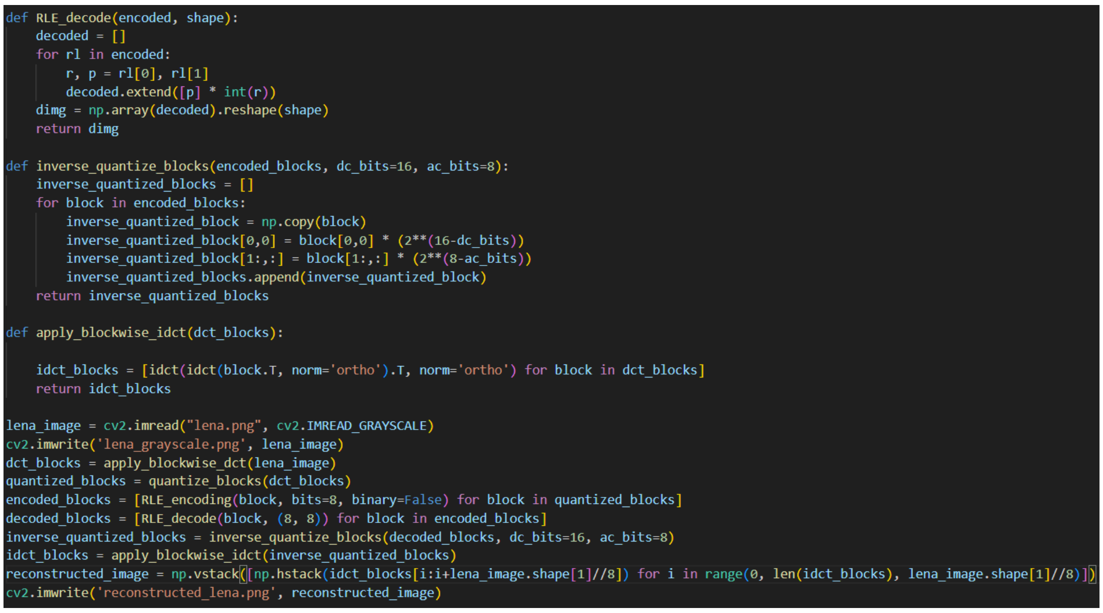
2. Brief introduction about my implementation



```python
def apply_blockwise_dct(image):
    blocks = [image[i:i+8, j:j+8] for i in range(0, image.shape[0], 8) for j in range(0, image.shape[1], 8)]
    dct_blocks = [dct(dct(block.T, norm='ortho').T, norm='ortho') for block in blocks]
    return dct_blocks

def quantize_blocks(dct_blocks, dc_bits=16, ac_bits=8):
    quantized_blocks = []
    for block in dct_blocks:
        quantized_block = np.round(block / (2**(16-dc_bits)))
        quantized_block[0,0] = np.round(block[0,0] / (2**(16-dc_bits)))
        quantized_block[1:,:] = np.round(block[1:,:] / (2**(8-ac_bits)))
        quantized_blocks.append(quantized_block)
    return quantized_blocks
```



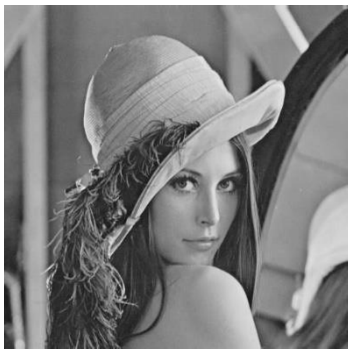
```python
def RLE_encoding(img, bits=8,  binary=True):
    if binary:
        ret,img = cv2.threshold(img,127,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)

    encoded = []
    shape=img.shape
    count = 0
    prev = None
    fimg = img.flatten()
    th=127
    for pixel in fimg:
        if binary:
            if pixel<th:
                pixel=0
            else:
                pixel=1
        if prev==None:
            prev = pixel
            count+=1
        else:
            if prev!=pixel:
                encoded.append((count, prev))
                prev=pixel
                count=1
            else:
                if count<(2**bits)-1:
                    count+=1
                else:
                    encoded.append((count, prev))
                    prev=pixel
                    count=1
    encoded.append((count, prev))

    return np.array(encoded)
```



```python
def RLE_decode(encoded, shape):
    decoded = []
    for rl in encoded:
        r, p = rl[0], rl[1]
        decoded.extend([p] * int(r))
    dimg = np.array(decoded).reshape(shape)
    return dimg

def inverse_quantize_blocks(encoded_blocks, dc_bits=16, ac_bits=8):
    inverse_quantized_blocks = []
    for block in encoded_blocks:
        inverse_quantized_block = np.copy(block)
        inverse_quantized_block[0,0] = block[0,0] * (2**(16-dc_bits))
        inverse_quantized_block[1:,:] = block[1:,:] * (2**(8-ac_bits))
        inverse_quantized_blocks.append(inverse_quantized_block)
    return inverse_quantized_blocks

def apply_blockwise_idct(dct_blocks):

    idct_blocks = [idct(idct(block.T, norm='ortho').T, norm='ortho') for block in dct_blocks]
    return idct_blocks

lena_image = cv2.imread("lena.png", cv2.IMREAD_GRAYSCALE)
cv2.imwrite('lena_grayscale.png', lena_image)
dct_blocks = apply_blockwise_dct(lena_image)
quantized_blocks = quantize_blocks(dct_blocks)
encoded_blocks = [RLE_encoding(block, bits=8, binary=False) for block in quantized_blocks]
decoded_blocks = [RLE_decode(block, (8, 8)) for block in encoded_blocks]
inverse_quantized_blocks = inverse_quantize_blocks(decoded_blocks, dc_bits=16, ac_bits=8)
idct_blocks = apply_blockwise_idct(inverse_quantized_blocks)
reconstructed_image = np.vstack([np.hstack(idct_blocks[i:i+lena_image.shape[1]//8]) for i in range(0, len(idct_blocks), lena_image.shape[1]//8)])
cv2.imwrite('reconstructed_lena.png', reconstructed_image)
```

首先先將 Lena.png 轉換為 grayscale，再將圖片分割成 8x8 的小塊，並對每個塊進行 DCT，再對 DCT 後的 coefficients 進行 Quantize，之後對 Quantize 後的 block 進行 run length encoding。run length encoding 是一種 lossless 的壓縮技術，它通過記錄相同值的連續出現次數來進行壓縮。encoding 完成後為了恢復到原本的 size，進行 run length decoding 跟 inverse quantize blocks，最後用 IDCT 復原圖片。

3. Experimental result

Original image	Grayscale image	Recover image
		

上面表格最右邊的圖是最後 recover 的結果。