영상처리 실습 보고서

10주차 : JPEG Compression

학번	201802170
이름	하 상 호

1. 과제의 내용

JPEG Compression을 위한 7개의 함수 완성

• Img2block, DCT, my_zigzag_scanning, DCT_inv, block2img 함수 완성

• 보고서에 원본과 결과 사진 비교 및 수행 시간 기록
• 압축 전 이미지, 압축 후 이미지, 수행 시간을 보고서에 기록
• 수행 시간이 12초보다 긴 경우 감점
• 각 단계마다 block값 변화 확인을 통해 연산이 제대로 이루어졌는가 확인

2. 과제의 해결 방법

과제의 내용을 해결하기 위해 어떠한 방법을 사용했는지, 자세하게 기술한다.

```
pos = [[0, 1, 5, 6, 14, 15, 27, 28],
       [2, 4, 7, 13, 16, 26, 29, 42],
       [3, 8, 12, 17, 25, 30, 41, 43],
       [9, 11, 18, 24, 31, 40, 44, 53],
       [10, 19, 23, 32, 39, 45, 52, 54],
       [20, 22, 33, 38, 46, 51, 55, 60],
       [21, 34, 37, 47, 50, 56, 59, 61],
       [35, 36, 48, 49, 57, 58, 62, 63]]

sort_pos = []
```

zigzag scanning을 위해 순서를 value 값을 넣어서 정한 배열을 미리 선언

append() 을 통해서 새로운 요소를 빈 배열에 추가

```
sort_pos = []
for i in range(8):
    for j in range(8):
        sort_pos.append((pos[i][j], i, j))

sort_pos.sort()
```

크기가 8 * 8 이므로 2중포문 8 - 8을 사용한다.

```
def C(w, n):
    if w == 0:
        return (1 / n) ** 0.5
    else:
        return (2 / n) ** 0.5


def C_inv(w, n=8):
    dst = np.zeros((n, n))
    for i in range(n):
        for j in range(n):
            if w[i][j] == 0:
                dst[i][j] = (1 / n) ** 0.5
            else:
                dst[i][j] = (2 / n) ** 0.5
    return dst
```

dct 함수를 사용하기 위해 C 함수를 선언
dct inverse를 사용하기 위해 C_inv를 따로 선언 해준다.
inverse에서 해당 차원의 값이 0일 때 C 함수와 동일하게 이용한다.

```python
def img2block(src, n=8):
    ######################################
    # TODO                               #
    # img2block 완성                      #
    # img를 block으로 변환하기             #
    # blocks : 각 block을 추가한 list      #
    # return value : np.array(blocks)    #
    ######################################
    (h, w) = src.shape
    blocks = []

    for i in range(h // n):
        for j in range(w // n):
            blocks.append(src[i * n:(i + 1) * n, j * n:(j + 1) * n])

    return np.array(blocks).astype(np.float32)
```

blocks 의 언더플로우를 방지하기 위해 형변환을 해준다.

DCT : 이전 실습을 그대로 사용

```python
def my_zigzag_encoding(block, block_size=8):
    ######################################
    # TODO                               #
    # my_zigzag_encoding 완성             #
    ######################################
    zigzag_value = list()
    pos = sort_pos.copy()

    while pos:
        element = pos.pop(0)
        row, col = element[1], element[2]
        zigzag_value.append(block[row][col])
    zigzag_value = np.array(zigzag_value)
    index = block_size ** 2 - 1
    while zigzag_value[index] == 0:
        if index == 0:
            break;
        index -= 1
    result = zigzag_value[0:index + 1]
    zigzag_encode = np.append(result, 'EOB')
    return zigzag_encode
```

zigzag_encoding 실행
zigzag의 값들을 list()를 통해 저장해준다.

미리 만들어준 sort_pos를 pos 변수에 copy

while 문을 통해서 pop 값이 empty 일때까지 실행 하고 해당 위치에 따라 스캐닝을 진행 해준다.

index는 맨 마지막 원소부터 탐색을 진행한다.

```
def my_zigzag_decoding(block, block_size=8):
    ######################################
    # TODO                               #
    # my_zigzag_decoding 완성            #
    ######################################

    pos = sort_pos.copy()

    zigzag_decode = np.zeros((block_size, block_size))
    decoded = np.delete(block, len(block) - 1)

    index = 0

    while index < len(decoded):
        element = pos.pop(0)
        row, col = element[1], element[2]
        zigzag_decode[row][col] = decoded[index]
        index += 1

    return zigzag_decode
```

zigzag_decoding 실행
미리 만들어준 sort_pos를 pos 변수에 copy

decoded 변수를 통해 block 의 EOB를 제거하고 decode 작업을 수행
미리 적용한 순서를 통해 decode를 수행

```python
def DCT_inv(block, n=8):
    ##################################################
    # TODO                                           #
    # DCT_inv 완성                                    #
    # DCT_inv 는 DCT와 다름.                           #
    ##################################################
    dst = np.zeros(block.shape)

    v, u = dst.shape
    y, x = np.mgrid[0:u, 0:v]

    C1 = C_inv(y, n=n)
    C2 = C_inv(x, n=n)

    for v_ in range(v):
        for u_ in range(u):
            temp = block * np.cos(((2 * u_ + 1) * x * np.pi) / (2 * n)) * np.cos(((2 * v_ + 1) * y * np.pi) / (2 * n))
            dst[v_, u_] = np.sum(C1 * C2 * temp)
    dst = np.clip(dst, -128, 127)
    return np.round(dst)
```

DCT_inv 함수는 C_inv를 통해 작업한다.
mgrid를 통해 x ,y 차원을 나누어 주고 , c1, c2 변수를 선언하여 C_inv 함수를 적용한다.
그 후 dct 작업을 해주고, dst 에 C1 * C2 * temp 변수를 더해서 넣어준다.

dst = np.clip(dst, -128, 127)를 한 이유는 원래는 0 ~ 255이지만 이후 과정 중에 +128을 해주므로 -128 ~ 127

```python
def block2img(blocks, src_shape, n=8):
    ##################################################
    # TODO                                           #
    # block2img 완성                                  #
    # 복구한 block들을 image로 만들기                    #
    ##################################################
    (h, w) = src_shape
    p_h = h
    p_w = w
    if h % n != 0:
        p_h = h + (n - h % n)
    if w % n != 0:
        p_w = w + (n - w % n)
    dst = np.zeros((p_h, p_w))
    index = 0
    for i in range(p_h // n):
        for j in range(p_w // n):
            dst[i * n:(i + 1) * n, j * n:(j + 1) * n] = blocks[index]
            index += 1
    return dst[:h, :w].astype(np.uint8)
```

복구된 block 들을 다시 image 로 바꾸어준다. 크기를 8fh 맞추어 주고 dst 에 넣어준다.
return 할 때 원래 크기로 되돌리기 위해 형변환을 해준다.

3. 결과물

```
block =
 [[ 87  95  92  73  59  57  57  55]
 [ 74  71  68  59  54  54  51  57]
 [ 64  58  57  55  58  65  66  65]
 [ 57  63  68  66  74  89  98 104]
 [ 95 109 117 114 119 134 145 140]
 [128 139 146 139 140 148 151 143]
 [137 135 125 118 137 156 154 132]
 [122 119 113 110 128 144 140 142]]
b =
 [[-41. -33. -36. -55. -69. -71. -71. -73.]
 [-54. -57. -60. -69. -74. -74. -77. -71.]
 [-64. -70. -71. -73. -70. -63. -62. -63.]
 [-71. -65. -60. -62. -54. -39. -30. -24.]
 [-33. -19. -11. -14.  -9.   6.  17.  12.]
 [  0.  11.  18.  11.  12.  20.  23.  15.]
 [  9.   7.  -3. -10.   9.  28.  26.   4.]
 [ -6.  -9. -15. -18.   0.  16.  12.  14.]]
```

```
bd =
 [[-225.  -31.   17.    6.  -22.   -2.    4.    2.]
 [-242.   52.    1.  -21.    8.    2.    1.   -1.]
 [  -3.   51.    5.    9.    2.   -4.    2.   -1.]
 [ 103.   23.  -12.  -13.  -11.    3.   -4.    1.]
 [  -2.  -21.    4.  -10.   -1.   -2.    0.   -1.]
 [ -13.    2.    3.    8.   -5.    1.   -2.    1.]
 [   7.    8.   -5.   -7.    2.    1.    1.   -0.]
 [  11.    0.   -0.    8.   -4.    2.   -2.    1.]]
bq =
 [[-14.  -3.   2.   0.  -1.  -0.   0.   0.]
 [-20.   4.   0.  -1.   0.   0.   0.  -0.]
 [ -0.   4.   0.   0.   0.  -0.   0.  -0.]
 [  7.   1.  -1.  -0.  -0.   0.  -0.   0.]
 [ -0.  -1.   0.  -0.  -0.  -0.   0.  -0.]
 [ -1.   0.   0.   0.  -0.   0.  -0.   0.]
 [  0.   0.  -0.  -0.   0.   0.   0.  -0.]
 [  0.   0.  -0.   0.  -0.   0.  -0.   0.]]
```

```
<start Decoding>
bq2 =
 [[-224.  -33.   20.    0.  -24.   -0.    0.    0.]
 [-240.   48.    0.  -19.    0.    0.    0.   -0.]
 [  -0.   52.    0.    0.    0.   -0.    0.   -0.]
 [  98.   17.  -22.   -0.   -0.    0.   -0.    0.]
 [  -0.  -22.    0.   -0.   -0.   -0.    0.   -0.]
 [ -24.    0.    0.    0.   -0.    0.   -0.    0.]
 [   0.    0.   -0.   -0.    0.    0.    0.   -0.]
 [   0.    0.   -0.    0.   -0.    0.   -0.    0.]]
bd2 =
 [[-48. -39. -40. -54. -66. -69. -70. -75.]
 [-53. -47. -51. -65. -74. -72. -69. -70.]
 [-71. -66. -70. -80. -81. -70. -60. -58.]
 [-68. -62. -60. -62. -55. -38. -26. -24.]
 [-30. -22. -17. -15.  -5.   9.  15.  11.]
 [  2.   7.   9.   9.  15.  25.  24.  16.]
 [  3.   5.   1.  -2.   5.  16.  16.   8.]
 [ -3.  -4. -10. -14.  -4.  13.  19.  15.]]
```

```
b2 =
 [[ 80.  89.  88.  74.  62.  59.  58.  53.]
 [ 75.  81.  77.  63.  54.  56.  59.  58.]
 [ 57.  62.  58.  48.  47.  58.  68.  70.]
 [ 60.  66.  68.  66.  73.  90. 102. 104.]
 [ 98. 106. 111. 113. 123. 137. 143. 139.]
 [130. 135. 137. 137. 143. 153. 152. 144.]
 [131. 133. 129. 126. 133. 144. 144. 136.]
 [125. 124. 118. 114. 124. 141. 147. 143.]]
scale_factor :  1 differences between original and reconstructed =
 [[  7.   6.   4.  -1.  -3.  -2.  -1.   2.]
 [ -1. -10.  -9.  -4.   0.  -2.  -8.  -1.]
 [  7.  -4.  -1.   7.  11.   7.  -2.  -5.]
 [ -3.  -3.   0.   0.   1.  -1.  -4.   0.]
 [ -3.   3.   6.   1.  -4.  -3.   2.   1.]
 [ -2.   4.   9.   2.  -3.  -5.  -1.  -1.]
 [  6.   2.  -4.  -8.   4.  12.  10.  -4.]
 [ -3.  -5.  -5.  -4.   4.   3.  -7.  -1.]]
time :  4.287600994110107
(256, 256)
```

recover img

org img