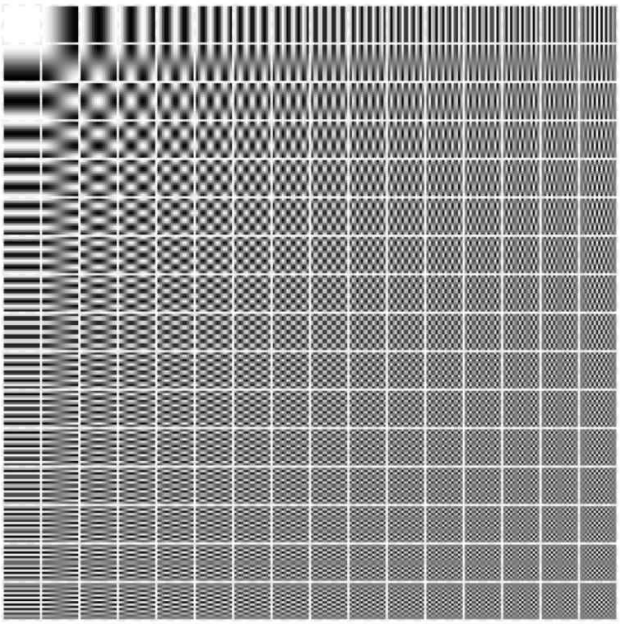
1. Use 2D DCT for 16x16 image block

■ Total 256 bases



[그림 1] 16X16 BASES

우선, 이미지를 RGB채널로 각각 분리하였습니다. 그리고 각각의 픽셀값을 double형으로 변환하였습니다.

```
1 —    I =imread('1.jpg');
2 —    red = I(:,:,1);
3 —    blue = I(:,:,2);
4 —    green = I(:,:,3);

6 —    red_img = im2double(red);
7 —    blue_img = im2double(blue);
8 —    green_img = im2double(green);
```

0.7294	0.7294	0.7294	0.7333	0.7333	0.7373
0.7333	0.7333	0.7373	0.7412	0.7412	0.7412
0.7373	0.7373	0.7412	0.7412	0.7412	0.7412
0.7412	0.7412	0.7412	0.7412	0.7490	0.7490
0.7412	0.7412	0.7412	0.7412	0.7490	0.7490
0.7373	0.7373	0.7451	0.7451	0.7451	0.7451

[그림 4] red channel의 픽셀값 일부분

DCT를 적용하기 위해 블록단위로 계산을 진행하였습니다.
이때, forward DCT는 다음과같은 식으로 정의될 수 있으므로 다음과 같은 코드로 실행하였습니다.

DCT

NxN residual 블록을 X라고 할 때 forward DCT 연산은 다음과 같은 행렬 곱으로 정의할 수 있습니다. 이때 행렬 A는 NxN 크기입니다.

$$Y = AXA^T$$

비디오 압축에서 많이 사용하는 4X4 DCT (Type-2)의 경우 A는 다음과 같이 정의됩니다.

$$A = \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix}$$

$$a = \frac{1}{2}$$

$$b = \sqrt{\frac{1}{2}} cos\left(\frac{\pi}{8}\right)$$

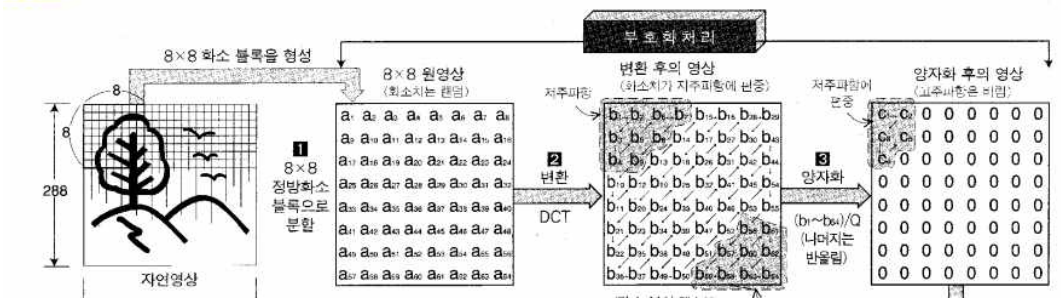
$$c = \sqrt{\frac{1}{2}} cos\left(\frac{3\pi}{8}\right)$$

```
10 —    T = dctmtx(16);
11 —    dct = @(block_struct) T * block_struct.data * T';
12
13 —    B_R = blockproc(red_img,[16 16],dct);
14 —    B_B = blockproc(blue_img,[16 16],dct);
15 —    B_G = blockproc(green_img,[16 16],dct);
```

■ Select 16 highest coefficients for each block

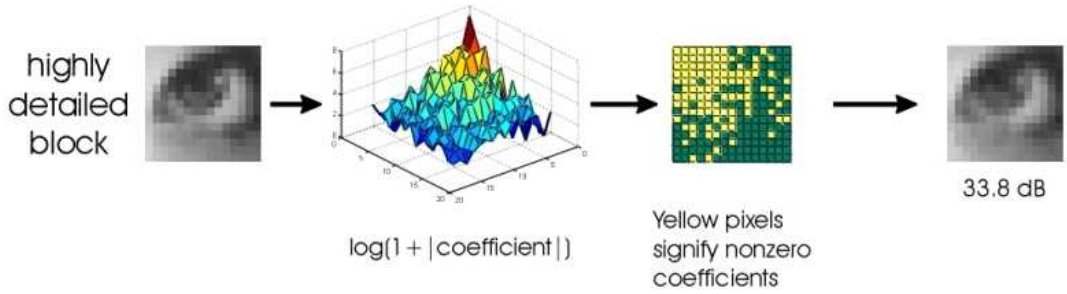
■ The other 240 coefficients are set to zero

각 픽셀을 DCT변환을 하게 되면 저주파에 편중되어 양자화를 거치면 나머지 성분들은 0이 됩니다.



[그림 7] DCT의 적용예시

하지만 실제 영상에서는 다음과같이 세밀하게 표현해야하는 경우는 저주파보다 고주파의 부분이 선택되는 경우가 있습니다. 하지만 대부분의 경우는 행렬의 좌상단에 계수들이 몰려 있으므로 다음과 같이 16개의 coefficient mask를 작성하였습니다.



[그림 8] 세밀한 표현이 필요한 영상의 coefficient그래프

```
17 —    mask = [ 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0
18               1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
19               1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
20               1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
21               1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
22               0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
23               0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
24               0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
25               0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
26               0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
27               0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
28               0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
29               0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
30               0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
31               0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
32               0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0];
```

각각의 DCT를 진행한 부분에 양자화를 위해 MASK를 씌워줬습니다.

```
34 —    B2_red = blockproc(B_R,[16 16],@(block_struct) mask .* block_struct.data);
35 —    B2_green = blockproc(B_G,[16 16],@(block_struct) mask .* block_struct.data);
36 —    B2_blue = blockproc(B_B,[16 16],@(block_struct) mask .* block_struct.data);
```

Inverse DCT with the selected coefficients and their corresponding 2D DCT bases

■ Image synthesis with 16 bases of highest coefficients

마스크를 씌운 영상(양자화를 거친)을 IDCT를 적용하여 결과를 도출해냈습니다.

IDCT

Inverse DCT는 forward DCT의 결과인 Y에 대해 다음과 같이 정의됩니다.

$$X = A^T Y A$$

여기서 다음이 성립합니다.

$$A^T A = I$$
$$A A^T = I$$

```
38 —      invdct = @(block_struct) T' * block_struct.data * T;
39
40 —      I2_red = blockproc(B2_red,[16 16],invdct);
41 —      I2_green = blockproc(B2_green,[16 16],invdct);
42 —      I2_blue = blockproc(B2_blue,[16 16],invdct);
```

```
44 —      rgbImage = cat(3, I2_red,I2_blue,I2_green);
45 —      imshow(rgbImage)
46 —      imwrite(rgbImage,'result.png');
```

Results for 3 different color images

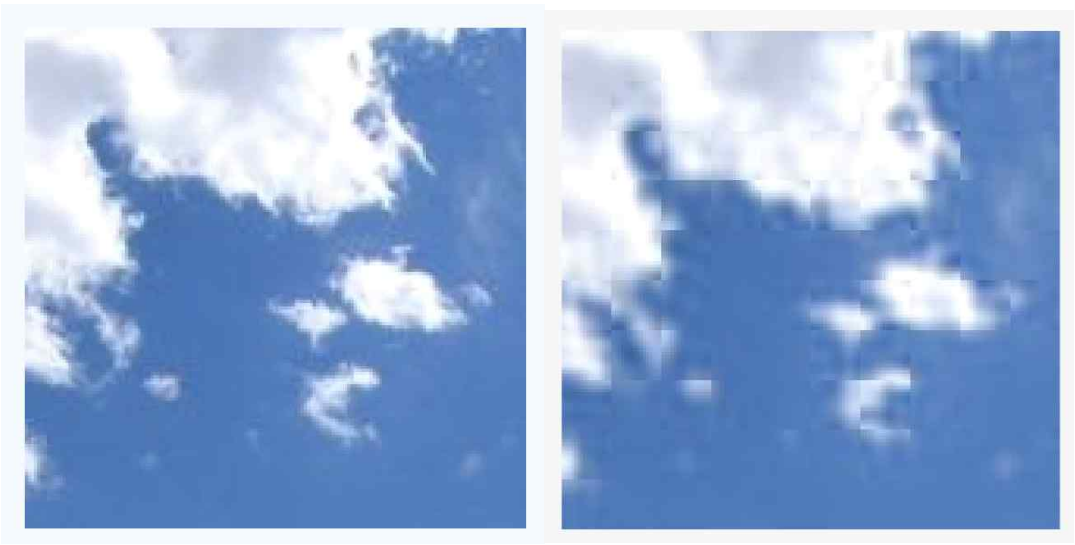
■ Perform RGB signals respectively and combine

1. 풍경사진

[그림 14] 원본 풍경 사진

[그림 15] DCT가 적용된 풍경사진

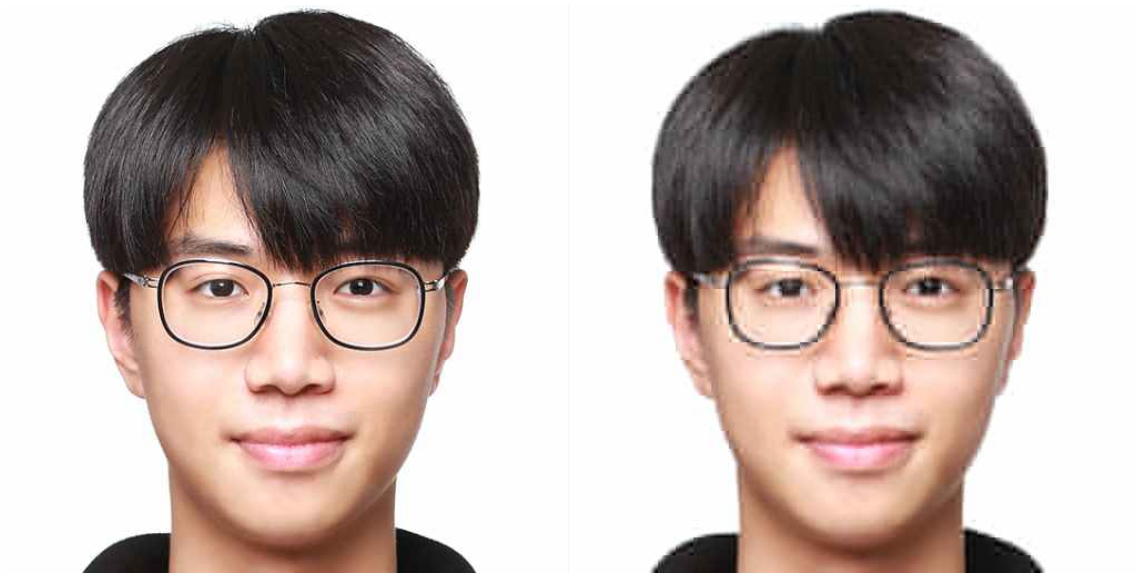
2. 160x160 풍경사진



[그림 16] 원본사진(좌), DCT를 적용시킨 사진(우)

결과를 더 자세히 보기위해 160x160 하늘사진을 적용시켜 보았습니다. 160x160의 사진이라 16x16블럭으로 나누었을 때, 총 10x10의 100칸사진이 만들어집니다. 이는 육안으로도 확인할 정도로 경계가 뚜렷했습니다. 또한, 원본사진과 비슷한 느낌이긴 하지만 세밀하게 표현된부분은 많이 줄어듦을 확인할 수 있습니다.

3. 512x512 얼굴사진



[그림 17] 원본사진(좌), DCT를 적용시킨 사진(우)

자세한 묘사가 필요한 얼굴도 깨지긴 했지만 얼굴을 인식할 수 있을정도로 표현이 됨을 확인할 수 있었습니다.