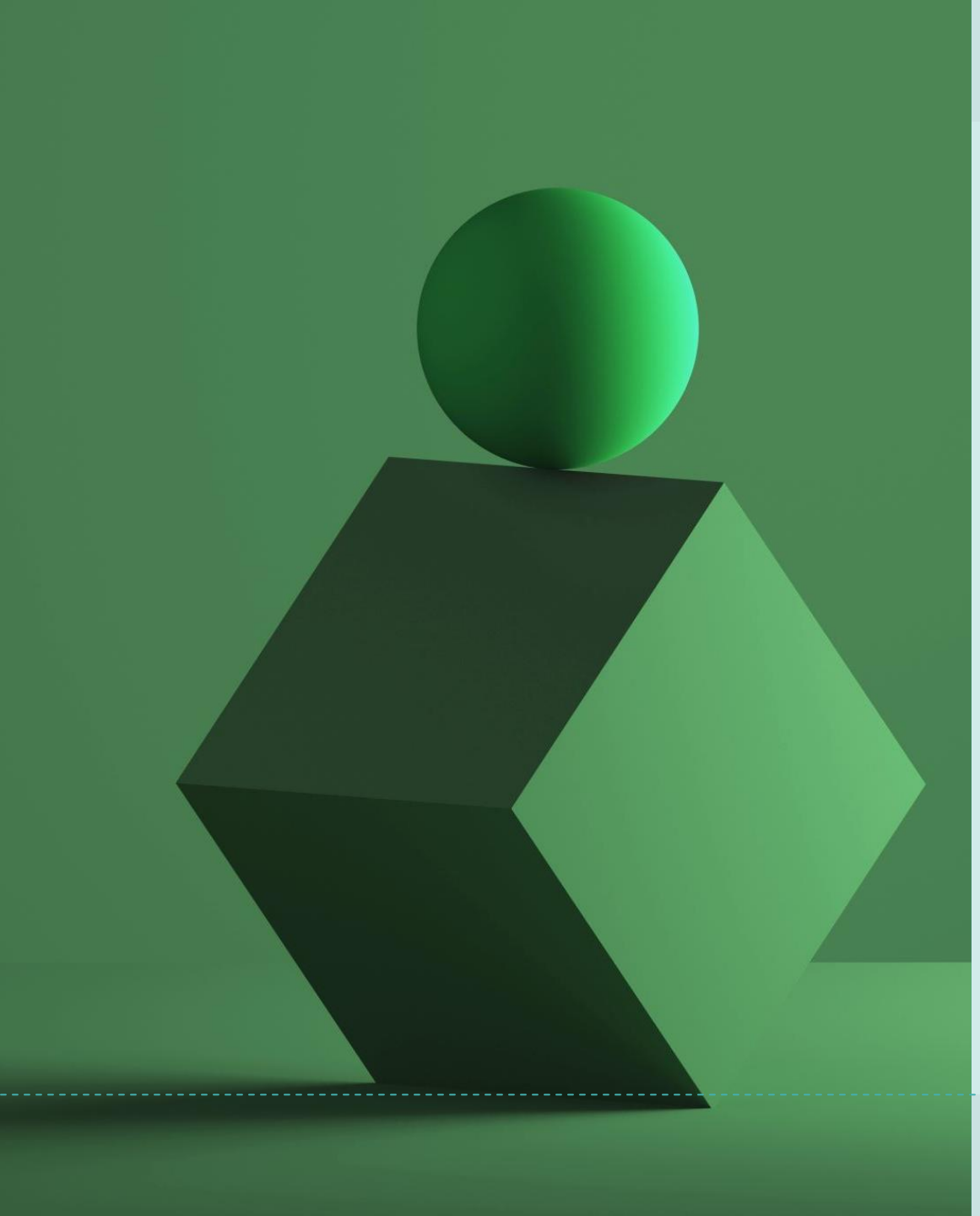


수치해석 HW9
DCT

2018008613 안상욱

1. 사진 선정

- 다음과 같은 세 개의 256 x 256 컬러 사진 3장을 선정했습니다. 16 x 16 크기의 block을 16 x 16개로 나누어서 각각 DCT를 적용해준 뒤 절댓값이 큰 16개의 coefficient를 제외하고는 0으로 설정해준 뒤 다시 IDCT를 해 주었고, 결과값을 이용해 다시 사진을 복원해 보았습니다.. 각각의 사진 이름은 picture1, picture2, picture3으로 설정했습니다.

2. C 함수 생성

- 먼저 다음과 같은 역할을 해 주는 C 함수를 만들어 주었습니다.

$$C_u = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } u = 0 \\ 1 & \text{else} \end{cases} \qquad C_v = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } v = 0 \\ 1 & \text{else} \end{cases}$$

```
def C(x):
    if x == 0:
        return 1/math.sqrt(2)
    return 1
```

3. 사진 읽어 오기

- 그 다음 cv2 모듈을 이용해 사진을 읽어왔고, 읽어온 사진을 R, G, B로 나누어서 256 * 256 크기의 배열인 SR, SG, SB를 만들었습니다. 그 후 16 * 16 크기인 블록으로 사용할 sr, sg, sb를 초기화 해주었고 DCT해준 결과를 담아줄 fr, fg, fb를 16 x 16 크기로 초기화해주었습니다. 그리고 16번째로 큰 절댓값을 찾아 줄 afr, afg, afb 배열을 초기화해주었습니다.

```
n = cv2.imread("picture1.jpg")
n = np.array(n)
an = np.zeros((256,256,3))
SR = np.zeros((256,256))
SG = np.zeros((256,256))
SB = np.zeros((256,256))

for i in range(256):
    for j in range(256):
        SR[i][j] = n[i][j][2]
        SG[i][j] = n[i][j][1]
        SB[i][j] = n[i][j][0]

sr = np.zeros((16,16))
sg = np.zeros((16,16))
sb = np.zeros((16,16))
fr = np.zeros((16,16))
fg = np.zeros((16,16))
fb = np.zeros((16,16))

afr = np.zeros(256)
afg = np.zeros(256)
afb = np.zeros(256)
```

4. DCT

- 그 후 i, j번째 block을 읽어와서 다음 식을 이용해 DCT 과정을 통해 F 배열을 구해주었습니다.

$$F_{vu} = \frac{1}{4} C_v C_u \sum_{y=0}^{N-1} \sum_{x=0}^{N-1} S_{yx} cos\left(v\pi \frac{2y+1}{2N}\right) cos\left(u\pi \frac{2x+1}{2N}\right)$$

```
for i in range(16):
    for j in range(16):
        for u in range(16):
            for v in range(16):
                sr[u][v] = SR[u + i * 16][v + j * 16]
                sg[u][v] = SG[u + i * 16][v + j * 16]
                sb[u][v] = SB[u + i * 16][v + j * 16]

        for u in range(16):
            for v in range(16):
                fr[u][v] = C(u)*C(v) / 4
                fg[u][v] = C(u)*C(v) / 4
                fb[u][v] = C(u)*C(v) / 4
                gr = 0
                gg = 0
                gb = 0
                for x in range(16):
                    for y in range(16):
                        gr += sr[x][y] * np.cos(u * math.pi * (2 * x + 1) / 32) * np.cos(v * math.pi * (2 * y + 1) / 32)
                        gg += sg[x][y] * np.cos(u * math.pi * (2 * x + 1) / 32) * np.cos(v * math.pi * (2 * y + 1) / 32)
                        gb += sb[x][y] * np.cos(u * math.pi * (2 * x + 1) / 32) * np.cos(v * math.pi * (2 * y + 1) / 32)
                fr[u][v] *= gr
                fg[u][v] *= gg
                fb[u][v] *= gb
```

5. 16개 coefficient 남기기

- 그 후 절댓값이 16번째로 큰 coefficient까지는 그대로 남겨두고 나머지 값들은 0으로 바꿔주었습니다.

```
for x in range(16):
    for y in range(16):
        afr[x * 16 + y] = abs(fr[x][y])
        afg[x * 16 + y] = abs(fg[x][y])
        afb[x * 16 + y] = abs(fb[x][y])
afr = np.sort(afr)[::-1]
afg = np.sort(afg)[::-1]
afb = np.sort(afb)[::-1]
for x in range(16):
    for y in range(16):
        if abs(fr[x][y]) < afr[15]:
            fr[x][y] = 0
        if abs(fg[x][y]) < afg[15]:
            fg[x][y] = 0
        if abs(fb[x][y]) < afb[15]:
            fb[x][y] = 0
```

6. IDCT 후 이미지 출력

- IDCT 과정을 2가지로 해 보았습니다. 먼저 0보다 작은 값은 0으로, 255보다 큰 값은 255로 바꿔주는 과정을 시행해 보았습니다.

```
        for x in range(16):
            for y in range(16):
                sr[x][y] = 0
                sg[x][y] = 0
                sb[x][y] = 0

                for u in range(16):
                    for v in range(16):
                        sr[x][y] += C(u) * C(v) * fr[u][v] * np.cos(u * math.pi * (2 * x + 1) / 32) * np.cos(v * math.pi * (2 * y + 1) / 32)
                        sg[x][y] += C(u) * C(v) * fg[u][v] * np.cos(u * math.pi * (2 * x + 1) / 32) * np.cos(v * math.pi * (2 * y + 1) / 32)
                        sb[x][y] += C(u) * C(v) * fb[u][v] * np.cos(u * math.pi * (2 * x + 1) / 32) * np.cos(v * math.pi * (2 * y + 1) / 32)

                sr[x][y] /= 4
                sg[x][y] /= 4
                sb[x][y] /= 4
                sr[x][y] = round(sr[x][y])
                sg[x][y] = round(sg[x][y])
                sb[x][y] = round(sb[x][y])
                if sr[x][y] < 0:
                    sr[x][y] = 0
                if sr[x][y] > 255:
                    sr[x][y] = 255
                if sg[x][y] < 0:
                    sg[x][y] = 0
                if sg[x][y] > 255:
                    sg[x][y] = 255
                if sb[x][y] < 0:
                    sb[x][y] = 0
                if sb[x][y] > 255:
                    sb[x][y] = 255

                SR[x + i * 16][y + j * 16] = sr[x][y]
                SG[x + i * 16][y + j * 16] = sg[x][y]
                SB[x + i * 16][y + j * 16] = sb[x][y]

for i in range(256):
    for j in range(256):
        an[i][j][2] = SR[i][j]
        an[i][j][1] = SG[i][j]
        an[i][j][0] = SB[i][j]

cv2.imwrite('p3.png', an)
```

7. 결과

- 다음과 같은 결과를 확인해 볼 수 있었습니다.

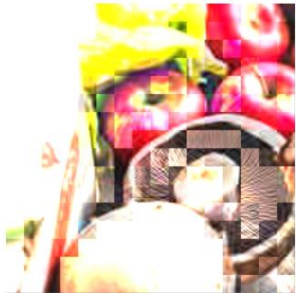
8. IDCT 후 이미지 출력

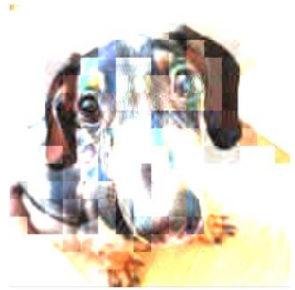
- 이번에는 min, max 값을 이용해 비율을 조정해서 0부터 255까지의 숫자를 갖도록 한 후 출력해보았습니다.

```
        for x in range(16):
            for y in range(16):
                sr[x][y] = 0
                sg[x][y] = 0
                sb[x][y] = 0

                for u in range(16):
                    for v in range(16):
                        sr[x][y] += C(u) * C(v) * fr[u][v] * np.cos(u * math.pi * (2 * x + 1) / 32) * np.cos(v * math.pi * (2 * y + 1) / 32)
                        sg[x][y] += C(u) * C(v) * fg[u][v] * np.cos(u * math.pi * (2 * x + 1) / 32) * np.cos(v * math.pi * (2 * y + 1) / 32)
                        sb[x][y] += C(u) * C(v) * fb[u][v] * np.cos(u * math.pi * (2 * x + 1) / 32) * np.cos(v * math.pi * (2 * y + 1) / 32)

                sr[x][y] /= 4
                sg[x][y] /= 4
                sb[x][y] /= 4

        max_r = np.max(sr)
        min_r = np.min(sr)
        max_g = np.max(sg)
        min_g = np.min(sg)
        max_b = np.max(sb)
        min_b = np.min(sb)

        for x in range(16):
            for y in range(16):
                if min_r < 0:
                    sr[x][y] -= min_r
                if max_r - min_r > 255:
                    sr[x][y] = sr[x][y] * 250 / (max_r - min_r)
                if min_g < 0:
                    sg[x][y] -= min_g
                if max_g - min_g > 255:
                    sg[x][y] = sg[x][y] * 250 / (max_g - min_g)
                if min_b < 0:
                    sb[x][y] -= min_b
                if max_b - min_b > 255:
                    sb[x][y] = sb[x][y] * 250 / (max_b - min_b)
                sr[x][y] = round(sr[x][y])
                sg[x][y] = round(sg[x][y])
                sb[x][y] = round(sb[x][y])

                SR[x + i * 16][y + j * 16] = sr[x][y]
                SG[x + i * 16][y + j * 16] = sg[x][y]
                SB[x + i * 16][y + j * 16] = sb[x][y]

for i in range(256):
    for j in range(256):
        an[i][j][2] = SR[i][j]
        an[i][j][1] = SG[i][j]
        an[i][j][0] = SB[i][j]

cv2.imwrite('p9.png', an)
```

9. 결과

- 다음과 같은 결과를 확인해 볼 수 있었습니다.





10. 결과 분석

- 복원한 결과를 0보다 작은 값은 0으로 255보다 큰 값은 255로 조정한 후 사진으로 출력한 결과 빛이 비치는 것 같은 결과가 나옴을 확인해 볼 수 있었습니다. 16개 외의 coefficient가 영향을 많이 주는 block에서는 값의 변화가 커서 원래 값을 온전히 복원하지 못해 원래 색상을 표현하지 못하는 block들이 꽤 많았습니다. 그렇지만 block 간의 오차는 적어서 매끄럽게 이어지는 사진 형태를 가짐을 알 수 있었습니다.
- 그에 비해 min, max 값을 이용해 전체적인 비율을 조정해서 0에서 255사이의 숫자로 바꾸어 주었을 때는 block의 형태가 눈에 띄는 점을 확인해 볼 수 있었습니다. min, max를 이용해 비율을 조정함으로써 전체적인 색상을 비율에 따라 잘 표시해 위의 방법보다 색상을 표현하지 못한 block이 적었지만, 각 block마다 비율을 이용해 색상을 조정했기 때문에 block간의 색상 차이가 나 부자연스러운 사진 형태를 가짐을 확인할 수 있었습니다.