Homework 4. CAMERA ISP & JPEG DEVELOPMENT

0. Process

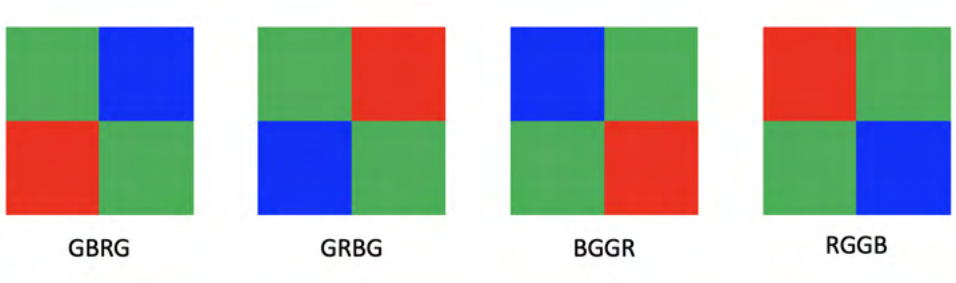
본 과제에서는 Raw 파일을 JPEG 이미지로 변환하는 ISP 파이프라인을 구현하였다. 요구사항으로 제시된 White balance와 CFA Demosaicing, Gamma correction을 구현하였고 추가적으로 Sharpening과 색감 보정을 위한 Saturation Correction, 대비 보정을 위한 Contrast correction을 구현했다.

순서는 다음과 같다.

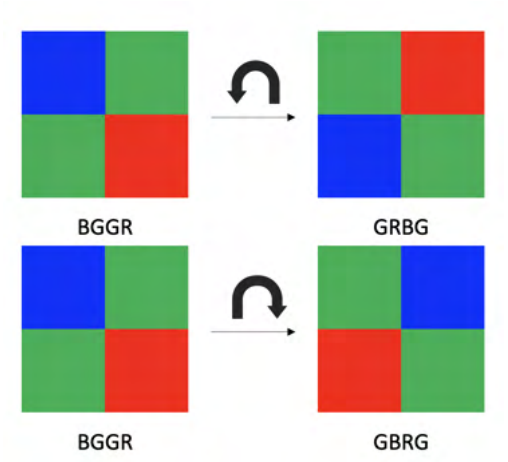
RAW file(tiff format) ⇒ White balance ⇒ CFA Demoasicing ⇒ Sharpening ⇒ Saturation Correction ⇒ Gamma correction ⇒ Contrast correction ⇒ JPEG Compression

```
if __name__ == '__main__':
    tiff_path = sys.argv[1]
    img = cv2.imread(tiff_path, cv2.IMREAD_UNCHANGED)
    img = img / (pow(2,16) - 1)
    cfa_type = get_cfa_type(tiff_path)
    balanced_img = white_balance(img, cfa_type)
    interpolated_img = interpolation(balanced_img, cfa_type)
    interpolated_img = (interpolated_img * 255).astype(np.uint8)
    sharp_img = unsharp(interpolated_img, 1)
    color_enhanced_img = saturation(sharp_img)
    result_img = gamma_correction(color_enhanced_img)
    result_img = contrast(result_img, 1.2, 2)
    filename = tiff_path.split('.')[0]
    cv2.imwrite(f'{filename}.jpg', result_img, [int(cv2.IMWRITE_JPEG_QUALITY), 95])
```

- **Image Read**
  - `cv2.imread()`를 통해 이미지를 읽어온 뒤, 연산 과정에서 오버플로우가 발생하지 않도록 normalize를 해주었다. normalize를 수행하면 numpy.uint16에서 numpy.float64로 형변환되기 때문에 데이터의 손실도 일어나지 않는다.
- **CFA type**
  - CFA(Color Filter Array)를 찾기 위한 함수 `get_cfa_type()`를 정의해 사용했다.
  - exifread 라이브러리의 함수를 이용해 이미지의 metadata를 읽어오고, CFA Array의 배열을 결정한다.
  - 기본적으로 주어진 사진과 추가한 사진은 모두 베이어 필터의 형태이다. 따라서 왼쪽에서 오른쪽으로, 위에서 아래로 읽었을 때 `'GBRG'` | `'GRBG'` | `'BGGR'` | `'RGGB'` 중 하나에 해당하게 된다.



- 같은 모델이라도 핸드폰을 돌려서 촬영하게 되면 배열의 순서가 바뀐다. 예를 들어, 갤럭시 S7 edge에서 세로로 촬영하는 경우의 배열은 BGGR이지만 이를 반시계 방향으로 회전시킨다면 GRBG이 될 것이며, 시계방향으로 회전시키면 GBRG가 될 것이다. 주어진 이미지의 경우 반시계 방향으로 돌린 뒤 촬영한 이미지로 CFA는 GRBG가 된다. 따라서 카메라의 모델이 무엇인지와 가로로 촬영된 이미지인지 세로로 촬영된 이미지인지 두 조건을 함께 고려해 CFA 타입을 결정했다.




```
def get_cfa_type(tiff_path):
    tags = exifread.process_file(open(tiff_path, 'rb'))
    metadata = {f"{tag}" : f"{value}" for tag, value in tags.items()}
    try:
        isVertical = metadata['Image ImageWidth'] < metadata['Image ImageLength']
        if metadata['Image Model'] == 'SM-G935L': # galaxy S7 edge
            return 'BGGR' if isVertical else 'GRBG'
        elif metadata['Image Model'] == 'SM-G991N': # galaxy S21
            return 'GRBG'
        else:
            raise Exception('Unknown Model')
    except Exception as e:
        print('Unknown Model', e)
```

1. White balance (gray world assumption)

1.1. Implementation

```
def white_balance(img, cfa_type):
    H, W = img.shape
    b = []
```

```
    g = []
    r = []

    for h in range(H):
        for w in range(W):
            idx = (h % 2) * 2 + w % 2
            if cfa_type[idx] == 'R':
                r.append(img[h][w])
            elif cfa_type[idx] == 'G':
                g.append(img[h][w])
            elif cfa_type[idx] == 'B':
                b.append(img[h][w])

    b_avg = np.mean(b)
    g_avg = np.mean(g)
    r_avg = np.mean(r)

    r_coeff = g_avg / r_avg
    b_coeff = g_avg / b_avg
    balanced_img = img.copy()
    for h in range(H):
        for w in range(W):
            idx = (h % 2) * 2 + w % 2
            if cfa_type[idx] == 'R':
                balanced_img[h][w] = balanced_img[h][w] * r_coeff
            elif cfa_type[idx] == 'B':
                balanced_img[h][w] = balanced_img[h][w] * b_coeff
    balanced_img = np.clip(balanced_img, 0, 1)
    return balanced_img
```

- White Balance
  - 먼저 tiff 이미지를 모두 순회하며 R,G,B 값을 각각의 배열에 추가한다. 이때 CFA type을 이용해 현재 픽셀 위치의 값이 R,G,B 중 어느 것에 해당하는지 결정한다.
  - `np.mean()` 메소드를 활용해 채널별 픽셀 값의 평균을 구한다.
  - 본 과제에서는 Gray World Assumption 방식으로 화이트 밸런싱을 수행한다. Gray World 알고리즘은 세상의 평균 색이 회색(무채색)이라고 가정한다. 따라서 Scene의 광원을 이미지의 평균 RGB 값으로 계산한다. 구체적인 수식은 다음과 같다.

$$\begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} = \begin{bmatrix} G_{avg}/R_{avg} & & \\ & 1 & \\ & & G_{avg}/B_{avg} \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} G_{avg}/R_{avg} * R \\ G \\ G_{avg}/B_{avg} * B \end{bmatrix}$$

    따라서 다시 이미지 배열을 순회하며 R 채널에는 G_{avg}/R_{avg} 값을, B 채널에는 G_{avg}/B_{avg}를 곱해준다.
  - 계산 이후에는 범위를 벗어나는 값들을 다시 0과 1 사이의 값이 되도록 처리해주고 반환한다.

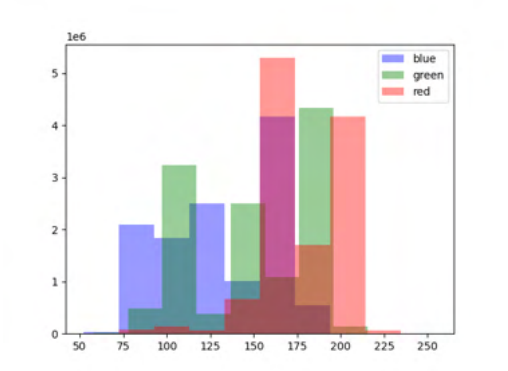
1.2. Results

- 아래 사진은 화이트 밸런스 유무에 따른 결과 차이를 보여주는 이미지이다. 왼쪽 이미지의 경우 화이트 밸런스 없이 CFA Interpolation과 Gamma Correction만을 해주었을 때 결과이고, 오른쪽은 화이트 밸런스와 CFA Interpolation, Gamma Correction을 모두 수행한 결과이다.

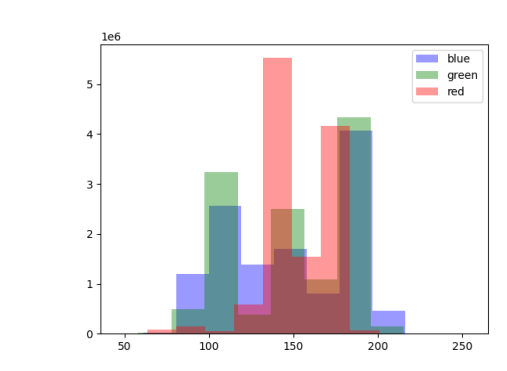
- 두 이미지의 히스토그램을 살펴보면, 화이트 밸런싱을 하지 않았을 때는 Red 채널의 분포가 오른쪽에 치우쳐 있어 intensity가 다른 채널에 비해 크다는 것을 알 수 있다. 화이트 밸런스를 해준 이미지의 히스토그램을 보면 Red 채널의 intensity가 감소하고 Blue 채널의 intensity는 비교적 증가했다. 각 채널의 intensity를 해당 채널 intensity의 평균 값으로 나누고 Green 채널의 intensity의 평균 값을 곱해주었기 때문에, 모든 채널의 intensity의 평균이 Green 채널의 intensity 평균과 같아지면서 균형이 맞춰지는 것이다.

without white balance

with white balance



histogram of above image



histogram of above image

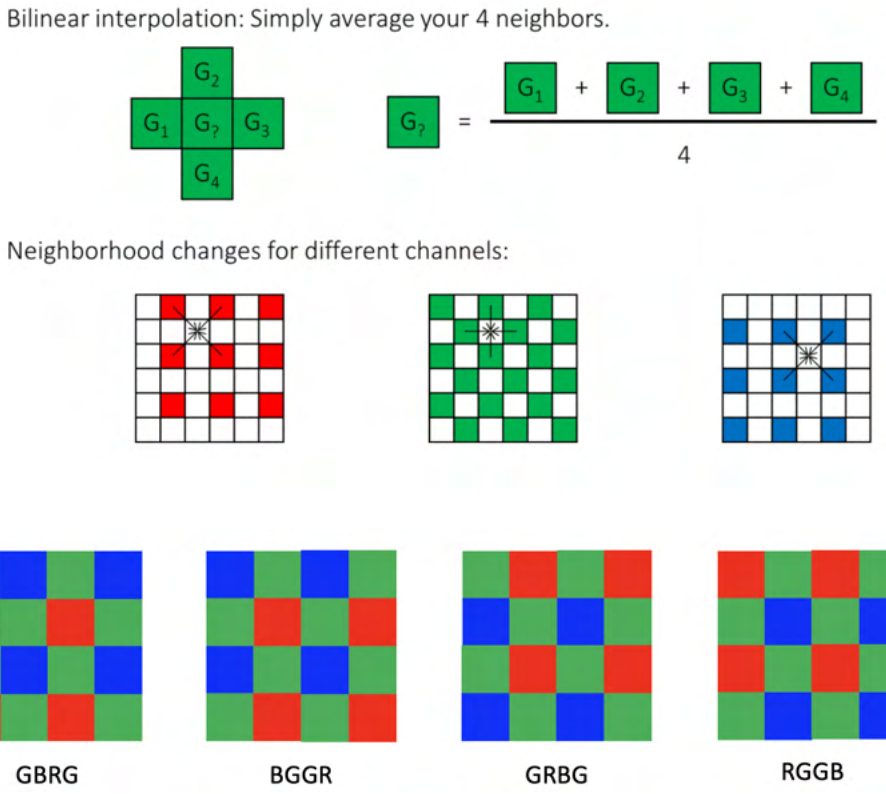
2. CFA interpolation(Demosaicing)

2.1. Implementation

```
def interpolation(img, cfa_type):
    H, W = img.shape
    interpolated_img = np.zeros((H,W,3))

    def isOut(h,w):
        return h < 0 or h >= H or w < 0 or w >= W

    def get_w(h,w):
        if w == 0:
            return img[h][w+1]
        elif w == W-1:
            return img[h][w-1]
```

```
        else:
            return np.mean([img[h][w-1], img[h][w+1]])

    def get_h(h,w):
        if h == 0:
            return img[h+1][w]
        elif h == H-1:
            return img[h-1][w]
        else:
            return np.mean([img[h+1][w], img[h-1][w]])

    def get_x(h,w):
        values = []
        if not isOut(h-1,w-1): values.append(img[h-1][w-1])
        if not isOut(h-1,w+1): values.append(img[h-1][w+1])
        if not isOut(h+1,w-1): values.append(img[h+1][w-1])
        if not isOut(h+1,w+1): values.append(img[h+1][w+1])
        return np.mean(values)

    def get_g(h,w):
        values = []
        if not isOut(h-1,w): values.append(img[h-1][w])
        if not isOut(h+1,w): values.append(img[h+1][w])
        if not isOut(h,w-1): values.append(img[h][w-1])
        if not isOut(h,w+1): values.append(img[h][w+1])
        return np.mean(values)

    for h in range(H):
        for w in range(W):
            idx = (h % 2) * 2 + w % 2
            if cfa_type[idx] == 'R':
                r = img[h][w]
                g = get_g(h,w)
                b = get_x(h,w)
                interpolated_img[h][w] = [b,g,r]
            elif cfa_type[idx] == 'G':
                w_val = get_w(h,w)
                h_val = get_h(h,w)
                if idx % 2 == 0:
                    b = w_val if cfa_type[idx + 1] == 'B' else h_val
                    r = h_val if cfa_type[idx + 1] == 'B' else w_val
                else:
                    b = w_val if cfa_type[idx - 1] == 'B' else h_val
                    r = h_val if cfa_type[idx - 1] == 'B' else w_val
                g = img[h][w]
                interpolated_img[h][w] = [b,g,r]
            elif cfa_type[idx] == 'B':
                r = get_x(h,w)
                g = get_g(h,w)
                b = img[h][w]
                interpolated_img[h][w] = [b,g,r]
    interpolated_img = np.clip(interpolated_img, 0, 1)
    return interpolated_img
```

- 앞서 찾은 CFA의 type에 따라 연산을 수행한다.
- 이미지를 순회하며 해당 위치의 채널이 무엇인지에 따라 연산을 수행한다. 보간 방식으로는 bilinear interpolation을 사용했다.



- R인 경우
  - R: 해당 픽셀의 값을 그대로 사용한다.
  - G: 현재 채널이 R이라면 인접한 G 채널은 CFA 타입에 상관없이 항상 현재 위치의 상하좌우에 위치한다. 따라서 현재 위치가 (h,w)라면 (h-1,w), (h+1,w), (h,w-1), (h,w+1) 중 이미지 영역을 벗어나지 않는 픽셀 값을 모두 읽어오고 평균을 구해 G 값으로 사용한다. 해당 연산을 하는 함수를 `get_g()`로 정의해 사용했다.
  - B: 현재 채널이 R이라면 인접한 B 채널의 상대적인 위치는 CFA 타입에 상관없이 항상 같다. 따라서 현재 위치가 (h,w)라면 (h-1,w-1), (h-1,w+1), (h+1,w-1), (h+1,w+1) 중 이미지 영역을 벗어나지 않는 픽셀 값을 모두 읽어오고 평균을 구해 B 값으로 사용한다. 해당 연산을 하는 함수를 `get_x()`로 정의해 사용했다.
- B인 경우
  - R: 현재 채널이 B라면 인접한 R 채널의 상대적인 위치는 CFA 타입에 상관없이 항상 같다. 따라서 `get_x()` 함수를 이용해 현재 위치가 (h,w)라면 (h-1,w-1), (h-1,w+1), (h+1,w-1), (h+1,w+1) 중 이미지 영역을 벗어나지 않는 픽셀 값을 모두 읽어오고 평균을 구해 R 값으로 사용한다.
  - G: 현재 채널이 B라면 인접한 G 채널은 CFA 타입에 상관없이 항상 현재 위치의 상하좌우에 위치한다. 따라서 `get_g()` 함수를 이용해 현재 위치가 (h,w)라면 (h-1,w), (h+1,w), (h,w-1), (h,w+1) 중 이미지 영역을 벗어나지 않는 픽셀 값을 모두 읽어오고 평균을 구해 G 값으로 사용한다.
  - B: 해당 픽셀의 값을 그대로 사용한다.
- G인 경우
  - R, B: 현재 채널이 G라면 인접한 R, B 채널의 상대적인 위치는 CFA 타입에 따라 달라진다. 따라서 G의 옆에 위치하는 픽셀의 채널에 따라 R값과 B값을 각각 구한다. G의 좌우에 위치한 채널이

B인 경우 B에는 좌우 값의 평균을, R에는 상하 값의 평균을 사용하고, G의 좌우에 위치한 채널이 R인 경우에는 반대로 R에는 좌우 값의 평균을, B에는 상하 값의 평균을 사용한다.

- G: 해당 픽셀의 값을 그대로 사용한다.

2.2. Results

- 아래 이미지는 tiff 파일을 읽어와 화이트 밸런싱을 수행한 뒤의 사진과, 그 이미지에 대해 CFA demosaicing을 해준 사진이다. CFA demosaicing을 하기 전에는 한 픽셀이 한 채널만의 intensity 정보를 갖고 있으며, 이를 이용해 interpolation을 해줌으로써 오른쪽과 같이 한 픽셀이 R,G,B 값을 모두 가진 컬러 이미지로 변환될 수 있다.

after white balance

after CFA demosaicing

3. Gamma correction

3.1. Implementation

```
def gamma_correction(img, gamma):
    img = img / 255.0
    result_img = img ** (1/gamma)
    result_img = (result_img * 255).astype(np.uint8)
    return result_img
```

- Gamma correction은 인간의 시각이 밝기에 대해 비선형적으로 반응하기 때문에, 이를 보정해주기 위한 과정이다. 아래와 같은 수식을 통해 보정하며, 구현은 간단하게 이미지에 1/gamma만큼 지수승을 해준 뒤 반환하는 함수를 만들어 사용했다.

$$I'(x,y) = I(x,y)^{\frac{1}{\gamma}}$$

3.2. Results

- 아래는 gamma 값에 따른 결과물이다. 위의 이미지는 화이트 밸런스와 CFA demosaicing만을 해주었을 때이고 아래 세 이미지는 다른 gamma 값에 대한 결과물이다. gamma 값이 클 수록 이미지가 밝아지는 것을 확인할 수 있다.

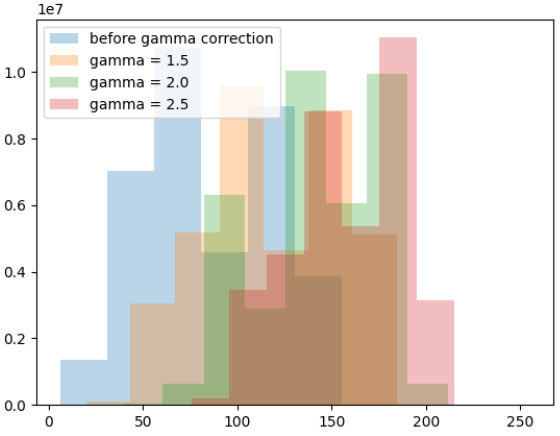
without gamma correction

gamma = 1.5

gamma = 2.0

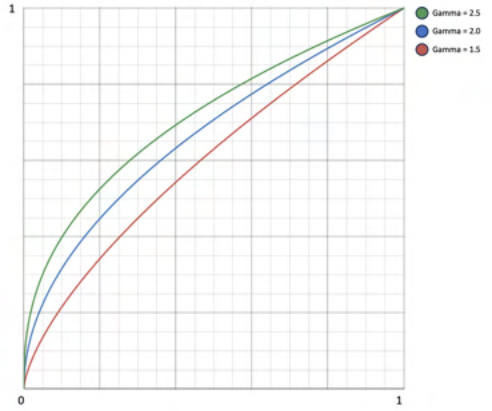
gamma = 2.5

- intensity 분포를 살펴보면 gamma correction을 하지 않았을 때가 가장 왼쪽에 치우쳐 있고, gamma 값이 커질수록 분포가 전반적으로 오른쪽으로 이동하며 intensity가 커져 이미지가 밝아지는 것을 알 수 있다.

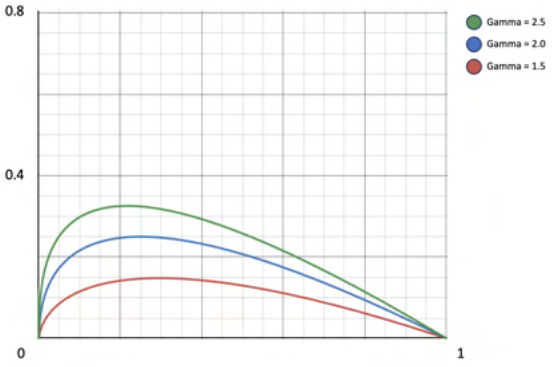


Gamma 값에 따른 히스토그램 변화(1)

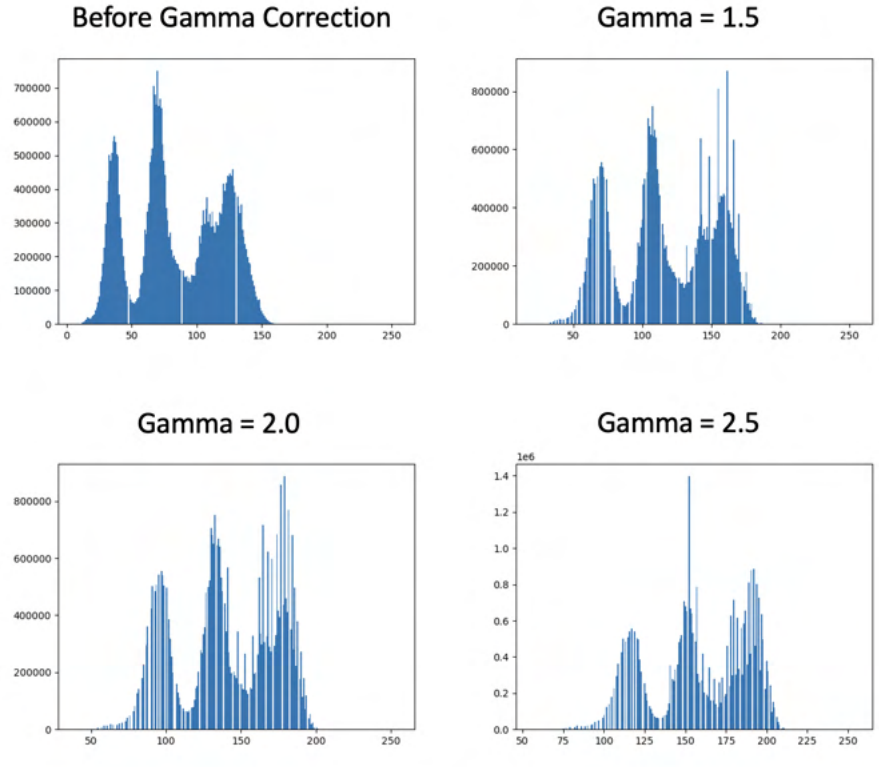
- 또한, 모든 intensity에 대해 동일한 정도로 값이 증가하는 것이 아니라, 아래와 같이 어두운 영역의 intensity증가량이 더 큰 것을 알 수 있다. 따라서 히스토그램이 모양을 유지하면서 그대로 이동하는 게 아니라 어두운 부분이 차지하는 비율이 작아지면서 오른쪽으로 이동하는 것이다.



I(x,y)^(1/gamma)의 그래프



intensity에 따라 달라지는 intensity 증가량



Gamma 값에 따른 히스토그램 변화(2)

4. Additional operations

4.1. Sharpening

4.1.1. Implementation

```
def unsharp(img, alpha):
    ycrcb_img = cv2.cvtColor(img, cv2.COLOR_BGR2YCrCb)
    y = ycrcb_img[:, :, 0].astype(np.float32)
    blurred_img = cv2.bilateralFilter(y, 9, 55, 55)
    ycrcb_img[:, :, 0] = np.clip(y + alpha * (y - blurred_img), 0, 255).astype(np.uint8)
    sharp_img = cv2.cvtColor(ycrcb_img, cv2.COLOR_YCrCb2BGR)
    return sharp_img
```

- 추가기능으로 sharpening을 구현하였다. 먼저 BGR 형태에서 YCrCb 형태로 바꾸고, y(밝기 채널)에 대해서만 sharpening을 해주었다.
- bilateralFilter를 적용해 y 채널의 high frequency를 제거하고 이를 다시 원래 y에서 빼주어 high frequency 이미지를 얻는다. unsharp 강도 조절을 위해 이 값에 alpha를 곱한 뒤 y 채널의 원래 값에 더해줌으로써 unsharp masking을 구현했다.

4.1.2. Results

- 공통적으로 화이트 밸런싱, CFA demosaicing, gamma correction을 적용해주었고, 오른쪽 이미지는 추가로 gamma correction 이전에 sharpening을 해준 이미지이다.
- 글씨 부분을 확대해서 살펴보면 경계 부분이 더 뚜렷해진 것을 육안으로 확인할 수 있다.

before

after

before / after

4.2. Saturation

4.2.1. Implementation

```
def saturation(img):
    hsv_image = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    hsv_image[:,:,1] = cv2.add(hsv_image[:,:,1], 20)
    return cv2.cvtColor(hsv_image, cv2.COLOR_HSV2BGR)
```

- 색감 조정 없이 화이트 밸런싱만을 수행한 뒤 결과물을 확인했을 때, 대체적으로 채도가 낮은 이미지가 생성되는 것을 확인했다.
- 이를 해결하기 위해 이미지의 채도를 높이고자 BRG 이미지를 HSV 형태로 변경하고 모든 픽셀에 대해 s(saturation) 채널 값만을 20만큼 증가시켜주었다. 이후 다시 BGR 채널로 변환해 반환하도록 했다.

4.2.2. Results

- 공통적으로 white balance, CFA demosaicing, sharpening, gamma correction을 적용해주었고, 오른쪽 이미지는 추가로 gamma correction 이전에 채도를 높여준 이미지이다.

before

after

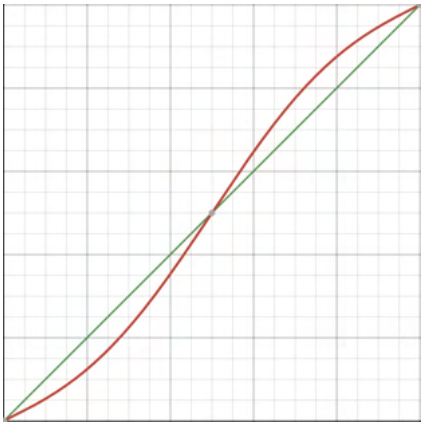
4.3. Contrast

4.3.1. Implementation

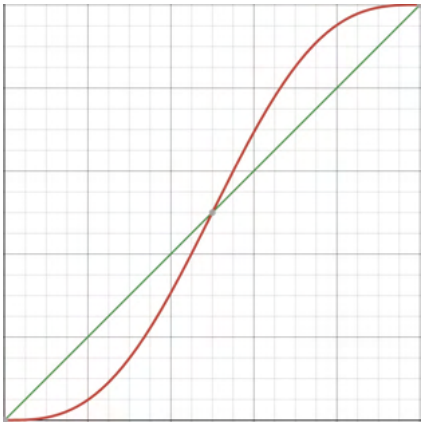
```
def contrast(img, magnitude):
    img = img / 255.0
    img = np.clip(img - magnitude * np.sin(2*(np.pi)*img) / (4 * np.pi), 0, 1)
    img = (img * 255).astype(np.uint8)
    return img
```

- gamma correction만을 해주었을 때는 갤럭시로 생성되는 jpg 이미지에 비해 대비가 약한 경향이 있었다. 따라서 이미지의 대비를 강하게 만들어주기 위해, 위와 같은 형태로 이미지의 intensity를 조절해주었다.
- 어두운 영역은 더 어둡게 만들고 밝은 영역은 더 밝게 만들 수 있도록 위의 그래프처럼, 아래의 식처럼 계산해주어 대비를 더 강하게 만들어줬다. 여기서 m은 magnitude로, magnitude 값에 따라 그래프 형태가 조금씩 달라지며, magnitude가 커지면 대비가 커진다.

$$x' = x - \frac{m}{4\pi}\sin\left(2\pi x\right)$$



magnitude = 1



magnitude = 2

4.3.2. Results

before contrast

after contrast (magnitude = 1.0)

after contrast (magnitude = 2.0)

5. JPEG Compression

```
result_img = contrast(result_img, 1.2, 2)
filename = tiff_path.split('.')[0]
cv2.imwrite(f'{filename}.jpg', result_img, [int(cv2.IMWRITE_JPEG_QUALITY), 95])
```

- JPEG Compression은 opencv의 imwrite를 이용해 jpg 파일로 저장하는 방식으로 해결했다. 품질 값(IMWRITE_JPEG_QUALITY)은 0~100 사이의 값이 될 수 있으며 임의로 95로 설정하여 저장했다.

6. Results

- **단계별 결과물**

  RAW file(tiff format) ⇒ White balance ⇒ CFA Demoasicing ⇒ Sharpening(alpha 1) ⇒ Saturation Correction ⇒ Gamma correction(gamma 2.8) ⇒ Contrast correction(magnitude 1.8) ⇒ JPEG Compression

- 20230122_141317

White balance

CFA Demoasicing

Sharpening

Saturation Correction

Gamma correction

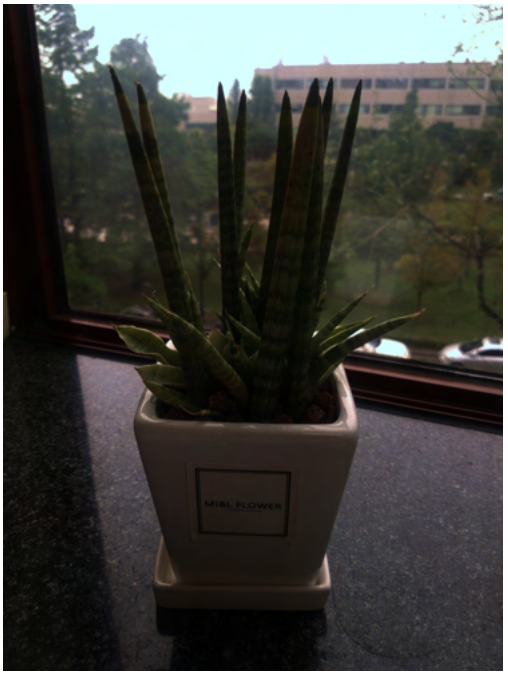
Contrast correction

- 20200917_170725

White balance

CFA Demoasicing

Sharpening



Saturation Correction

Gamma correction

Contrast correction

- 20200917_170725

White balance

CFA Demoasicing

Sharpening

Saturation Correction

Gamma correction

Contrast correction

- **최종 결과물**

20200917_170725.jpg(mine)

20200917_170725.jpg(mine)

20230122_141317.jpg(mine)

- **비교**

20200917_170725.jpg(mine)

20200917_170725.jpg(galaxy)

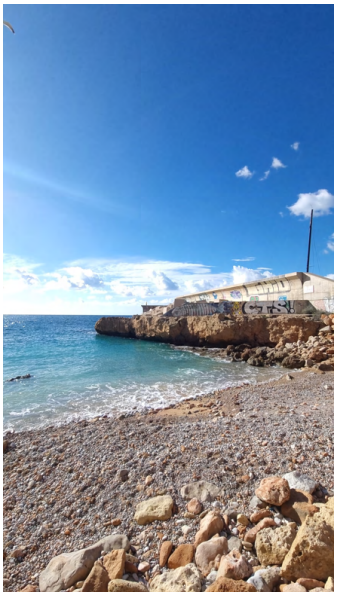
20200917_170725.jpg(photoshop)

20200917_170725.jpg(mine)

20200917_170725.jpg(galaxy)

20200917_170725.jpg(photoshop)

20230122_141317.jpg(mine)



20230122_141317.jpg(galaxy)

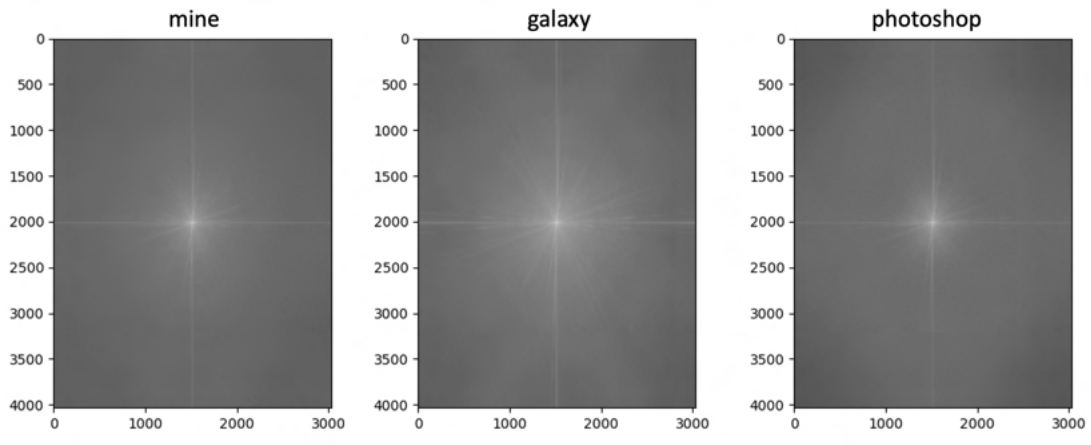
20230122_141317.jpg(photoshop)

- **edge**

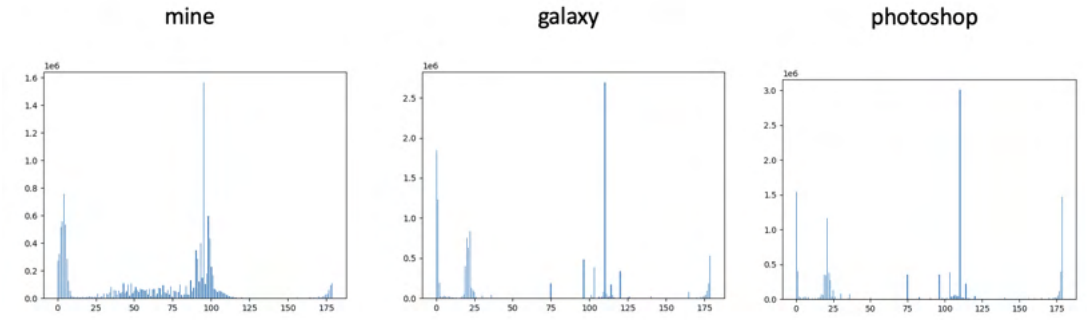
- 왼쪽부터 내 결과물, 갤럭시 결과물, 포토샵 결과물이다. 엣지 부분을 살펴보면 갤럭시로부터 생성된 이미지에서의 대비가 가장 큰 것을 확인할 수 있다.

- 또한, 세 이미지를 푸리에 변환하여 frequency domain에서의 스펙트럼을 살펴보면, 갤럭시로 생성된 이미지의 스펙트럼에서 더 많은 선들(엣지를 나타냄)이 존재하며, 선들이 뻗어있는 영역이 비교적 넓어 높은 주파수를 갖는다고 할 수 있다.

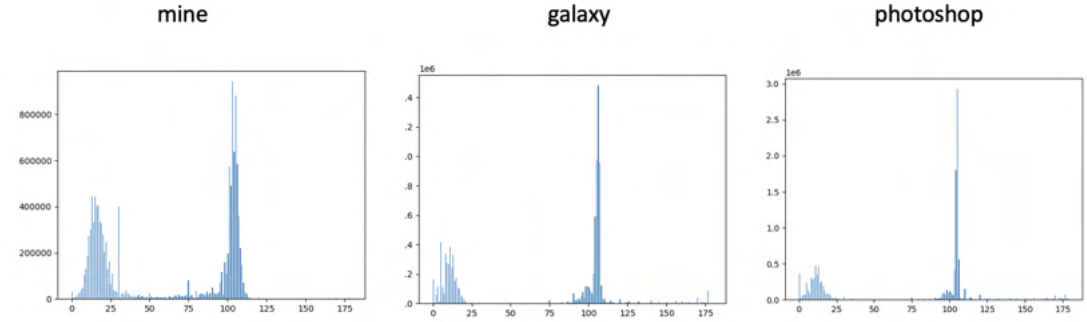
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

img = cv2.imread('20200917_170725_result.jpg', cv2.IMREAD_GRAYSCALE).astype(np.float32) / 255.
img_f = np.fft.fft2(img)
spectrum = np.log(np.abs(img_f))
plt.imshow(np.fft.fftshift(spectrum), cmap = 'gray')
plt.show()
```



- **white balance**
  - 20200917_170725.jpg, 20230122_141317.jpg 이미지의 경우 갤럭시, 포토샵으로 생성된 이미지와의 색감이 조금 다르다. 화이트 밸런싱 방법에 따라 색 온도가 달라져 색감 차이가 생겼을 것이라 추측한다.
  - 이미지의 hue 분포를 각각 히스토그램으로 나타내면 아래와 같다.



20200917_170725.jpg



20230122_141317.jpg

hue 값에 따른 색은 아래와 같다.

따라서 `20200917_170725.jpg`에서는 갤럭시와 포토샵으로 생성한 이미지에 비해 내 코드로 생성한 이미지의 hue 값들이 60~120 사이에 많이 분포되어 있어 초록빛으로 보인다.

`20230122_141317.jpg` 에서는 내 코드로 생성한 이미지의 hue 값들이 0~30 사이에 많이 분포되어 있어 비교적 붉게 보이는 것이다.

- Gray world assumption 대신 이미지의 종류에 따라 다른 방식으로 화이트 밸런싱을 해준다면 차이가 적어질 것이라 생각한다.