2.1 Luminance normalization

```
def rgb_to_ycbcr(image):
    # RGB -> YCbCr, return: Y - luminance
    R = image[:,:,0]
    G = image[:,:,1]
    B = image[:,:,2]
    Y = 0.299 * R + 0.587 * G + 0.114 * B
    Y = Y.astype(np.float64)
    return Y, R, G, B
def normalise_luminance(Y):
    # Y - luminance [0, 1] 범위로 정규화 (로그 정규화)
    epsilon = 1
    Y_normalised = np.log(Y + epsilon) / np.log(Y.max() + epsilon)
    return Y_normalised
```

2.2 Optimal gamma correction parameter estimation

Divide the histogram of the input image into two parts: dark and bright regions

```
def divide_luminance(Y):
    threshold = 0.5
    dark = Y <= threshold
    bright = Y > threshold
    # dark, bright 은 2 차원 boolean 배열. 해당부분이 True, 아닌 부분이 False 로 됨
    return dark, bright
# 밝은 영역과 어두운 영역을 각각 (3024, 4032) 형태로 유지
def create_region_masks(Y_normalised, dark, bright):
    dark_region = np.zeros_like(Y_normalised)
    bright_region = np.zeros_like(Y_normalised)
    # 어두운 부분은 Y_normalised 의 값을 유지, 나머지는 0
    dark_region[dark] = Y_normalised[dark]
    # 밝은 부분은 Y_normalised 의 값을 유지, 나머지는 0
    bright_region[bright] = Y_normalised[bright]
    return dark_region, bright_region
```

Newton Method

```
def f(gamma, region, sigma, N):
    # 최적화 위해서 0 만들어야됨
    return (1/N) * np.sum(region ** gamma) - sigma
def df(gamma, region, N):
    # f 의 도함수 정의
    epsilon = 1e-5 # 로그 0 방지
    return (1/N) * np.sum(region ** gamma * np.log(region + epsilon))
def newton_method(gamma, region, sigma, N, gamma_min, gamma_max, es=1e-7, ea=100,
iter_count=0):
    if ea <= es:
        print(f"Converged gamma: {gamma:.10f} after {iter_count} iterations.")
        return gamma

    gamma_old = gamma
    try:
        f_val = f(gamma, region, sigma, N)
        df_val = df(gamma, region, N)

        # 중간 결과 출력
        print(f"Iteration {iter_count + 1}: gamma = {gamma:.10f}, f(gamma) = {f_val:.10f},
df(gamma) = {df_val:.10f}")
        if np.isnan(f_val) or np.isnan(df_val) or df_val == 0:
            print("Newton's method 실패: f_val 또는 df_val 에 문제가 발생했습니다.")
            return gamma_old
        gamma = gamma - f_val / df_val
```

```
        # 범위를 gamma_min 과 gamma_max 로 제한
        gamma = min(gamma_max, max(gamma_min, gamma))

        # 새로운 ea 계산
        ea = abs(gamma - gamma_old)
        print(f"Updated gamma = {gamma:.10f}, ea = {ea:.10f}")
    except Exception as e:
        print(f"오류 발생: {e}")
        return gamma_old
    # 다음 iteration
    return newton_method(gamma, region, sigma, N, gamma_min, gamma_max, es, ea, iter_count
+ 1)
```

2.3 Fusion of corrected images

```
def calculate_weight(bright_region, sigma_w=0.5):
    return np.exp(-(bright_region ** 2) / (2 * sigma_w ** 2))
def fusion(Y_normalised, gamma_dark, gamma_bright, weight):
    dark_result = Y_normalised ** gamma_dark
    bright_result = Y_normalised ** gamma_bright
    # 합성, 결과는 원본 이미지와 동일한 크기 (3024, 4032)
    Y_o = weight * dark_result + (1 - weight) * bright_result
    Y_o_scaled = np.clip(Y_o * 255, 0, 255).astype(np.uint8) #0~1 인 Y_o 를 0~255 로
    return Y_o_scaled
```

2.4 Adaptive color restoration

```
def color_restoration(Y_o, Y_channel, input_R, input_G, input_B, bright_region):
    # 출력 결과 배열 초기화
    output_R = np.zeros_like(input_R, dtype=np.float64)
    output_G = np.zeros_like(input_G, dtype=np.float64)
    output_B = np.zeros_like(input_B, dtype=np.float64)

    # s 계산
    s = 1 - np.tanh(bright_region)
    epsilon = 1e-5  # 작은 값을 더해 0 으로 나누는 경우를 방지
    # 개별 채널별 출력 계산
    output_R = Y_o * ((input_R / (Y_channel + epsilon)) ** s)
    output_G = Y_o * ((input_G / (Y_channel + epsilon)) ** s)
    output_B = Y_o * ((input_B / (Y_channel + epsilon)) ** s)

    output_R = np.clip(output_R, 0, 255)
    output_G = np.clip(output_G, 0, 255)
    output_B = np.clip(output_B, 0, 255)
    output_image = np.stack([output_R, output_G, output_B], axis=-1).astype(np.uint8)
    return output_image
```

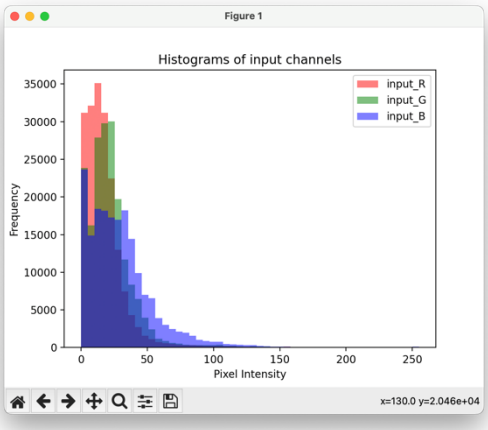


Figure 1
Histograms of input channels
input_R
input_G
input_B
Frequency
Pixel Intensity
x=130.0 y=2.046e+04

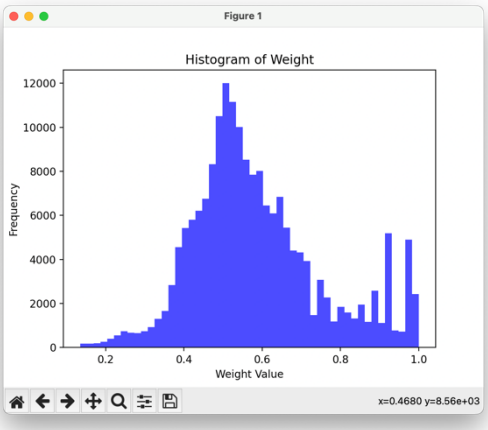


Figure 1
Histogram of Weight
Frequency
Weight Value
x=0.4680 y=8.56e+03


Y_o_scaled

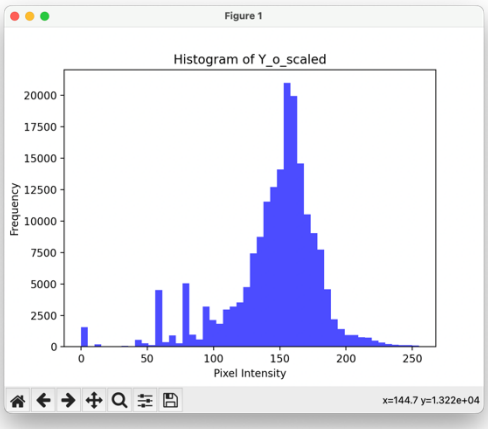


Figure 1
Histogram of Y_o_scaled
Frequency
Pixel Intensity
x=144.7 y=1.322e+04

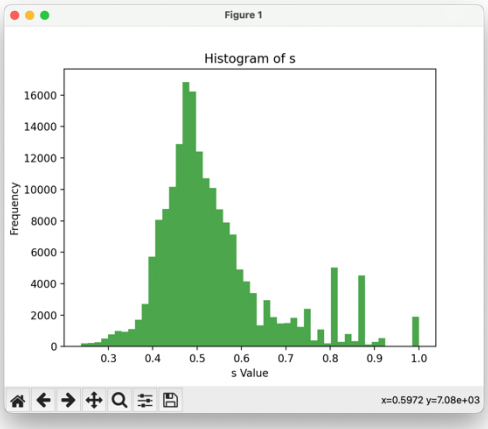


Figure 1
Histogram of s
Frequency
s Value
x=0.5972 y=7.08e+03

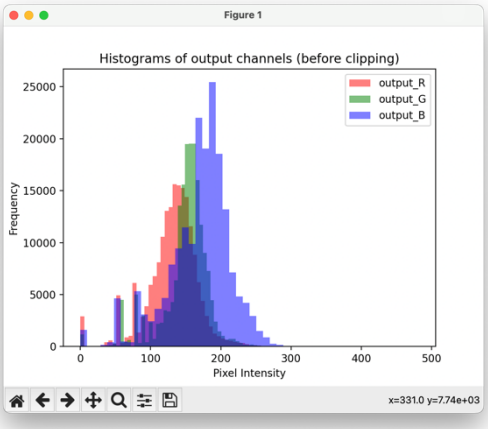


Figure 1
Histograms of output channels (before clipping)
output_R
output_G
output_B
Frequency
Pixel Intensity
x=331.0 y=7.74e+03

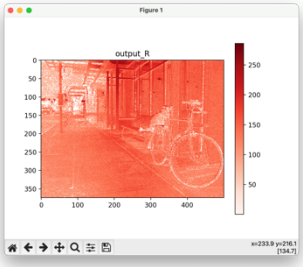


Figure 1
output_R

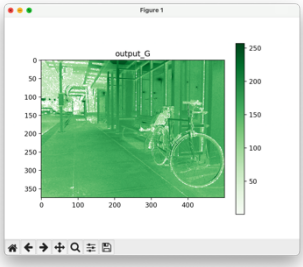


Figure 1
output_G

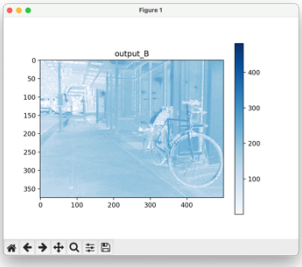
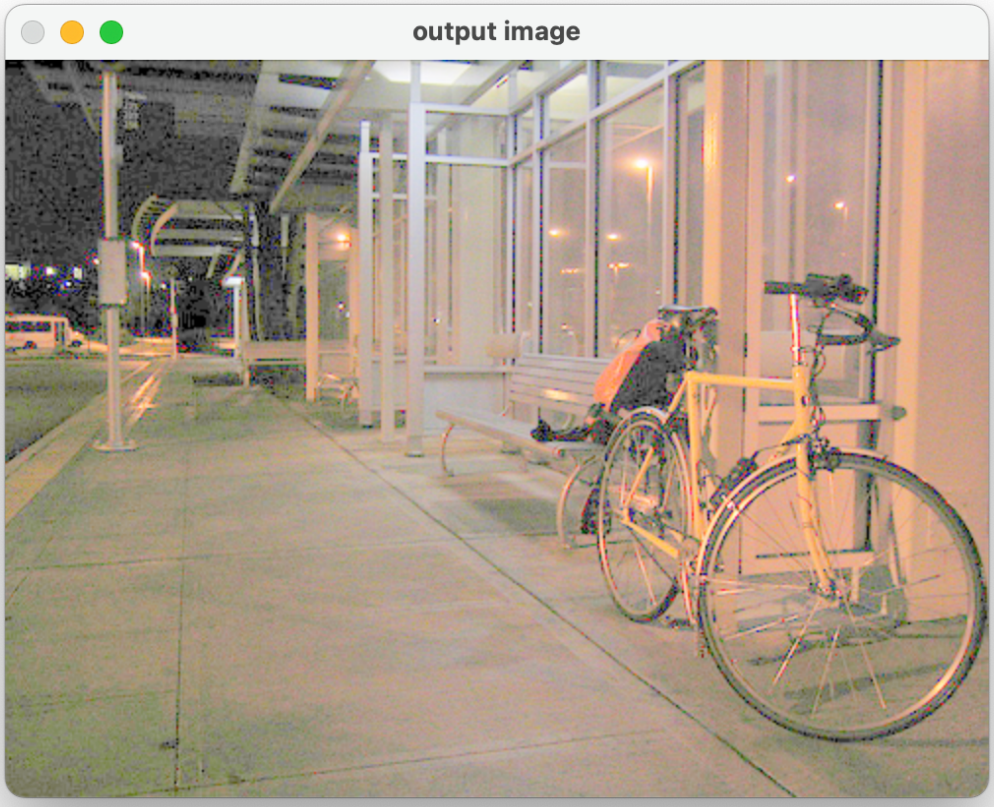


Figure 1
output_B


output image