توضیح مختصری از کد:

```
def main():
    image_path = "image.png"

    image_array = read_grayscale_image(image_path)
    image_height, image_width = image_array.shape

    value, frequency = get_value_frequencies(image_array)
    cumulative_frequency = np.cumsum(frequency)

    draw_histogram(value, frequency,
                   title="Image Histogram (Before mapping)",
                   x_label="Intensity value",
                   y_label="Count")
    draw_cumulative_frequency_plot(value, cumulative_frequency,
                                   title="Cumulative Frequency (Before
mapping)",
                                   x_label="Intensity value",
                                   y_label="Count (Cumulative)")

    mapper = create_mapper(value, cumulative_frequency, image_height,
image_width)
    map_image(image_array, mapper)

    value, frequency = get_value_frequencies(image_array)
    cumulative_frequency = np.cumsum(frequency)

    draw_histogram(value, frequency,
                   title="Image Histogram (After mapping)",
                   x_label="Intensity value",
                   y_label="Count")
    draw_cumulative_frequency_plot(value, cumulative_frequency,
                                   title="Cumulative Frequency (After
mapping)",
                                   x_label="Intensity value",
                                   y_label="Count (Cumulative)")

    store_image(image_array)
```

با توجه به تابع main برنامه، ابتدا عکس به صورت سیاه سفید خوانده می‌شود. سپس به کمک get_value_frequencies، پیکسل های منحصر به فرد در آرایه value و تعداد رخداد هرکدام در آرایه frequency قرار می‌گیرد. سپس به کمک توابع draw_histogram و draw_cumulative_frequency_plot، هیستوگرام و پلات مربوطه قبل از تبدیل رسم می‌شود. در ادامه یک mapper جهت تبدیل پیکسل ها توسط تابع create_mapper ساخته می‌شود و در تابع map_image آن ها تبدیل می‌شوند. برای مصور سازی دوباره، مقادیر پیکسل ها و تعداد رخداد ها دوباره محاسبه و هیستوگرام و پلات مربوطه رسم می‌شوند. در آخر نیز تصویر خروجی به کمک تابع store_image ذخیره می‌شود.

با توجه به این که تبدیل عکس به سیاه سفید و تبدیل کردن پیکسل ها بخش اصلی کد می‌باشند، در ادامه به توضیح آن دو میپردازیم.

در ابتدا به توابع خواندن عکس به صورت سیاه سفید می‌پردازیم:

```
def read_grayscale_image(image_path):
    """Reads the image using PIL.Image and then converts the image to
grayscale"""
    image = Image.open(image_path)
    gray_scale = convert_to_gray_scale(image)

    return gray_scale


def convert_to_gray_scale(image):
    """
    Converts RGB image to grayscale based on the following formula:
    grayscale = ( (0.3 * R) + (0.59 * G) + (0.11 * B) )

    Reference of the formula:
https://www.tutorialspoint.com/dip/grayscale_to_rgb_conversion.htm

    :param image: RGB image read by PIL.Image
    :return: gray_sale np array that elements are in range [0, 255]
    """
    image_array = np.array(image, dtype=np.float32)
    R, G, B = image_array[..., 0], image_array[..., 1], image_array[..., 2]
    gray_scale = 0.3 * R + 0.59 * G + 0.11 * B
    return np.array(gray_scale, dtype=np.uint8)
```

تصویر با استفاده از Image موجود در کتابخانه Pillow در تابع read_graysacale_image خوانده می‌شود و سپس در تابع convert_to_gray_scale به سیاه سفید تبدیل می‌شود.

برای تبدیل به سیاه سفید ابتدا مقادیر پیکسل موجود در عکس در ۳ کانال مختلف RGB در یک آرایه ۳ بعدی نامپای ریخته می‌شود. سپس با توجه به وبسایت اشاره شده در docstring موجود در کد، رنگ های مختلف طول موج مختلف و لذا تاثیر متفاوتی در شکل گیری تصویر دارند. لذا میانگین وزن دار بین این سه کانال گرفته می‌شود به طوری که رنگ قرمز با طول موج بیشتر، تاثیر کمتری نسبت به رنگ سبز با طول موج کمرت داشته باشد و بدین صورت تصویر سیاه سفید ایجاد میشود. در نهایت نیز نتیجه به یک آرایه از نوع uint8 (از ۰ تا ۲۵۵) تبدیل می‌شود.

توابع تبدیل پیکسل ها:

```
def create_mapper(value, cumulative_frequency, image_height, image_width):
    """Creates a mapper that is a dictionary to map each value intensity of
the image to a new intensity"""
    color_levels = value.shape[0]

    mapper = {}
    for color, cum_sum in zip(value, cumulative_frequency):
        mapper[color] = np.ceil((color_levels - 1) * cum_sum / (image_width *
image_height))

    return mapper


def map_image(image_array, mapper):
    """Map intensity values of the image_array into new values based on the
```

```
mapper"""
    image_height, image_width = image_array.shape
    for i in range(image_height):
        for j in range(image_width):
            image_array[i, j] = mapper[image_array[i, j]]
```

به کمک تابع create_mapper، تبدیل را انجام می‌دهیم. به طوری که مشابه رابطه زیر هر رنگ (Intensity value) به رنگ جدید نگاشته می‌شود:

$$T(c) = round(\#color\_levels - 1) * cumulative\_sum(c) / (image\_width * image\_height))$$

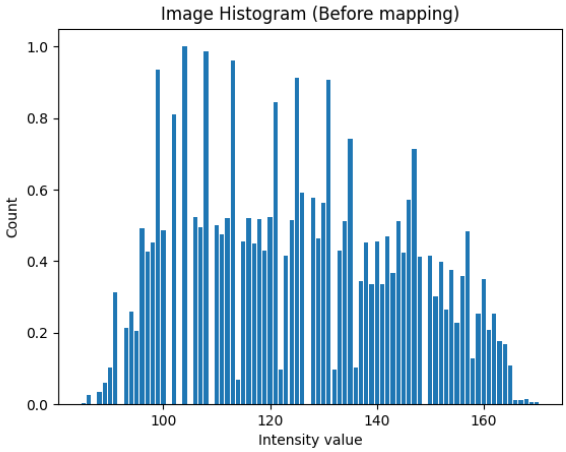
همانطور که مشاهده می‌کنیم، در داخل حلقه for این تابع، به ازای هر پیکسل منحصر به فرد در آرایه values این نگاشت صورت گرفته است.

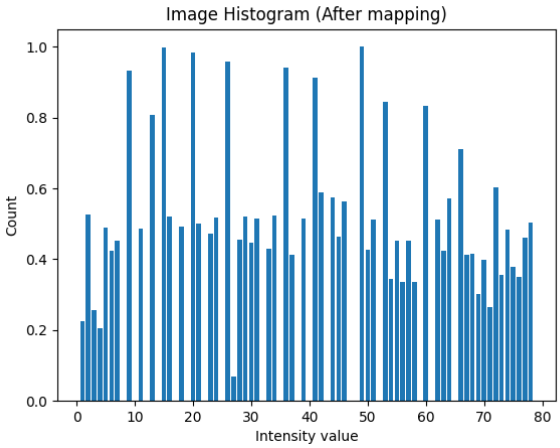
برای نگاشت پیکسل ها به مقدار جدید نیز تابع map_image فراخوانی میشود که هر پیکسل در محل i,j ام را به مقدار جدید نگاشت می‌کند.

پاسخ به پرسش ها:

۱- خروجی حاصل پس از نگاشت برابر است با:

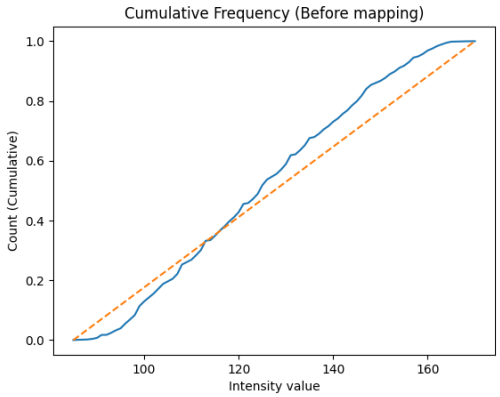
۲- هیستوگرام تصویر ورودی و خروجی:



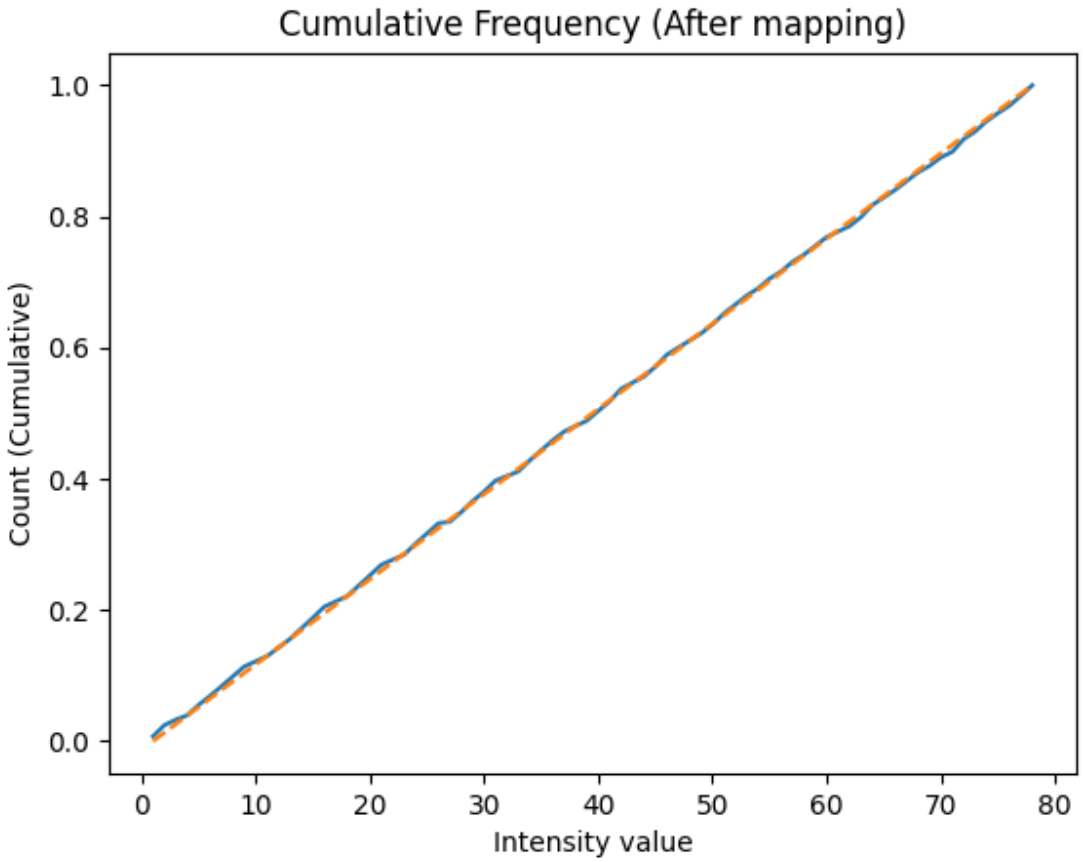


با توجه به دو هیستوگرام فوق مشاهده می‌کنیم که قبل از نگاشت، پیکسل هایی که مقادیر بزرگی داشته اند و روشن تر بوده اند، به تعداد بیشتری تکرارشده اند و باعث شده است که عکس contrast کمی داشته باشد و روشن به نظر برسد. اما پس از تبدیل مشاهده می‌کنیم که مقادیر بین ۰ تا حدود ۷۸ جمع شده اند و باعث تیر تر شدن مقادیر پیکسل شده است و contrast را افزایش داده است.

۳- نمودار جمع تجمعی تصاویر:



با توجه به تصویر فوق مشاهده می‌کنیم که پیکسل هایی که مقادیر بالاتر از ۱۲۰ را داشته‌اند، به مقدار بیش تر از آن نگاشت شده‌اند و بالای خط صاف نارنجی رنگ قرار گرفته اند. این سبب می‌شود که پیکسل ها مقادیر روشن تر از چیزی که باید داشته باشند را اختیار کنند.



پس از متعادل‌سازی مشاهده می‌کنیم که پیکسل‌ها همگی به همان میزان که نیاز داشتند تکرار شده اند و تعادل سازی رعایت شده است.