Lab 2.2

Image Enhancement with Statistical Operation

Histogram Equalization

```
[1]: import cv2
     import matplotlib.pyplot as plt
     import numpy as np
     from mpl_toolkits.axes_grid1 import ImageGrid
```

[1] จะเป็นการ Import Library ที่จำเป็นที่ต้องใช้ใน Lab 2.2 นี้

```
[2]: ### START CODE HERE ###
     # img = cv2.imread('images/test.png')
     img = cv2.imread('images/watermelon.jpg')
     img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
     plt.imshow(img)
     plt.axis('off')
     plt.show()
     ### END CODE HERE ###
```

[2] - จะเป็นการโหลดรูปมาใช้งาน โดยใช้การ Read watermelon.jpg using cv2.imread

- Convert BGR to RGB using cv2.cvtColor

- และสุดท้ายเป็นการแสดงรูปภาพ show watermelon image use plt ที่ได้หลังจากการ convert

[3]:
```python
### START CODE HERE ###
def plot_histogram(image):
    # Split the image into channels
    channels = cv2.split(image)

    # Define histogram parameters
    hist_size = [256]  # Number of bins (one for each intensity level)
    ranges = [0, 256]  # Range of pixel intensities (0 to 255)

    # Calculate the histogram for each color channel
    hist = []
    for channel in channels:
        hist.append(cv2.calcHist([channel], [0], None, hist_size, ranges))

    return hist
### END CODE HERE ###
```

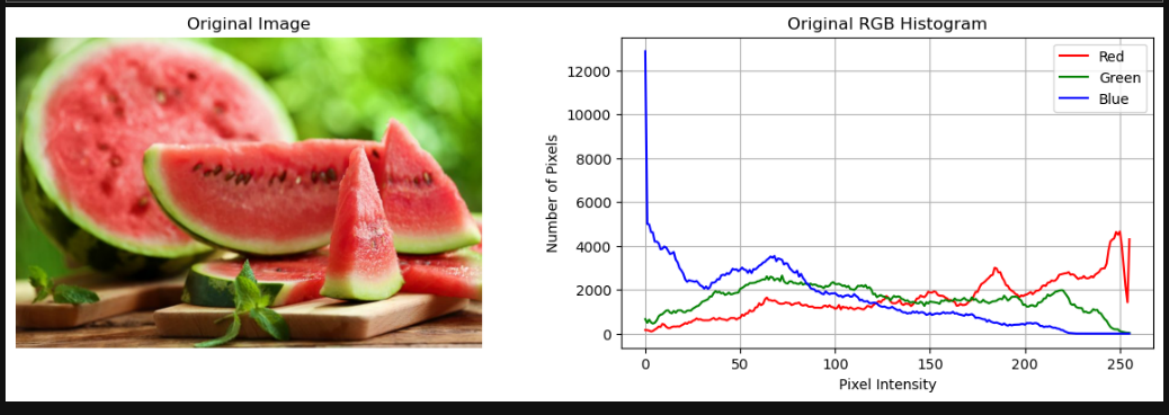
[3] - write Function to plot histogram เนื่องจากเป็นการ Lab นี้เป็นการเรียนรู้การใช้ Histogram Equalization จึงต้องใช้ function นี้เพื่อเป็นการแบ่งแยก จำนวนของ pixel ต่อ สีในแต่ละ channel โดยจะรับ input parameter มาเป็นรูปที่จะนำมาคิดเป็น histogram และ return ออกไปเป็น list : hist ที่แทน histogram ของแต่ละ channel

[3.1] - split image into Channels using cv2.split เป็นการแยก channel เพื่อจัดเก็บข้อมูล histogram ที่ได้ในแต่ละ channel สี (R, G, B)

[3.2] - define parameters (number of bins for the histogram = 256, range of pixel intensities = 0 to 256) -> Cover all possible intensity value for 8 bits image

[3.3] - compute the histogram using cv2.calcHist -> loop all channels -> return hist โดยผลลัพธ์ที่ได้จะเป็นข้อมูล histogram ของแต่ละ channel โดยจะเก็บทั้งหมดใน list : hist โดยที่ hist[0] จะแทน channel R, hist[1] จะแทน channel G และ hist[2] จะแทน channel B

```
[4]: ### START CODE HERE ###

     fig, axs = plt.subplots(1, 2, figsize=(15, 4))

     axs[0].imshow(img)
     axs[0].set_title('Original Image')
     axs[0].axis('off')

     hist = plot_histogram(img)
     # Plot the histograms
     axs[1].set_title("Original RGB Histogram")
     axs[1].plot(hist[0], color='r', label='Red')
     axs[1].plot(hist[1], color='g', label='Green')
     axs[1].plot(hist[2], color='b', label='Blue')
     axs[1].set_xlabel("Pixel Intensity")
     # axs[1].set_xlim([0, 255])  # Set x-axis range to match the histogram bins
     # axs[1].set_xticks(range(0, 256, 50))  # Set x-axis ticks every 50 units
     axs[1].set_ylabel("Number of Pixels")
     axs[1].grid(True)
     axs[1].legend()
     plt.show()

     ### END CODE HERE ###
```



[4] - จะเป็นการแสดงรูปภาพคู่กับ histogram ที่ได้ของรูปภาพนั้นๆ โดยที่ histogram ทั้ง 3 channel จะถูกแสดงอยู่ในกราฟขนาด/อันเดียวกัน เพื่อเป็นการแสดงรูปภาพต้นแบบก่อนที่จะไปทำการ Histogram Equalization และ Histogram Matching ในข้อต่อๆไป

- display Original image and Original RGB -Histogram in different axis using plt (โดยจะส่งรูปเข้าไป compute histogram by plot_histogram function)

[6]:
```python
def equalize_image(image):
    """
    Equalizes the histogram of an image for each channel (BGR format).

    Args:
        image: The image to be equalized (BGR format).

    Returns:
        numpy.ndarray: The image with equalized histogram (BGR format).
    """

    # Split the image into channels
    channels = cv2.split(image)

    # Equalize histogram for each channel
    equalized_channels = []
    for channel in channels:
        equalized_channels.append(cv2.equalizeHist(channel))

    # Merge equalized channels back into BGR image
    image_eq = cv2.merge(equalized_channels)

    return image_eq
```

[6] - ต่อมาเป็นการทำให้ภาพเป็นการทำการให้รูปๆนั้นมีการกระจาย เฉดสี ให้เท่าๆกัน โดยการใช้ Histogram Equalization ซึ่งเป็นวิธีที่กระจายเฉดสีให้เท่ากัน ตามสัดส่วนเชิงปริมาณ (ความน่าจะเป็น) ของเฉดสีนั้นๆ (อิงตาม slide อาจารย์)

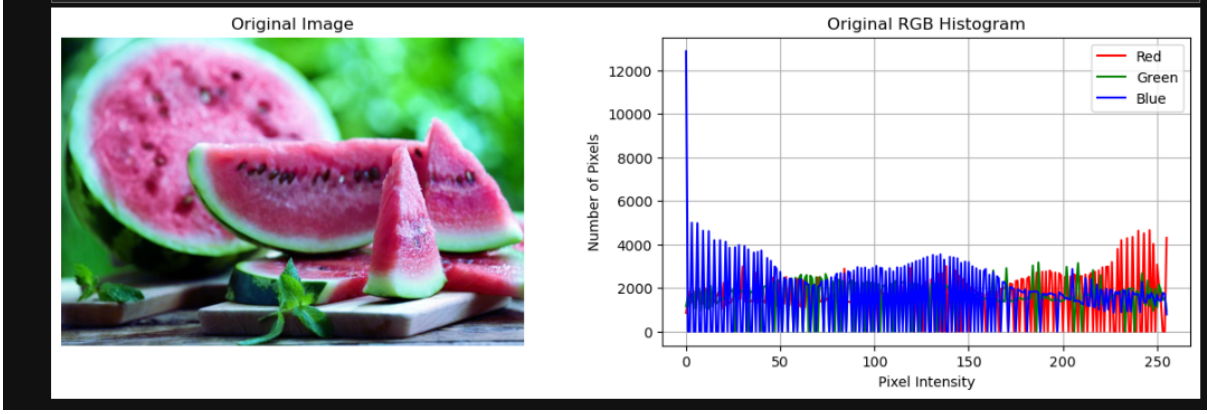
- ซึ่งกลุ่มของผมจะแยก function เพิ่ม โดยการ write equalize image function (function for spreading out the most frequent intensity values to enhances the contrast of the image) โดยที่รับ input parameter มาเป็นรูปที่ต้องการทำ Histogram Equalization และ return ออกไปเป็นรูปใหม่ที่มีการทำการกระจายเฉดแสงแล้วออกไป โดยที่ function: equalize_image สามารถอธิบายได้ดังนี้

[6.1] - split image into Channels using cv2.split (0: R, 1:G, 2:B)

[6.2] - loop to equalize histogram for each channel using cv2.equalizeHist

[6.3] - merge equalized channels back into RGB image using cv2.merge จะได้รูปภาพที่มีการทำ Histogram Equalization แล้วนั้นเอง

```
[7]: ### START CODE HERE ###
     fig, axs = plt.subplots(1, 2, figsize=(15, 4))

     equalized_image = equalize_image(img)

     axs[0].imshow(equalized_image)
     axs[0].set_title('Original Image')
     axs[0].axis('off')

     hist = plot_histogram(equalized_image)
     # Plot the histograms
     axs[1].set_title("Original RGB Histogram")
     axs[1].plot(hist[0], color='r', label='Red')
     axs[1].plot(hist[1], color='g', label='Green')
     axs[1].plot(hist[2], color='b', label='Blue')
     axs[1].set_xlabel("Pixel Intensity")
     # axs[1].set_xlim([0, 255])  # Set x-axis range to match the histogram bins
     # axs[1].set_xticks(range(0, 256, 50))  # Set x-axis ticks every 50 units
     axs[1].set_ylabel("Number of Pixels")
     axs[1].grid(True)
     axs[1].legend()
     plt.show()

     ### END CODE HERE ###
```



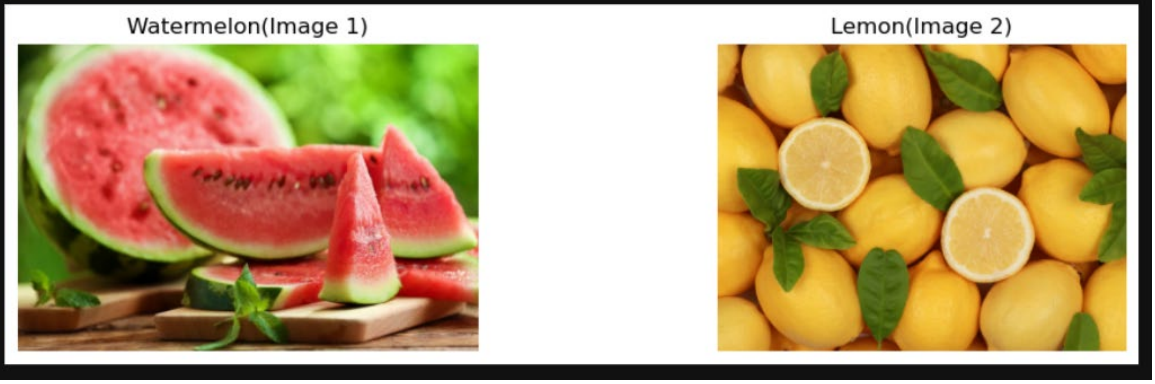
[7] - เช่นเดียวกันกับรูปต้นแบบ บล็อกนี้จะเป็นการเขียนคำสั่งให้แสดงรูปภาพที่มีการกระจายเฉดแสง คู่กับ กราฟ histogram ของรูปที่มีการกระจายเฉดแสงนั้นเอง

- display Original image and equalized RGB Histogram in different axis using plt (equalize image by equalized_image function)

- โดยจะเห็นว่ารูปที่ได้มีการกระจายเฉดแสง ซึ่งสามารถดูได้จากกราฟที่แต่ละ channel มีการทำ histogram equalization ทำให้ถ้าสัดส่วนของเฉดสีมีความน่าจะเป็นที่มีมาก จะมีระยะห่างของเฉดสีนั้นกับเฉดสีก่อนหน้าด้วยระยะห่างมาก ทำให้เสดสีของ channel B และ R (เท่าที่สังเกตได้) มีการกระจายตัวที่ห่างจากเดิมนั้นเอง

Image Histogram Matching

[8]:
```
### START CODE HERE ###
# img1 = cv2.imread('images/test.png')
# img2 = cv2.imread('images/test2.png')
img1 = cv2.imread('images/watermelon.jpg')
img2 = cv2.imread('images/lemon.png')

img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2RGB)

fig, axs = plt.subplots(1, 2, figsize=(12, 3))

axs[0].imshow(img1)
axs[0].set_title('Watermelon(Image 1)')
axs[0].axis('off')


axs[1].imshow(img2)
axs[1].set_title('Lemon(Image 2)')
axs[1].axis('off')
plt.show()
### END CODE HERE ###
```



[8] - เป็นการนำรูป 2 รูปโหลดเข้ามาใช้งาน นั้นคือการ Read watermelon.jpg and lemon.png using cv2.imread ซึ่งกำหนดให้ รูปแตงโมเป็น img1 และรูป lemon เป็น img2 ซึ่ง img1 จะมีการแลก cdf กับรูป img2 ในการทำ histogram matching ในบล็อกต่อๆไปนั้นเอง

- Convert BGR to RGB using cv2.cvtColor

- แสดงรูปภาพทั้ง 2 โดยการ show image in different axis using plt

[9]:
```
def compute_pdf_cdf(hist):
    pdf, cdf = [],[]
    for channel in range(len(hist)):
        pdf.append(hist[channel] / np.sum(hist[channel]))
        cdf.append(np.cumsum(pdf[channel]))
    return pdf, cdf
```

[9] - ซึ่งในการทำ histogram matching จะต้องมีการใช้ pdf เพื่อใช้ในการคำนวณหา cdf ในกระบวนการต่อมาทำให้กลุ่มของพวกผมมีการเขียน Function แยกออกมาเพื่อคำนวณหา pdf และ cdf โดยเฉพาะ โดยจะรับ input เป็น histogram ของทั้ง 3 channel และคำนวณผลลัพธ์ return ออกมาเป็น list ของ pdf และ cdf ของแต่ละ channel นั้นเอง

- write function to compute the Probability Density Function (PDF) and Cumulative Distribution Function (CDF) from the histogram of each channel of an image โดยใน Function มีกระบวนการทำงานดังนี้

[9.1] - loop each channels in histogram

[9.2] - compute pdf by dividing the histogram values by the sum of the histogram values (normalize step to ensures that the pdf represents the probability distribution of pixel intensities)

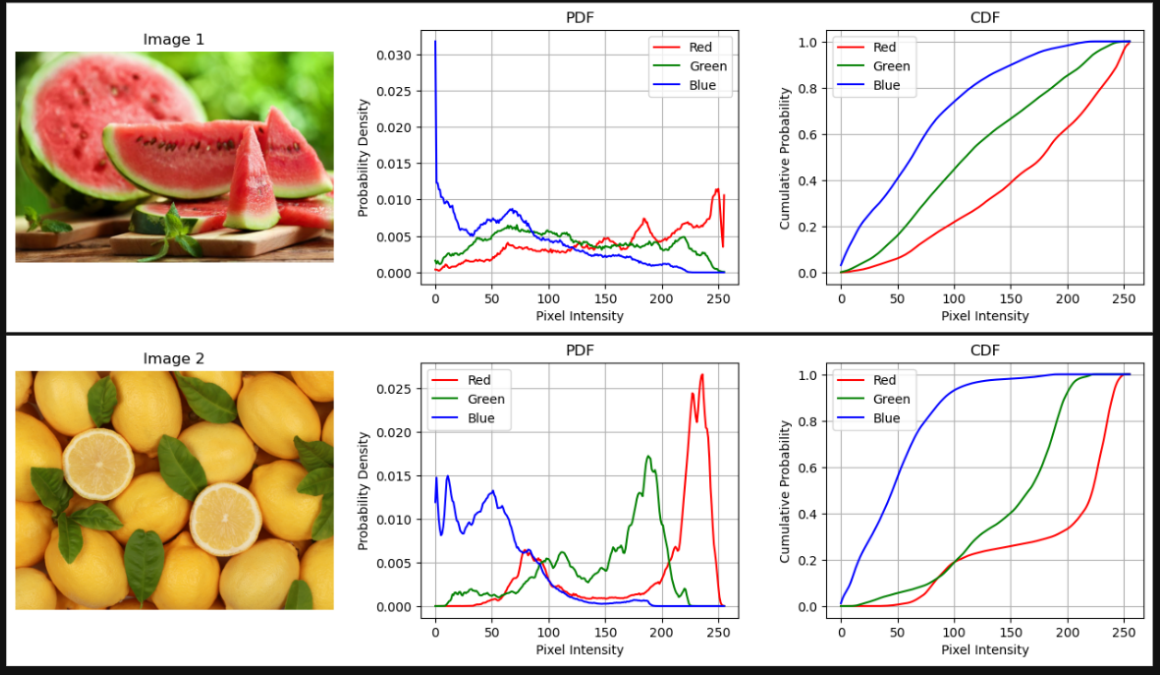
[9.3] - compute cdf by take the cumulative sum of the pdf values (np.cumsum)

[10]:
```python
def plot_image_pdf_cdf(image, title_name):

    # Compute PDF and CDF
    hist = plot_histogram(image)
    pdf_img, cdf_img = compute_pdf_cdf(hist)

    # Create figure and subplots
    fig, axs = plt.subplots(1, 3, figsize=(13, 4))

    # Plot Image
    axs[0].imshow(image)
    axs[0].set_title(f'{title_name}')
    axs[0].axis('off')

    # Plot PDFs
    axs[1].plot(pdf_img[0], color='r', label='Red')
    axs[1].plot(pdf_img[1], color='g', label='Green')
    axs[1].plot(pdf_img[2], color='b', label='Blue')
    axs[1].set_title('PDF')
    axs[1].set_xlabel("Pixel Intensity")
    axs[1].set_ylabel("Probability Density")
    axs[1].grid(True)
    axs[1].legend()

    # Plot CDFs
    axs[2].plot(cdf_img[0], color='r', label='Red')
    axs[2].plot(cdf_img[1], color='g', label='Green')
    axs[2].plot(cdf_img[2], color='b', label='Blue')
    axs[2].set_title('CDF')
    axs[2].set_xlabel("Pixel Intensity")
    axs[2].set_ylabel("Cumulative Probability")
    axs[2].grid(True)
    axs[2].legend()

    plt.tight_layout(pad=2.0)
    # Display the plot
    plt.show()
```

[10] - ซึ่งต่อมากลุ่มของพวกผมได้มีการเขียน function แยกขึ้นมาอีก function นึงซึ่งเป็น function ที่รับ parameter มาเป็นรูปภาพ และชื่อ tittle โดยจะมีการ plot รูป และคำนวณจาก function: compute_pdf_cdf ที่อธิบายไปปก่อนหา เพื่อจะได้มา plot กราฟ pdf และ cdf ของแต่ละ channel ของรูปนั้นนั้นเอง เพื่อดูว่ารูป img1 และ img2 มี pdf และ cdf เป็นอย่างไร

- write function to plot image, plot computed pdf and plot computed cdf (using compute_pdf_cdf function) in different axis using plt

```
[11]: image_paths = [img1, img2]

for i, img in enumerate(image_paths):
    plot_image_pdf_cdf(img, f"Image {i+1}")
```



[11] - โดยบล็อกนี้ก็จะเป็นการใช้ function : plot_image_pdf_cdf ที่อธิบายไปบล็อกก่อนหน้ามาแสดง โดยการ loop every image to display image, pdf, cdf (using plot_image_pdf_cdf function)

```
def histogram_match_using_lookup(img1, img2):
    # Calculate histograms
    hist_img1 = plot_histogram(img1)
    hist_img2 = plot_histogram(img2)

    # Compute PDFs and CDFs
    pdf_img1, cdf_img1 = compute_pdf_cdf(hist_img1)
    pdf_img2, cdf_img2 = compute_pdf_cdf(hist_img2)

    # Create lookup table
    lookup_table = []
    for channel in range(len(pdf_img1)):
        lookup_channel = np.zeros(256, dtype=np.uint8)
        for i in range(256):
            lookup_channel[i] = np.argmax(cdf_img2[channel] >= cdf_img1[channel][i])
        lookup_table.append(lookup_channel)

    # Apply lookup table to image
    matched_img = np.zeros_like(img1)
    for channel in range(len(img1.shape)):
        matched_img[:,:,channel] = lookup_table[channel][img1[:,:,channel]]

    return matched_img
```

[18] - ต่อมาก็จะเป็นการใช้ histogram matching โดยกลุ่มของผมได้เขียน function แยกออกมา โดยที่จะรับ input มาเป็นรูปแรกเริ่ม (img1) และรูปต้นแบบ (img2) ที่จะมีการใช้ cdf ของรูปนี้ไปแลกกับรูปแรกเริ่มนั้นเอง โดยจะ return ผลลัพธ์ออกไปเป็นรูปที่ทำการ histogram matching แล้วนั้นเอง

- write function to perform histogram matching between two images using lookup โดยใน function มีกระบวนการทำงานดังนี้

[18.1] - compute histogram using plot_histogram function ที่เขียนไว้ในบล็อกที่ [3]

[18.2] - compute pdf and cdf using compute_pdf_cdf function ที่เขียนไว้ในบล็อกที่ [9]

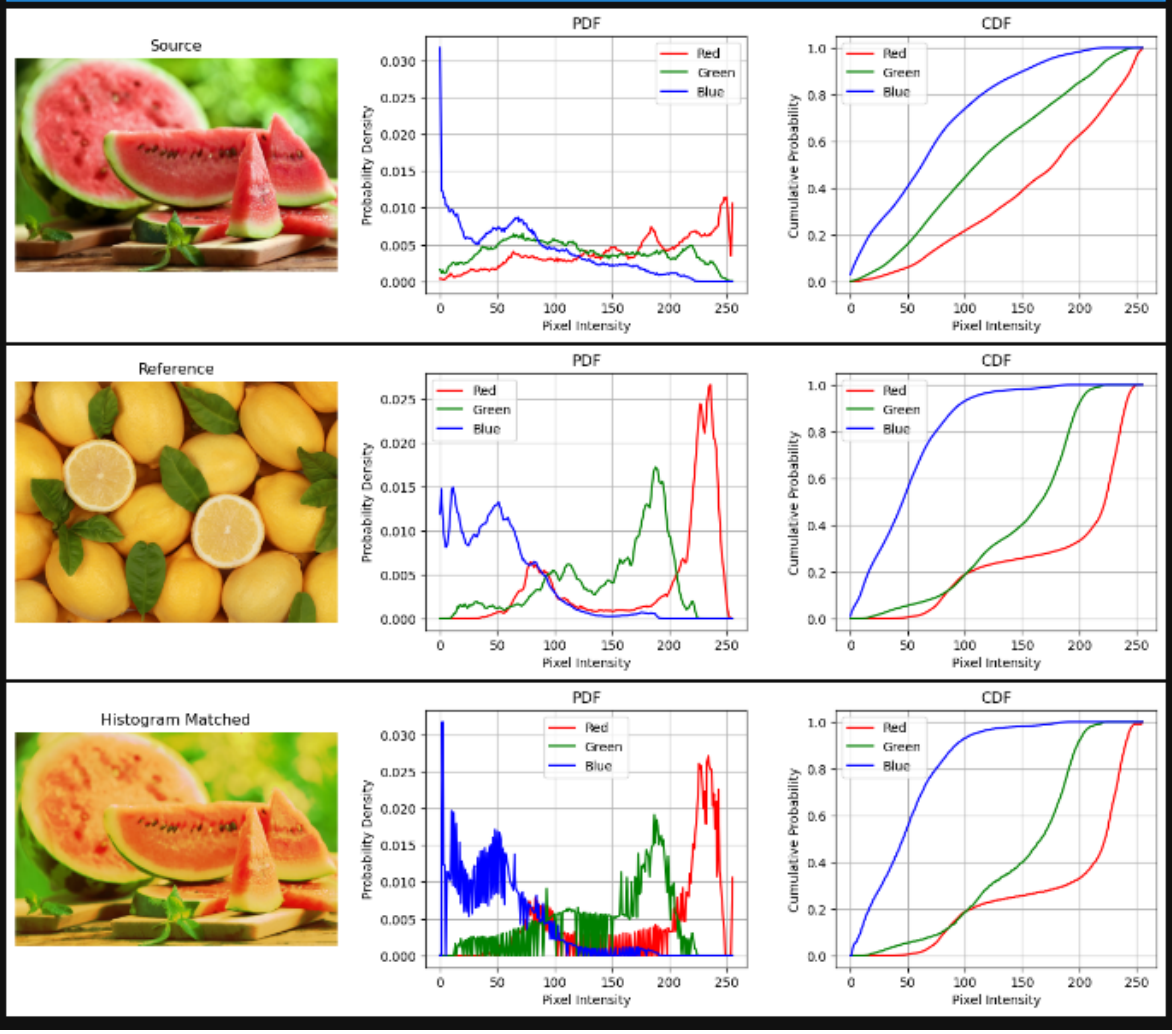
[18.3] - create lookup table for each color channel เพื่อที่เป็นการ loop เข้าไป cdf ของ img1 ซึ่งจะไป lookup cdf ของ img2 นั้นเอง

- create array with 256 elements, all set to zero
- loop to get index corresponds to the intensity value in the reference image which cdf is closest to (using np.argmax) โดยจะเก็บ cdf ที่ใกล้เคียงเป็น index ของ cdf ของ img2 นั้นเอง

[18.4] - apply lookup table to image เป็นการนำ index ที่ได้จากการ matching cdf จาก img2 มาสร้างเป็นรูป โดยมีวิธีการทำดังนี้

- create empty image array with the same shape as the source (ซึ่งก็คือ img1) using np.zeros_like
- loop to apply the lookup table to transform the pixel values of the source image for the current channel

```
matched_img = histogram_match_using_lookup(img1, img2)

plot_image_pdf_cdf(img1, "Source")
plot_image_pdf_cdf(img2, "Reference")
plot_image_pdf_cdf(matched_img, "Histogram Matched")

### END CODE HERE ###
```



[18] - ซึ่งต่อมาจะเป็นการเรียกใช้ function : histogram_match_using_lookup ที่ได้อธิบายไว้ก่อนหน้า โดยการกำหนดให้ img1 = รูปแตงโม และ img2 = รูปเลม่อนนั้นอง ซึ่งผลลัพธ์จะเก็บรูปที่ได้ในตัวแปร matched_img

- match Source image shape with Reference image CDF using histogram_match_using_lookup function

[18] - สุดท้ายก็จะเป็นการแสดงรูปภาพ, กราฟ pdf และ cdf ของรูป source, Reference และรูปที่ได้จากการ match histogram โดยการเรียกใช้ function : plot_image_cdf ที่เขียนไว้ในบล็อก [10] นั้นเอง

- ซึ่งจะเห็นว่ารูปที่ได้จากการทำ histogram matching จะมี cdf ที่ใกล้เคียงกับ ภาพ reference ซึ่งทำให้เฉดสีของภาพที่ได้มีการเปลี่ยนไป โดยมีลักษณะอยู่ในโทนเดียวกันกับภาพ reference ซึ่งการทำ histogram matching เป็นเทคนิคหนึ่งในการประมวลผลภาพที่ใช้ในการเปลี่ยนแปลงการกระจายของความเข้มของภาพ (intensity

distribution) ให้มีลักษณะเหมือนกับฮิสโทแกรมที่ต้องการ โดยเป้าหมายหลักคือเพื่อปรับปรุงคุณภาพของภาพให้สอดคล้องกับเงื่อนไขที่ต้องการ หรือเพื่อให้ภาพหลายภาพมีลักษณะที่คล้ายคลึงกันมากขึ้น

ซึ่ง histogram matchimg มักใช้ใน

- **ปรับปรุงคุณภาพภาพ:**
    - **แก้ไขปัญหาแสงไม่สม่ำเสมอ:** เมื่อภาพมีส่วนที่สว่างเกินไปหรือมืดเกินไป การปรับฮิสโทแกรมจะช่วยให้ภาพมีความสว่างที่สมดุลมากขึ้น
    - **ปรับปรุงคอนทราสต์:** ทำให้รายละเอียดของภาพชัดเจนขึ้น
    - **ปรับสี:** สามารถปรับสีของภาพให้เป็นไปตามที่ต้องการ เช่น ทำให้ภาพดูอบอุ่นขึ้นหรือเย็นลง
- **ทำให้ภาพหลายภาพมีลักษณะคล้ายคลึงกัน:**
    - **การรวมภาพ:** เมื่อต้องการนำภาพหลายภาพมารวมกัน การปรับฮิสโทแกรมจะช่วยให้ภาพแต่ละภาพมีลักษณะที่คล้ายคลึงกัน ทำให้ภาพที่ได้มีความต่อเนื่องและเป็นธรรมชาติมากขึ้น