

تمرین ۶ سوال ۱-۱

چکیده

در این تمرین قصد داریم تصویر را به فرمت HSI نمایش دهیم. در نهایت هر Channel را به صورت جداگانه نمایش داده و کاربرد هر کدام را بررسی می کنیم.

مقدمه

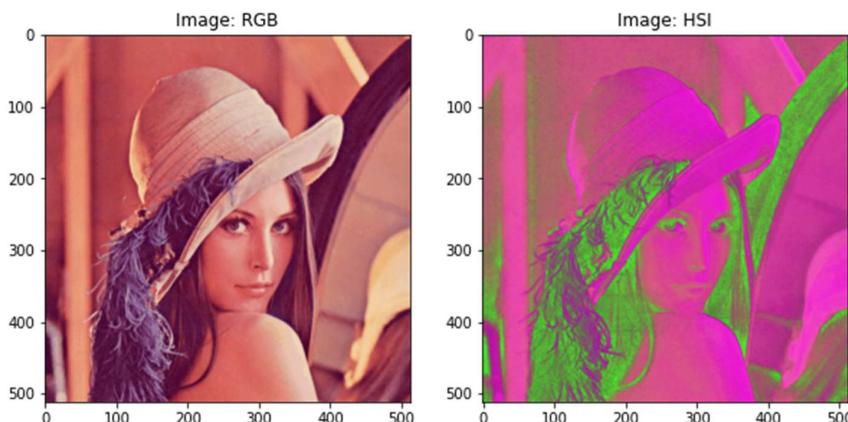
برای تبدیل RGB به HSI لازم است تا طبق فرمول های گفته شده در اسلاید های درس عمل کرده و مقدار معادل هر پیکسل با مقادیر R و G و B را به H و S و I تبدیل کنیم.

$$I = \frac{1}{3}(R + G + B)$$
$$H = \cos^{-1} \left\{ \frac{\frac{1}{2}[(R - G) + (R - B)]}{[(R - G)^2 + (R - B)(G - B)]^{1/2}} \right\}$$
$$S = 1 - \frac{3}{(R + G + B)}[\min(R, G, B)]$$

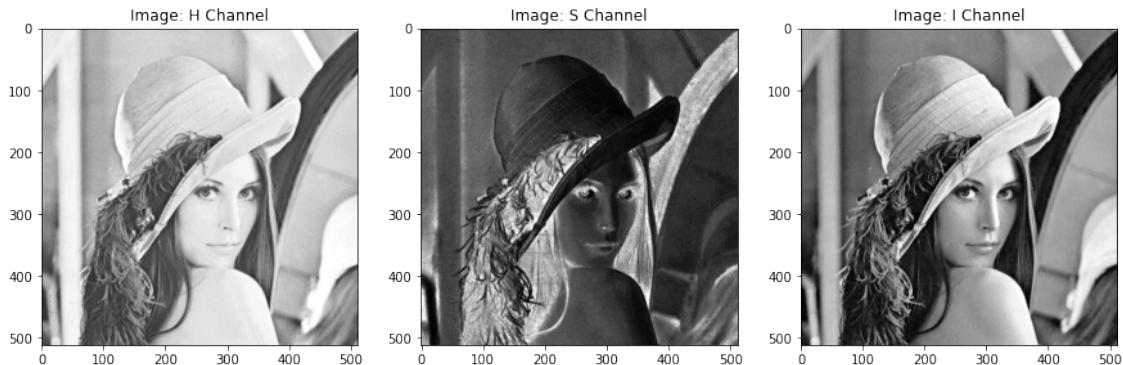
در نهایت و پس از بازسازی تصویر با Channel های جدید، هر کدام را نیز به صورت جداگانه نمایش می دهیم. با دقت در هر کدام از Channel ها، متوجه نوع هر کدام می شویم.
Hue (H) یا فام به طول موج رنگ غالبی که توسط بیننده دریافت می شود، گفته می شود.
Saturation (S) یا اشباع میزان خالص بودن رنگ را با میزان مخلوط شدن آن با رنگ سفید نمایان می کند. این رابطه یک رابطه معکوس بوده و هرچه کمتر با سفید مخلوط شده باشد، این خالص بودن بیشتر است و برعکس.
Intensity (I) یا شدت میانگین بدون وزن رنگ ها را مشخص می کند. این Channel بیشترین اطلاعات تصویر را دارد.

شرح نتایج

در ابتدا تصویر در فرمت RGB را به همراه تصویر در فرمت HSI نمایش می دهیم.



در ادامه با استفاده از خواندن لایه های مختلف تصویر، Channel های H و S و I را به صورت جداگانه میبینیم.



```
In [ ]: import cv2
import numpy as np
import math
import matplotlib.pyplot as plt
from google.colab import drive
```

```
In [ ]: drive.mount('/content/drive')
```

Mounted at /content/drive

```
In [ ]: img_bgr = cv2.imread("/content/drive/MyDrive/Images/6/Lena.bmp")
img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
```

```
In [ ]: def RGB_TO_HSI(image):
    with np.errstate(divide='ignore', invalid='ignore'):
        image = np.float32(image)/255

        red = image[:, :, 0]
        green = image[:, :, 1]
        blue = image[:, :, 2]

        def calc_intensity(red, blue, green):
            return np.divide(blue + green + red, 3)

        def calc_saturation(red, blue, green):
            minimum = np.minimum(np.minimum(red, green), blue)
            saturation = 1 - (3 / (red + green + blue + 0.001) * minimum)
            return saturation

        def calc_hue(red, blue, green):
            hue = np.copy(red)

            for i in range(0, blue.shape[0]):
                for j in range(0, blue.shape[1]):
                    hue[i][j] = 0.5 * ((red[i][j] - green[i][j]) + (red[i][j] - blue[i][j])) / math.sqrt((red[i][j] - green[i][j])**2 + ((red[i][j] - blue[i][j]) * (green[i][j] - blue[i][j])))
                    hue[i][j] = math.acos(hue[i][j])

            if blue[i][j] <= green[i][j]:
                hue[i][j] = hue[i][j]
            else:
                hue[i][j] = ((360 * math.pi) / 180.0) - hue[i][j]
            return hue

        hsi = cv2.merge((calc_hue(red, blue, green), calc_saturation(red, blue, green), calc_intensity(red, blue, green)))
    return hsi
```

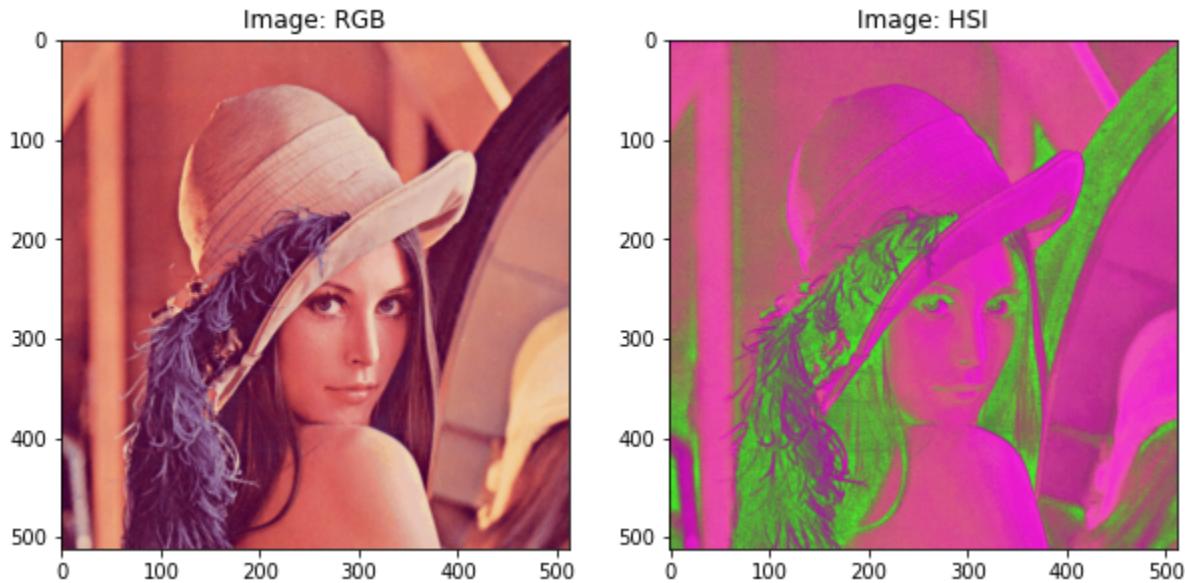
```
In [ ]: img_hsi = RGB_TO_HSI(img_rgb)

fig, plot = plt.subplots(1, 2, figsize = (10, 5))

plot[0].imshow(img_rgb)
plot[0].set_title("Image: RGB")

plot[1].imshow(img_hsi)
plot[1].set_title("Image: HSI")
```

Out[]: Text(0.5, 1.0, 'Image: HSI')



```
In [ ]: img_h = img_hsi[:, :, 0]
img_s = img_hsi[:, :, 1]
img_i = img_hsi[:, :, 2]

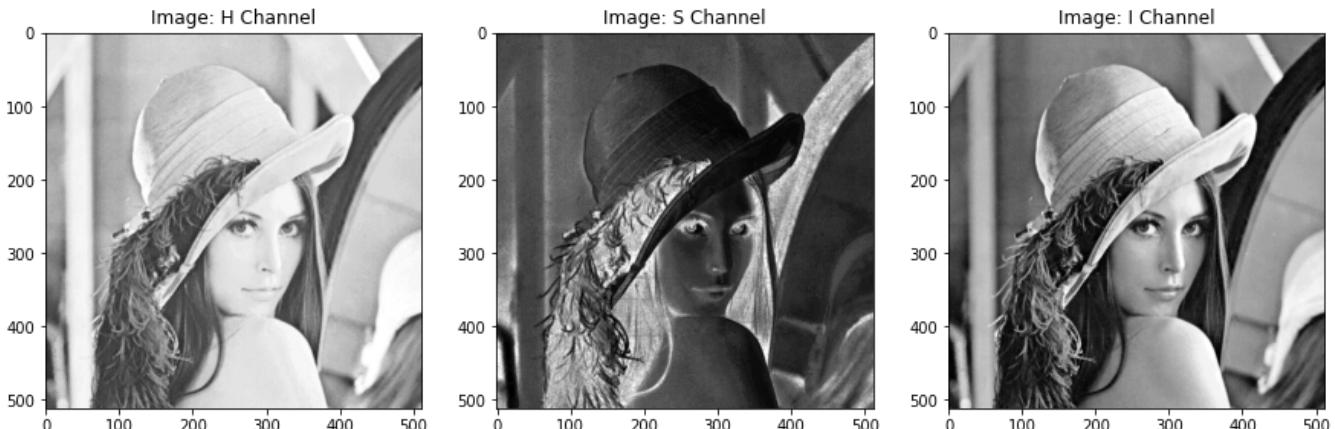
fig, plot = plt.subplots(1, 3, figsize = (15, 5))

plot[0].imshow(img_h, cmap='gray')
plot[0].set_title("Image: H Channel")

plot[1].imshow(img_s, cmap='gray')
plot[1].set_title("Image: S Channel")

plot[2].imshow(img_i, cmap='gray')
plot[2].set_title("Image: I Channel")
```

Out[]: Text(0.5, 1.0, 'Image: I Channel')



تمرين ۶ سوال ۱

چکیده

در اين تمرين با Quantization يك تصوير رنگی که به صورت Grayscale در برنامه import شده است آشنا می شويم. در ادامه با مقادير MSE و PSNR به بررسی کيفيت تصاویر در هر Quantization Level می پردازيم.

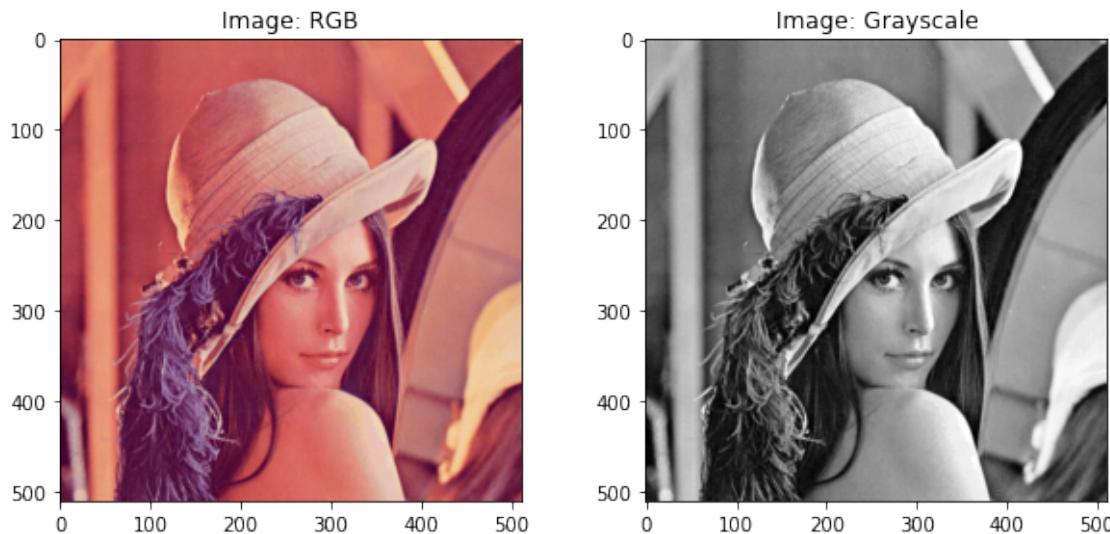
مقدمه

در تمارين قبلی برای تابع Quantization از تغییر تک به تک پیکسل ها با توجه به بازه تعیین شده توسط Level استفاده کردم ولی در این تمرین به جای استفاده از همان تابع، تابعی با استفاده از Clustering تعريف کردم. در این Level K-Means Clustering با استفاده از Quantize کردن رنگ های تصوير، با تعداد Cluster ها برابر با رنگ ها با توجه به نزدیک ترین رنگ در همسایگی خود خوش بندی می شود و در نهايیت Label ها و مراکز Cluster ها بیان کننده رنگ هر دسته خواهد بود.

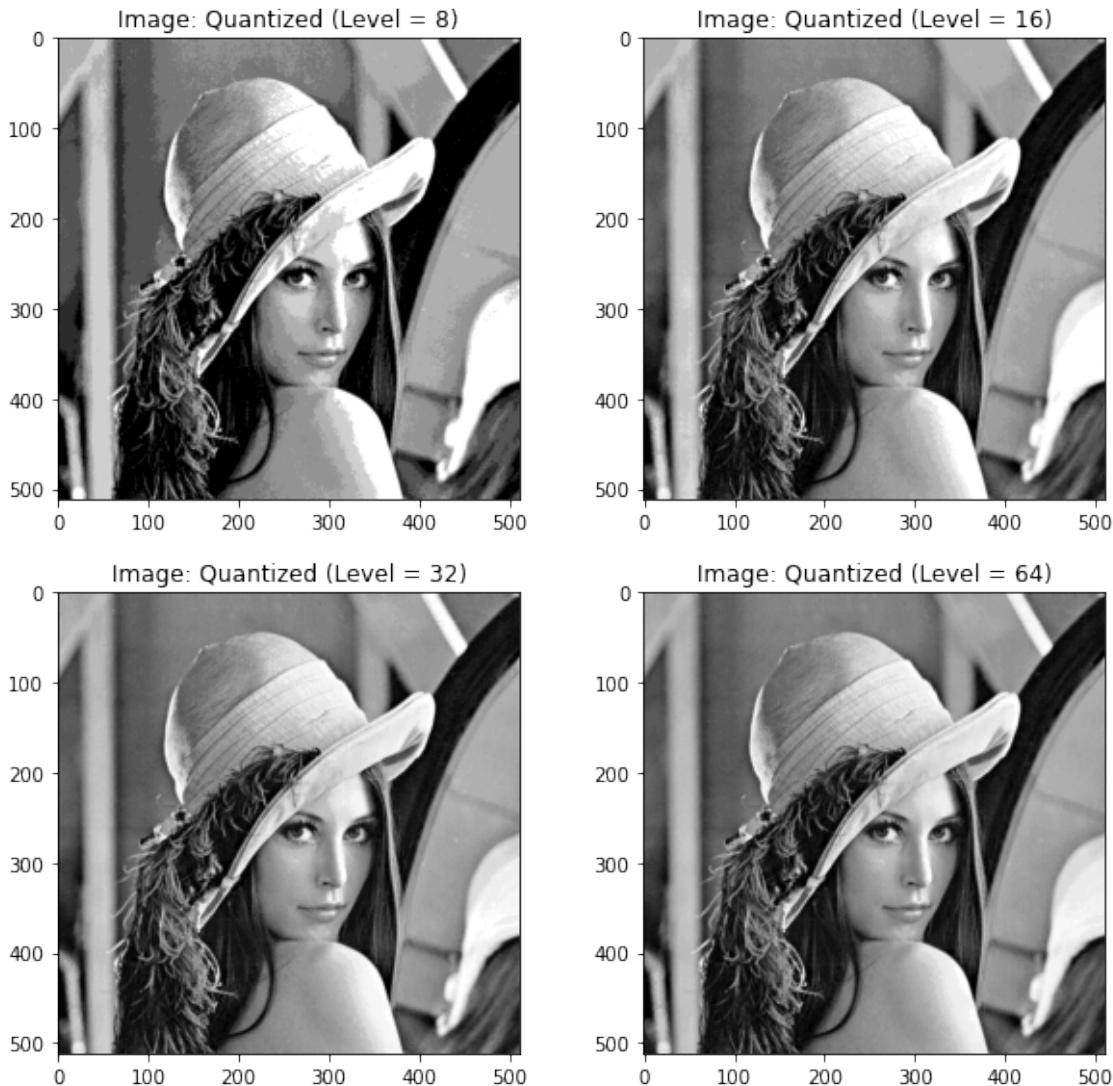
پس برای تشکیل تصویر Quantize شده ابتدا تصویر رنگی را به فرمت Grayscale می خوانیم و با استفاده از تابع بالا، آن را Quantize کرده و تصویر جدید را نمایش می دهیم. در نهايیت با محاسبه مقادير MSE و PSNR مراحل را به پایان می رسانیم. این عملیات را برای هر Level مختلف تصویر تکرار کرده تا برای مقایسه و مشاهده کاربرد نمونه های بیشتری داشته باشیم.

شرح نتایج

برای دیدن نتایج، ابتدا تصویر رنگی و تصویر Grayscale اصلی را نمایش می دهیم.



در ادامه تصویر بازسازی شده و Quantize شده با Level های مختلف، ۸، ۱۶، ۳۲ و ۶۴ را می بینیم.



همانطور که مشاهده می شود، با بالا رفتن مقدار Level تصویر با رنگ های بیشتری نمایش داده شده و به تصویر اصلی نزدیک تر می شود. همچنین تفاوت بین رنگ های همسایه و سایه ها به نرمی انجام شده و تصویر مناسب تری در اختیارمان قرار می دهد. همچنین تصویر با ۶۴ Level با اینکه از تعداد یک چهارم رنگ تصویر اصلی استفاده کرده است، باز هم برای چشم انسان خیلی قابل تشخیص نبوده و می تواند در کاربرد های مشخصی جایگزین آن شود. مقادیر MSE و PSNR که در ادامه آورده می شود نیز گویای همین گفته هاست.

Quantized Image MSE & PSNR			
Level = 8	&	MSE = 39.65225825175571	& PSNR = 32.14812434829716
Level = 16	&	MSE = 10.796759136959697	& PSNR = 37.79786947988076
Level = 32	&	MSE = 2.1886541422155954	& PSNR = 44.729032224130535
Level = 64	&	MSE = 0.4045796103752088	& PSNR = 52.06076369048712

با بالاتر رفتن مقدار Level MSE کمتر و مقدار PSNR بیشتر می شود.

```
In [ ]: import cv2
import numpy as np
from sklearn.cluster import KMeans
import math
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from google.colab import drive
```

```
In [ ]: drive.mount('/content/drive')
```

Mounted at /content/drive

```
In [ ]: img_bgr = cv2.imread("/content/drive/MyDrive/Images/6/Lena.bmp")
img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
img_gray = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2GRAY)
```

```
In [ ]: def quantization (image, level):
    k_means = KMeans(n_clusters=level)
    k_means.fit(image.reshape((-1,1)))

    labels = k_means.labels_
    quantized_image = k_means.cluster_centers_[labels]

    return quantized_image.reshape((image.shape))
```

```
In [ ]: def psnr(original, contrast):
    mse = np.mean((original - contrast) ** 2)

    if mse == 0:
        return 100
    PIXEL_MAX = 255.0

    PSNR = 20 * math.log10(PIXEL_MAX / math.sqrt(mse))

    return PSNR
```

```
In [ ]: levels = [8, 16, 32, 64]

quantized_images_mse = np.zeros(len(levels))
quantized_images_psnr = np.zeros(len(levels))

fig, plot = plt.subplots(int(len(levels) // 2) + 1, 2, figsize = (10, 5 * (int(len(levels) // 2) + 1)))

plot[0][0].imshow(img_rgb)
plot[0][0].set_title("Image: RGB")

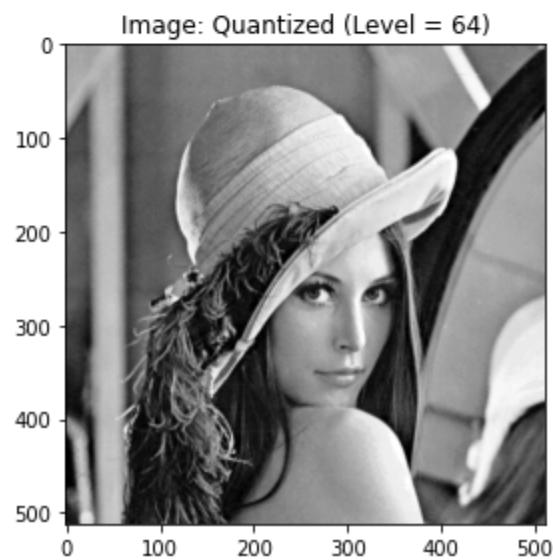
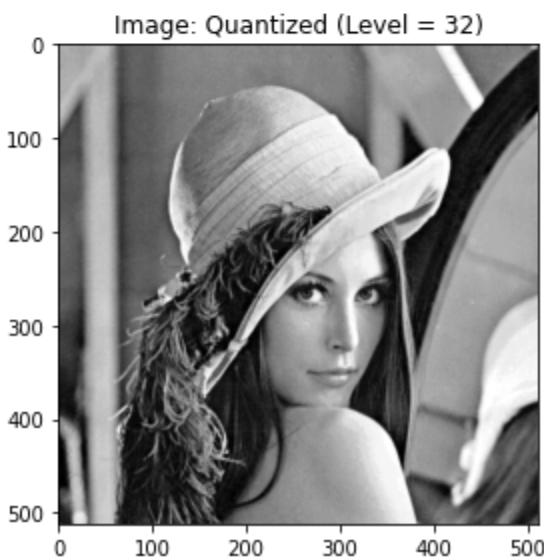
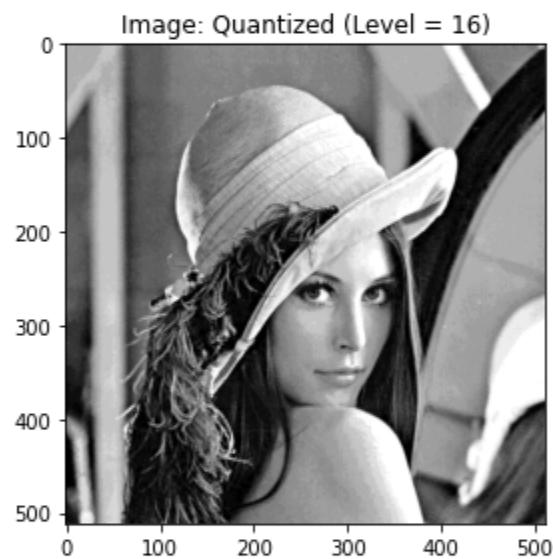
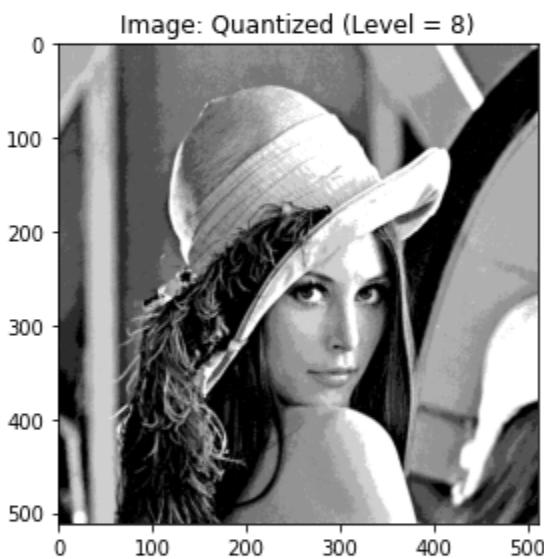
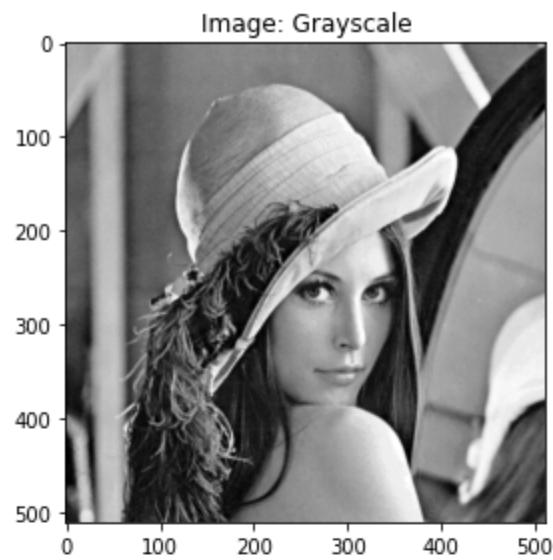
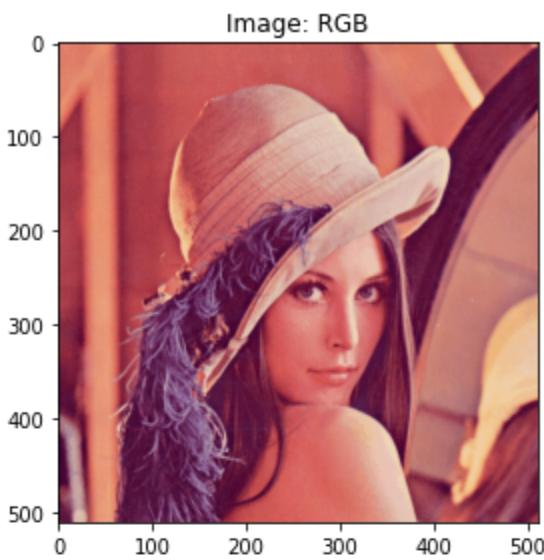
plot[0][1].imshow(img_gray, cmap='gray')
plot[0][1].set_title("Image: Grayscale")

for level_index in range (2, len(levels) + 2):

    quantized_image = quantization(img_gray, levels[level_index - 2])

    quantized_images_mse[level_index - 2] = mean_squared_error(img_gray, quantized_image)
    quantized_images_psnr[level_index - 2] = psnr(img_gray, quantized_image)

    plot[int(level_index // 2)][int(level_index % 2)].imshow(quantized_image, cmap='gray')
    plot[int(level_index // 2)][int(level_index % 2)].set_title("Image: Quantized (Level = " + str(levels[level_index - 2]) + ")")
```



```
In [ ]: print("Quantized Image MSE & PSNR")
for level_index in range (len(levels)):
    print("Level = " + str(levels[level_index]) + " & MSE = " + str(quantized_images_mse[level_index]) + " & PSNR = " + str(quantized_images_psnr[level_index]))
```

```
Quantized Image MSE & PSNR
Level = 8 & MSE = 39.65225825175571 & PSNR = 32.14812434829716
Level = 16 & MSE = 10.796759136959697 & PSNR = 37.79786947988076
Level = 32 & MSE = 2.1886541422155954 & PSNR = 44.729032224130535
Level = 64 & MSE = 0.4045796103752088 & PSNR = 52.06076369048712
```

تمرين ۶ سوال ۱-۲

چکیده

در اين تمرين به بررسی Color Space جديد مى پردازيم.

مقدمه

است که در ادامه به توضیح آنها می پردازیم. Adobe RGB و RG Chromaticity Color Space های انتخاب شده YIQ است که در ادامه به توضیح آنها می پردازیم.

منبع

شرح نتایج

:YIQ

Y به معنای Luma Information و Q به معنای Chrominance Information مقادير I را مشخص می کنند. در NTSCTV استفاده می شوند و در كشور های آمريكا و ژاپن از آنها استفاده می شود. همچنان از آن برای Human Color Response استفاده می شود.

RGB به YIQ تبدیل

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} \approx \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.5959 & -0.2746 & -0.3213 \\ 0.2115 & -0.5227 & 0.3112 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

تبدیل YIQ به RGB

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0.956 & 0.619 \\ 1 & -0.272 & -0.647 \\ 1 & -1.106 & 1.703 \end{bmatrix} \begin{bmatrix} Y \\ I \\ Q \end{bmatrix}$$

می توان Channel های مختلف آن را مشاهده کرد:



RGB Chromaticity

شامل دو بعد نرمالایز شده RGB بوده که در Computer Vision Application ها از آن استفاده می شود.

تبدیل rgb در RG Chrimaticity به

$$r = \frac{R}{R + G + B}$$

$$g = \frac{G}{R + G + B}$$

$$b = \frac{B}{R + G + B}$$

$$r + g + b = 1$$

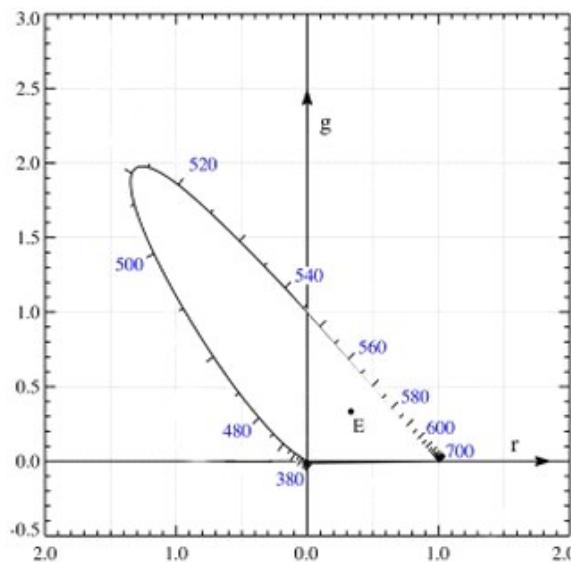
تبدیل rgb به RGB در RG Chrimaticity

$$R = \frac{rG}{g}$$

$$G = G$$

$$B = \frac{(1 - r - g)G}{g}$$

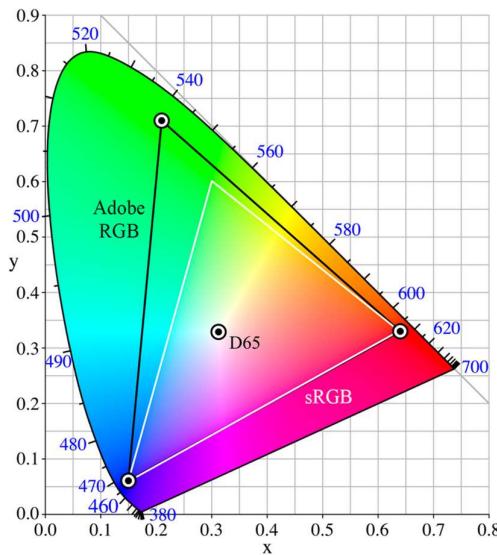
نمودار Chromaticity



:Adobe RGB

توسط Adobe Ink ارائه شده و این Color Space بهبود یافته Gamut sRGB بوده و در تلاش است تا رنگ های CMYK موجود در پرینتر های رنگی را با رنگ های اولیه RGB پیاده سازی کند.

Chromaticity
نمودار



تمرین ۶ سوال ۲-۲

چکیده

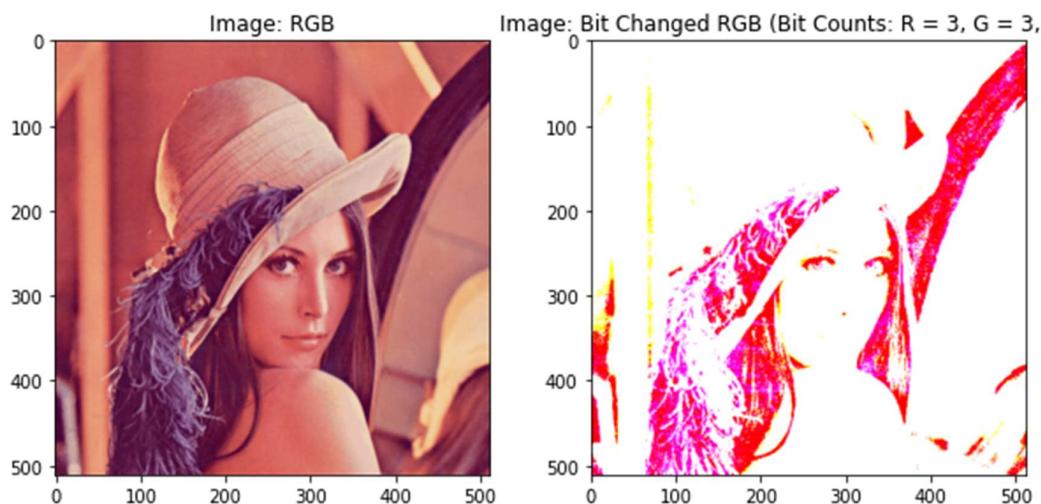
در این تمرین به تغییر تعداد بیت های ارائه شده توسط هر رنگ پرداخته و تصویر رنگی جدید را بازسازی می کنیم و تفاوت رنگ های آن با تصویر اصلی را مشاهده می کنیم.

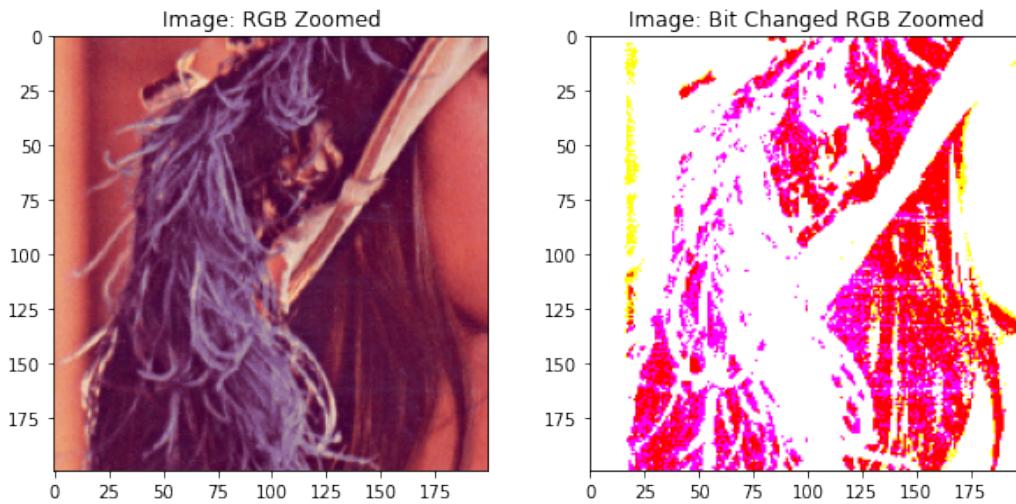
مقدمه

هر رنگ در تصویر با ۸ بیت نمایش داده می شود ولی در این تمرین این تعداد بیت را برای هر Channel به صورت جداگانه تغییر داده و تصویر جدید را بازسازی می کنیم.
برای این کار ابتدا تصویر رنگی را می خوانیم و هر Channel را به صورت جداگانه دریافت می کنیم. سپس هر کدام از Channel ها را به مقدار $8 - \text{NewBitCount}$ شیفت می دهیم. در نهایت با کنار هم گذاشتن Channel های جدید، تصویر را بازسازی می کنیم.

شرح نتایج

در این بخش تصویر اصلی را در کنار تصویر بازسازی شده با تغییر بیت (۳ بیت برای R، ۳ بیت برای G و ۲ بیت برای B) می بینیم.





با زوم کردن روی بخشی از تصویر، تفاوت رنگ ها را به صورت واضح تر می توانیم مشاهده کنیم. چون ۳ مقدار در تشکیل هر رنگ دخیل هستند، با تغییر هر سه مقدار دیگر رنگ جدید به رنگ اولیه شباهتی نداشته و تصویر نیز کیفیت خود را به کل از دست می دهد.

```
In [ ]: import cv2
import numpy as np
from sklearn.cluster import KMeans
import math
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from google.colab import drive
```

```
In [ ]: drive.mount('/content/drive')
```

Mounted at /content/drive

```
In [ ]: img_bgr = cv2.imread("/content/drive/MyDrive/Images/6/Lena.bmp")
img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
```

```
In [ ]: def color_image_change_bits (image, r_new_bits, g_new_bits, b_new_bits):
    image_r, image_g, image_b = cv2.split(image)

    bit_changed_image_r = image_r >> (8 - r_new_bits)
    bit_changed_image_g = image_g >> (8 - g_new_bits)
    bit_changed_image_b = image_b >> (8 - b_new_bits)

    bit_changed_image = np.zeros((512,512,3))

    bit_changed_image[:, :, 0] = bit_changed_image_r
    bit_changed_image[:, :, 1] = bit_changed_image_g
    bit_changed_image[:, :, 2] = bit_changed_image_b

    return bit_changed_image
```

```
In [ ]: r_new_bits = 3
g_new_bits = 3
b_new_bits = 2

bit_changed_image = color_image_change_bits(img_rgb, r_new_bits, g_new_bits, b_new_bits)

fig, plot = plt.subplots(2, 2, figsize = (10, 10))

plot[0][0].imshow(img_rgb)
plot[0][0].set_title("Image: RGB")

plot[0][1].imshow(bit_changed_image)
plot[0][1].set_title("Image: Bit Changed RGB (Bit Counts: R = " + str(r_new_bits) + ", G = " + str(g_new_bits) + ", B = " + str(b_new_bits) + ")")

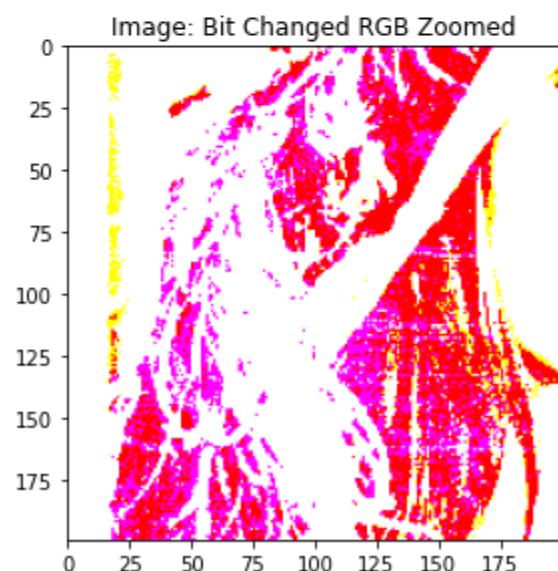
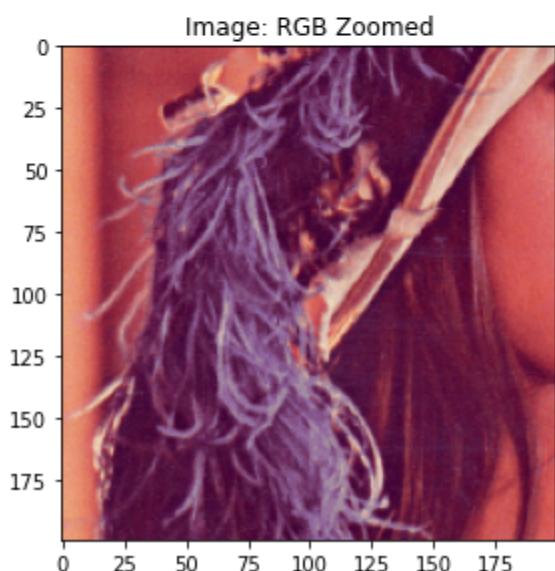
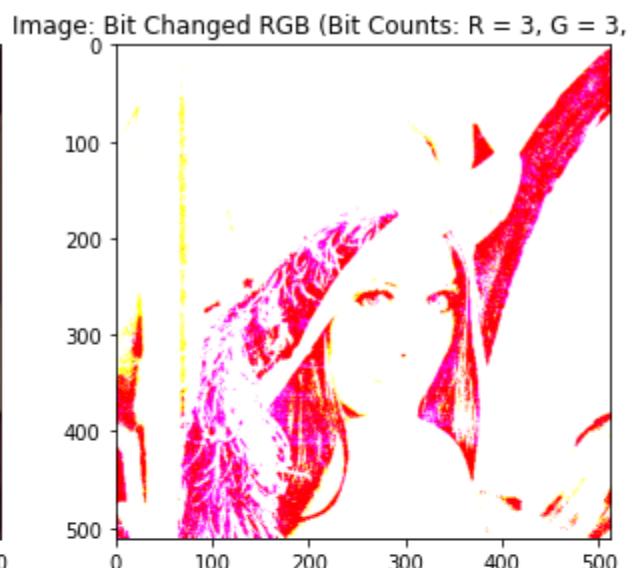
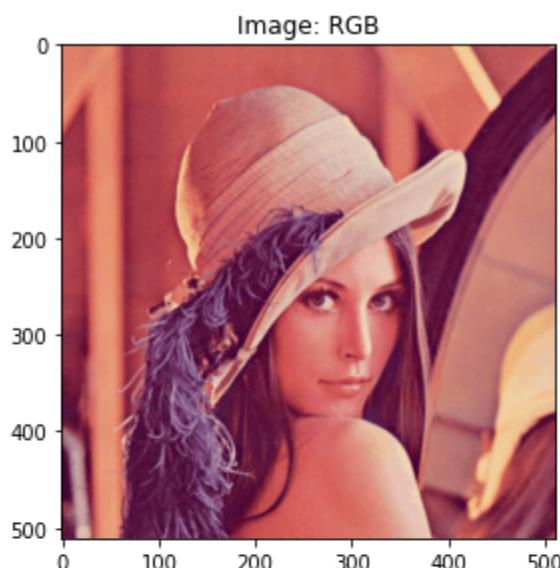
plot[1][0].imshow(img_rgb[250:450, 50:250])
plot[1][0].set_title("Image: RGB Zoomed")

plot[1][1].imshow(bit_changed_image[250:450, 50:250])
plot[1][1].set_title("Image: Bit Changed RGB Zoomed")
```

WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

Out[]: Text(0.5, 1.0, 'Image: Bit Changed RGB Zoomed')



تمرین ۶ سوال ۲-۳

چکیده

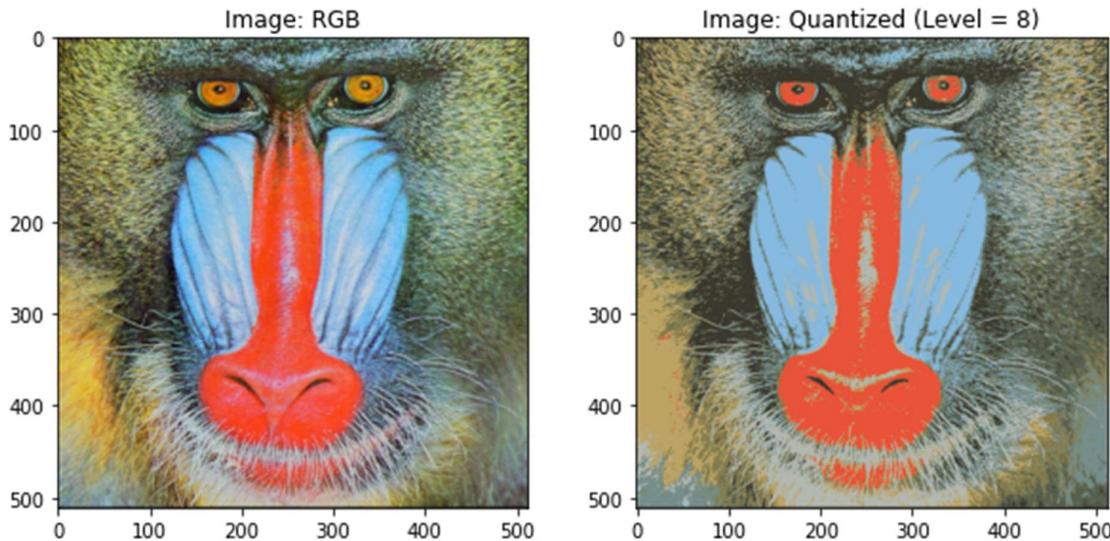
در این تمرین به Quantize کردن تصویر رنگی پرداخته و بهترین کیفیت تصویر از دیدگاه رنگ را برای نتیجه انتخاب می کنیم.

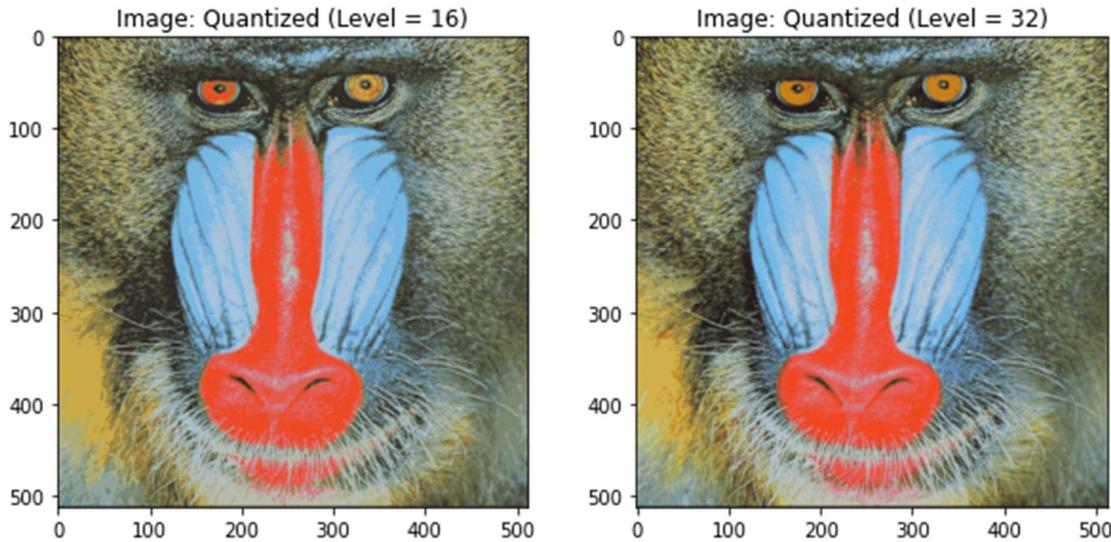
مقدمه

برای Quantize کردن تصویر رنگی باید به موضوعی توجه کنیم. این تصاویر را بر خلاف تصاویر سیاه و سفید نمی توانیم با تغییر هر Channel و کنار هم گذاشتنشان و تشکیل تصویر جدید Quantize کنیم. در تصاویر رنگی، اگر هر Channel را به مدار Level مشخص شده Quantize کنیم، به جای نتیجه رنگ ها به تعداد رنگ ها را به تعداد Level \times Level \times Level خواهیم داشت چراکه هر Channel می تواند در کنار ۲ مقدار دیگر یک مقدار جدید رنگی حاصل کند و این عملیات را نمی توان روی هر Channel به صورت جداگانه انجام داد. بنابراین از K-Menas Clustering برای Quantize کردن استفاده می کنیم و هر ۳ چنل را به صورت همزمان به الگوریتم می دهیم. در این حالت هر رنگ ساخته شده از ۳ رنگ در نزدیک ترین دسته قرار گرفته و با Label ها و مرکز Cluster ها، رنگ های جدید را بدست آورده و تصویر را نمایش می دهیم.

شرح نتایج

در این قسمت تصویر اصلی را در کنار تصاویر Quantize شده می بینیم.





همانطور که مشاهده می شود، رنگ ها حتی در Level 8 نیز بسیار شبیه تصویر بوده (بجز قسمت هایی مانند چشم ها که به طور کل قرمز دیده می شوند) ولی بسیاری از جزئیات تصویر مثل مو ها از بین رفته است. در Level های بالاتر می بینیم که برخی رنگ ها مثل رنگ چشم ها کم کم برطرف شده و جزئیات بیشتری دیده می شوند.
مقادیر MSE و PSNR را نیز می توانیم در زیر مشاهده کنیم.

Quantized Image MSE & PSNR

Level = 8 & MSE = 86.48134231567383	& PSNR = 28.76157938991043
Level = 16 & MSE = 77.58442306518555	& PSNR = 29.233058258971596
Level = 32 & MSE = 66.68076705932617	& PSNR = 29.890797737953775

همانطوری که مشاهده می شود، با افزایش Level ها مقدار MSE افزایش و PSNR کاهش می یابد. (تغییرات بین Level ها نسبتاً جزئی است و این مقادیر با تغییر Level آنقدر تغییر نکرده است).
اگر مبنای انتخاب تصویر، بافنون آن روی یک قالی باشد، به نظر من هم کم بودن جزئیات برای راحت تر بودن کار بافنده مناسب تر است و بنابراین انتخاب Level 8 بهتر است چراکه با اینکه تعداد رنگ های خیلی کمتری استفاده می شود باز هم تصویر مشابه تصویر اولیه بوده و گویای یا مفهوم است و منظور تصویر عوض نشده است. (نتیجه انتخاب در کاربرد های دیگر قطعاً متفاوت است).

```
In [ ]: import cv2
import numpy as np
from sklearn.cluster import KMeans
import math
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from google.colab import drive
```

```
In [ ]: drive.mount('/content/drive')
```

```
In [ ]: img_bgr = cv2.imread("/content/drive/MyDrive/Images/6/Baboon.bmp")
img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
```

```
In [ ]: def quantization (image, level):
    k_means = KMeans(n_clusters=level)
    k_means.fit(image.reshape((-1,3)))

    labels = k_means.labels_
    quantized_image = k_means.cluster_centers_[labels]

    return quantized_image.reshape((512,512,3)).astype('uint8')
```

```
In [ ]: def psnr(original, contrast):
    mse = np.mean((original - contrast) ** 2)

    if mse == 0:
        return 100
    PIXEL_MAX = 255.0

    PSNR = 20 * math.log10(PIXEL_MAX / math.sqrt(mse))

    return PSNR
```

```
In [ ]: levels = [8, 16, 32]

quantized_images_mse = np.zeros(len(levels))
quantized_images_psnr = np.zeros(len(levels))

fig, plot = plt.subplots(int(len(levels) // 2) + 1, 2, figsize = (10, 5 * (int(len(levels) // 2) + 1)))

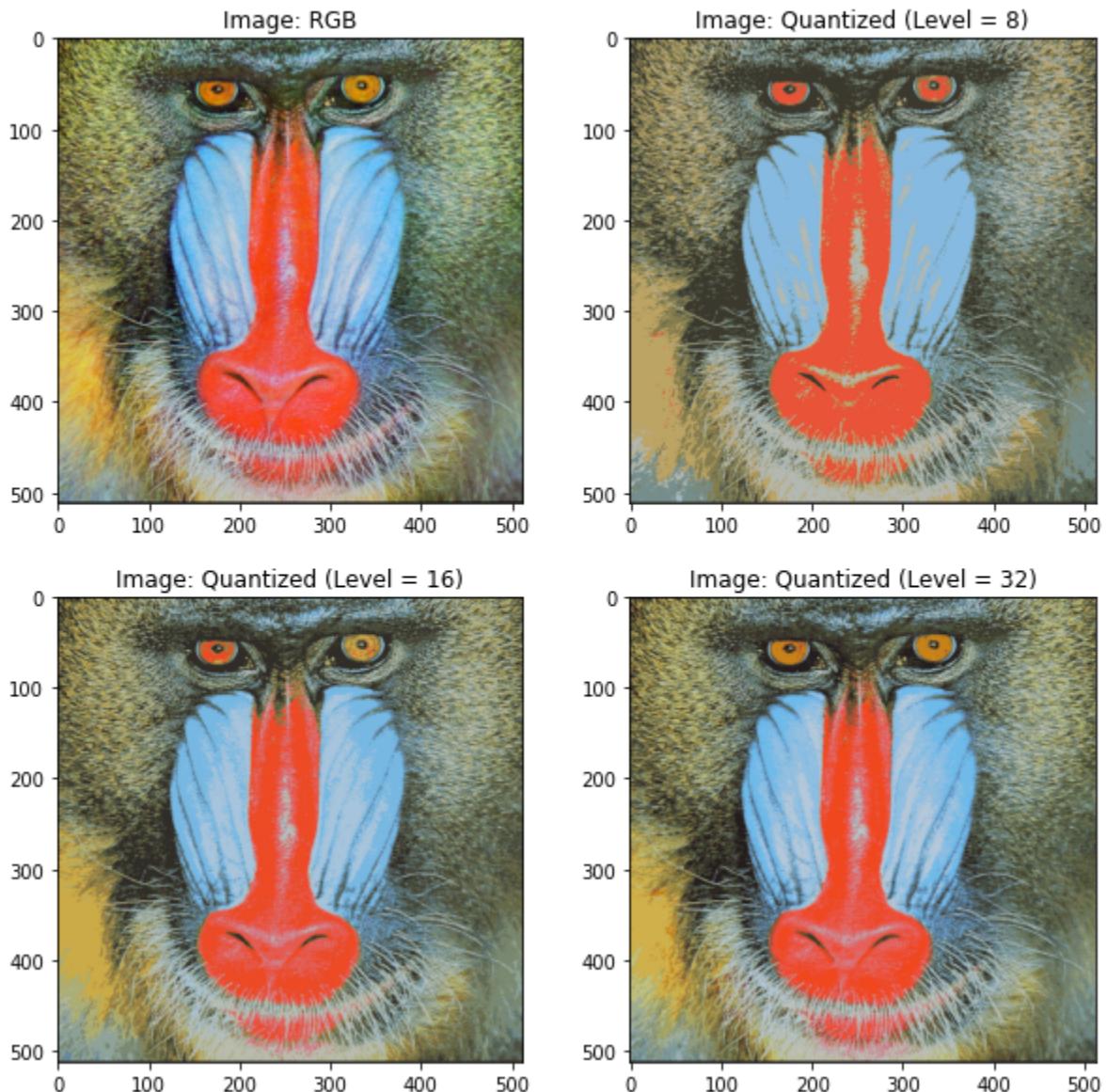
plot[0][0].imshow(img_rgb)
plot[0][0].set_title("Image: RGB")

for level_index in range(1, len(levels) + 1):

    quantized_image = quantization(img_rgb, levels[level_index - 1])

    quantized_images_mse[level_index - 1] = mean_squared_error(img_rgb.reshape((-1,3)), quantized_image.reshape((-1,3)))
    quantized_images_psnr[level_index - 1] = psnr(img_rgb.reshape((-1,3)), quantized_image.reshape((-1,3)))

    plot[int(level_index // 2)][int(level_index % 2)].imshow(quantized_image, cmap='gray')
    plot[int(level_index // 2)][int(level_index % 2)].set_title("Image: Quantized  
(Level = " + str(levels[level_index - 1]) + ")")
```



```
In [ ]: print("Quantized Image MSE & PSNR")
for level_index in range (len(levels)):
    print("Level = " + str(levels[level_index]) + " & MSE = " + str(quantized_images_mse[level_index]) + " & PSNR = " + str(quantized_images_psnr[level_index]))
```

```
Quantized Image MSE & PSNR
Level = 8 & MSE = 86.48134231567383 & PSNR = 28.76157938991043
Level = 16 & MSE = 77.58442306518555 & PSNR = 29.233058258971596
Level = 32 & MSE = 66.68076705932617 & PSNR = 29.890797737953775
```