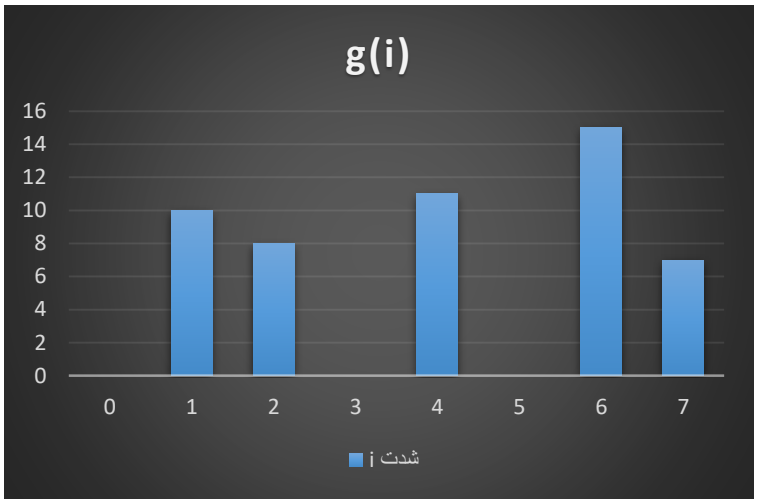
-1

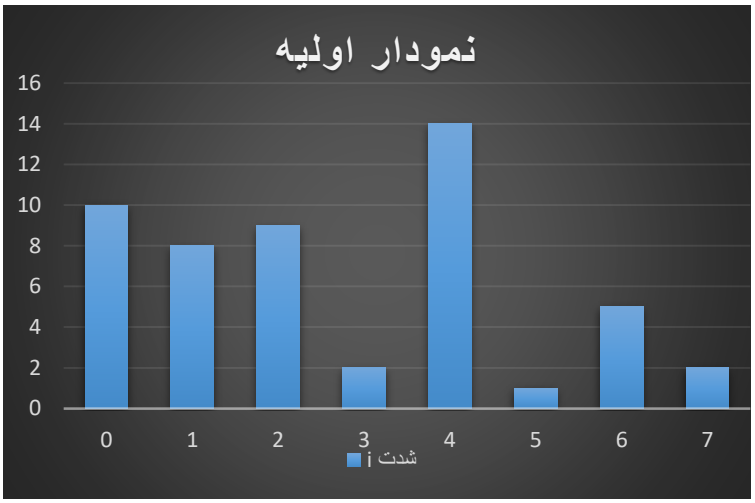
برای مسطح سازی هیستوگرام، باید تابع تبدیل شدت رنگ را بیابیم که تعداد پیکسل ها را برای هر شدت رنگ به طور مساوی توزیع کند. برای این منظور، می‌توانیم از تابع توزیع تجمعی (CDF) استفاده کنیم.

با توجه به تابعCDF ، می‌توانیم تابع تبدیل شدت مناسب را بیابیم.

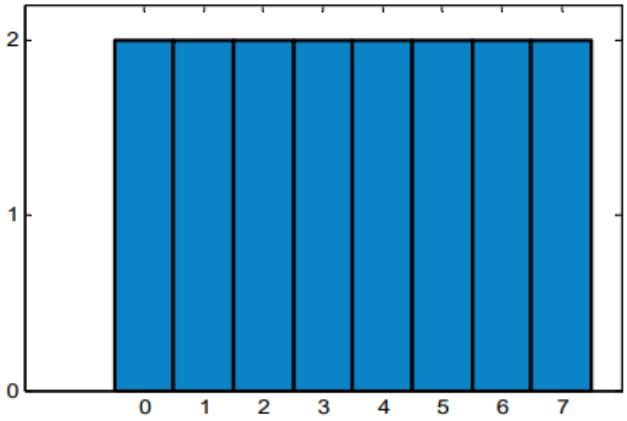
شدت رنگ	0	1	2	3	4	5	6	7
تعداد پیکسل	10	8	9	2	14	1	5	2
P(i)	10/51	8/51	9/51	2/51	14/51	1/51	5/51	2/51
S(i)	10/51	18/51	27/51	29/51	43/51	44/51	49/51	51/51
G(i)	1	2	4	4	6	6	7	7
تعداد جدید	0	10	8	0	11	0	15	7

در نهایت، می‌توانیم تصویر جدید را با استفاده از جدول مقادیر شدت جدید بسازیم.





همانگونه که ملاحظه میشود، هیستوگرام در تصویر دوم به هیستوگرام یکنواخت نزدیکتر است .



نمونه هیستوگرام یکنواخت

-2

برای پیاده‌سازی تابع مسطح سازی هیستوگرام با استفاده از کتابخانه numpy و توابع استاندارد پایتون، ابتدا باید تصویر ورودی را به یک آرایه numpy تبدیل کنیم.

سپس با استفاده از تابع **numpy.histogram** می‌توانیم هیستوگرام تصویر را بدست آوریم.

سپس با تقسیم هر بین‌بازه به تعداد پیکسل‌های آن بین‌بازه، هیستوگرام را مسطح می‌کنیم.

کد پیاده سازی شده به صورت زیر است:

```
import numpy as np


def flatten_histogram(image):
    # convert image to numpy array
    image_array = np.array(image)
    # calculate histogram of the image
    hist, bin_edges = np.histogram(image_array, bins=256, range=(0, 256))
    # normalize the histogram
    hist_flat = hist / np.sum(hist)


    return hist_flat
```

در این تابع، تصویر ورودی به صورت یک آرایه **numpy** با نام **image_array** تبدیل شده است.

سپس با استفاده از تابع **numpy.histogram**، هیستوگرام تصویر با **(bins=256** تعداد بین‌بازه‌ها) و **range=(0, 256)** (حداکثر و حداقل مقادیر پیکسل) محاسبه شده است. نتیجه‌ی این محاسبه در متغیرهای **hist** و **bin_edges** ذخیره شده است.

سپس با تقسیم هر بین‌بازه به تعداد پیکسل‌های آن بین‌بازه، هیستوگرام را مسطح می‌کنیم و در متغیر **hist_flat**ذخیره می‌کنیم. در نهایت، مقدار **hist_flat**به عنوان خروجی تابع بازگردانده می‌شود.

-3

$$\text{DoF} := o_2^* - o_1^* = \frac{2of^2c^*N\,(o-f)}{f^4 - c^{*2}N^2(o-f)^2}$$

عدد **f** : **1.8**

فاصله کانونی: **50** میلیمتر

فاصله شی : **2** متر

دقت دوربین: **0.03** میلیمتر

$$\frac{2 * 2 * (50 * 10^{-3})^2 * 0.03 * 10^{-3} * 1.8 * (2 - 50 * 10^{-3})}{(50 * 10^{-3})^4 - (0.03 * 10^{-3})^2 * 1.8^2 * (2 - 50 * 10^{-3})^2}$$

با جایگذاری مقادیردر فرمول کسر بالا به وجود میاید و بعد از انجام محاسبات عدد **0.16877** بدست میاید.

Dof = 0.16877 m

-4

تکنیک مسطح سازی هیستوگرام به این صورت است که با توجه به توزیع تعداد پیکسل‌ها در هر شدت رنگ، یک تابع تبدیل از مقادیر اولیه شدت رنگ به مقادیر جدید آن‌ها به نحوی تعریف می‌شود که هیستوگرام جدید برای تصویر، به‌جای توزیع عادی تعداد پیکسل‌ها در محدوده مختلف شدت رنگ، به‌صورت مسطح (یعنی همه‌ی بن‌های هیستوگرام با ارتفاع یکسان) شود.

اما این تکنیک به‌شدت به توزیع تعداد پیکسل‌ها در هر محدوده شدت رنگی وارد شده برای تصویر وابسته است. بنابراین، اگر توزیع پیکسل‌ها به‌طور یکنوا و یا شبه یکنوا در امتداد محدوده‌ی شدت رنگی باشد، هیستوگرام پس از مسطح سازی به‌شدت به‌صورت مسطح خواهد بود. اما در صورتی که توزیع پیکسل‌ها به‌صورت نامنظم در امتداد محدوده‌ی شدت رنگی باشد، مسطح سازی هیستوگرام به‌شدت ممکن است توزیع پیکسل‌ها را در هنگامی که به‌صورت مسطح شده‌اند، تحت تأثیر قرار دهد. در نتیجه هیستوگرام نهایی، به‌طور مسطح شده‌ای نخواهد بود.

-5

یرای پیاده سازی توابع خطی، لگاریتمی و توانی، ابتدا باید فرمول کلی هر یک از آنها را بررسی کرده و سپس آنها را به شکل تابع پیاده سازی کنیم. در ادامه، توابع خطی، لگاریتمی و توانی را به ترتیب پیاده سازی می کنیم.

تابع خطی:

تابع خطی به شکل زیر تعریف می شود:

f(x) = a *x + b

در این فرمول، **a** و **b** پارامترهای تابع خطی هستند.

حال تابع خطی را به شکل تابع پایتون پیاده سازی می کنیم

```
def linear_transform(image, a, b):
    """
     apply linear transform to the input image with the given parameters a and b
    """
    return a * image + b
```

در این تابع، **image** تصویری است که می خواهیم بر روی آن تبدیل خطی اعمال کنیم و **a** و **b** پارامترهای تابع خطی هستند که در ورودی تابع دریافت می شوند.

تابع لگاریتمی

تابع لگاریتمی به شکل زیر تعریف می شود:

$$f(x) = a * \log_b(x + 1)$$

در این فرمول،**a** و **b** پارامترهای تابع لگاریتمی هستند.

حال تابع لگاریتمی را به شکل تابع پایتون پیاده سازی می کنیم:

```
import numpy as np
def logarithmic_transform(image, a, b):
    """
    apply logarithmic transform to the input image with the given parameters a and b
    """
    return a * np.log10(image + 1) / np.log10(b)
```

در این تابع، **image** تصویری است که می خواهیم بر روی آن تبدیل لگاریتمی اعمال کنیم و **a** و **b** پارامترهای تابع لگاریتمی هستند که در ورودی تابع دریافت می شوند. برای محاسبه لگاریتم به شکل پایتونی از تابع **np.log10** استفاده شده است.

تابع توانی به شکل زیر تعریف میشود:

f(x) = a * x^b

در این فرمول، **a** و **b** پارامترهای تابع توانی هستند.

حال تابع توانی را به شکل تابع پایتون پیاده سازی می کنیم.

```
def power_transform(image, a, b):
    """
    apply power transform to the input image with the given parameters a and b
    """
    return a * np.power(image, b)
```

در این تابع، **image** تصویری است که می خواهیم بر روی آن تبدیل توانی اعمال کنیم و **a** و **b** پارامترهای تابع توانی هستند که در ورودی تابع دریافت می شوند. برای محاسبه توان به شکل پایتونی از تابع **np.power** استفاده شده است.

حال برای نمایش تصاویر و نتایج تبدیل با استفاده از**matplotlib** ، تصاویر را با استفاده از تابع **imread** از کتابخانه **matplotlib** می خوانیم و سپس توابع تبدیل را بر روی آنها اعمال می کنیم و تصاویر تبدیل شده را با استفاده از تابع **imshow** نمایش می دهیم.

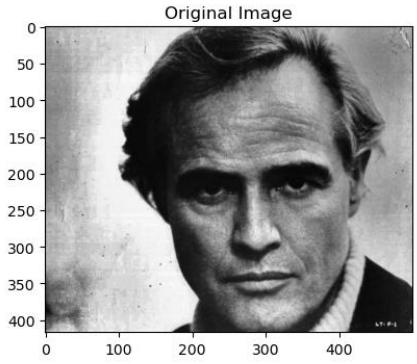
```
import matplotlib.pyplot as plt
# read the image
image = plt.imread('image.jpg')
# apply linear transform with parameters a=1, b=0
linear_image = linear_transform(image, 1, 0)
# apply logarithmic transform with parameters a=1, b=10
logarithmic_image = logarithmic_transform(image, 1, 10)
# apply power transform with parameters a=1, b=0.5
power_image = power_transform(image, 1, 0.5)
# show the original and transformed images
fig, axs = plt.subplots(2, 2, figsize=(10,10))
axs[0, 0].imshow(image)
axs[0, 0].set_title('Original Image')
axs[0, 1].imshow(linear_image)
axs[0, 1].set_title('Linear Transform')
axs[1, 0].imshow(logarithmic_image)
axs[1, 0].set_title('Logarithmic Transform')
axs[1, 1].imshow(power_image)
```

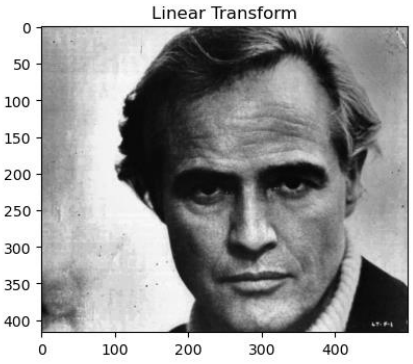
axs[1, 1].set_title('Power Transform')

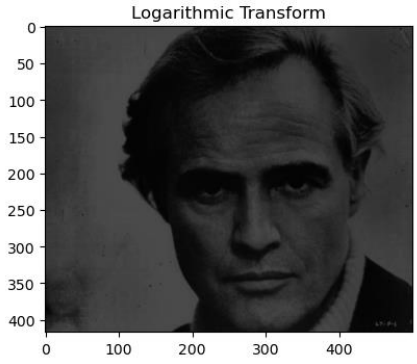
plt.show()

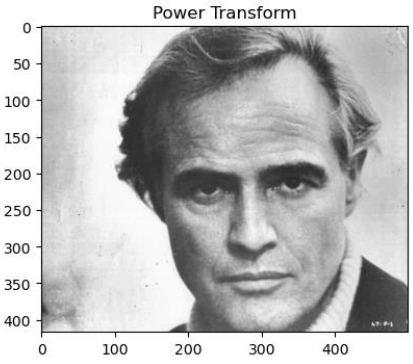
در این برنامه، تصویری با نام **'image.jpg'** در همان فولدر برنامه قرار دارد و با استفاده از توابع تبدیل، تصویر اصلی به تصاویر تبدیل شده تبدیل می شود. در نهایت، تصاویر اصلی و تصاویر تبدیل شده با استفاده از تابع **imshow** در نمودارهای جداگانه در نظر گرفته شده و با استفاده از تابع**set_title** ، برچسب هر تصویر نیز قرار داده شده است. در نهایت، با استفاده از تابع**show** ، تصاویر روی نمودار نمایش داده می شوند.

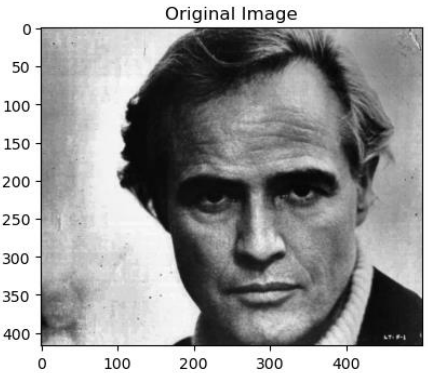
```
import numpy as np
import matplotlib.pyplot as plt

def linear_transform(image, a, b):
    """
    apply linear transform to the input image with the given parameters a and b
    """
    image = image.astype('float') / 255.0
    return a * image + b

def logarithmic_transform(image, a, b):
    """
    apply logarithmic transform to the input image with the given parameters a and b
    """
    image = image.astype('float') / 255.0
    return a * np.log10(image + 1) / np.log10(b)

def power_transform(image, a, b):
    """
    apply power transform to the input image with the given parameters a and b
    """
    image = image.astype('float') / 255.0
    return a * np.power(image, b)

# read the image
image = plt.imread('image.jpg')
# apply linear transform with parameters
linear_image = linear_transform(image, 1, 0)
# apply logarithmic transform with parameters
logarithmic_image = logarithmic_transform(image, 1, 10)
# apply power transform with parameters
power_image = power_transform(image, 1, 0.5)
# show the original and transformed images
fig, axs = plt.subplots(2, 2, figsize=(10,10))
axs[0, 0].imshow(image)
axs[0, 0].set_title('Original Image')
axs[0, 1].imshow(linear_image)
axs[0, 1].set_title('Linear Transform')
axs[1, 0].imshow(logarithmic_image)
axs[1, 0].set_title('Logarithmic Transform')
axs[1, 1].imshow(power_image)
axs[1, 1].set_title('Power Transform')
plt.show()
```

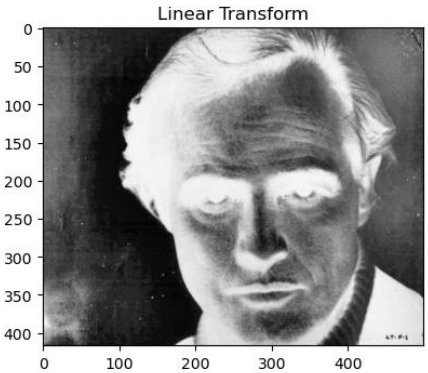

Original Image

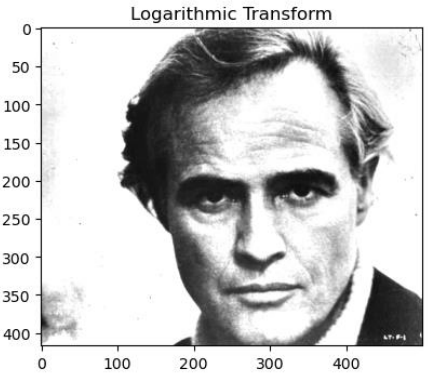

Linear Transform

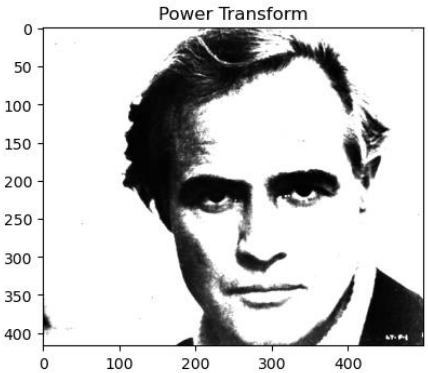

Logarithmic Transform


Power Transform


Original Image


Linear Transform


Logarithmic Transform


Power Transform