هیستوگرام تصویر:

هیستوگرام تصویر یک نمودار آماری است که توزیع شدت روشنایی یا رنگ پیکسل‌های یک تصویر را نمایش می‌دهد. در این نمودار، محور افقی سطوح شدت روشنایی (مثلاً از ۰ تا ۲۵۵ در تصاویر خاکستری) و محور عمودی تعداد پیکسل‌هایی را نشان می‌دهد که در هر سطح شدت قرار دارند. به کمک هیستوگرام می‌توان ویژگی‌های مهم تصویر مانند روشنایی کلی، کنتراست و میزان پراکندگی شدت‌ها را تحلیل کرد. برای مثال، اگر هیستوگرام بیشتر در سمت چپ متمرکز باشد، تصویر تیره است و اگر در سمت راست باشد، تصویر روشن است. همچنین، گستردگی هیستوگرام نشان‌دهنده‌ی میزان کنتراست تصویر است؛ هرچه گسترده‌تر باشد، کنتراست بالاتر است. این ابزار در پردازش تصویر کاربردهای زیادی دارد، از جمله بهبود کیفیت تصویر، آستانه‌گذاری و تشخیص الگو.

یک هیستوگرام ساده جهت یادآوری:

```
numbers_list = np.array([3,2,3,5,1,9,8,8,4,6])

plt.figure(figsize=[10,3])
plt.subplot(121);plt.hist(numbers_list);plt.title('Histogram')
plt.subplot(122);plt.hist(numbers_list,4,color='green');plt.title('Histogram with 8 bins')
```

این کد یک آرایه‌ی عددی به نام `numbers_list` تعریف می‌کند و سپس با استفاده از کتابخانه‌ی `matplotlib` دو هیستوگرام از این داده‌ها رسم می‌نماید. در خط اول، آرایه شامل ۱۰ عدد ذخیره می‌شود. سپس یک شکل با اندازه‌ی افقی بزرگ‌تر (`figsize=[10,3]`) ساخته شده و به کمک دستور `subplot`، این شکل به دو بخش تقسیم می‌گردد. در بخش سمت چپ(`subplot(121)`) ، هیستوگرام با تعداد bin پیش‌فرض رسم می‌شود و عنوان آن "Histogram" گذاشته شده است. در بخش سمت راست(`subplot(122)`) ، دوباره هیستوگرام داده‌ها نمایش داده می‌شود اما این بار تعدادbin ها به صورت دستی روی مقدار 4تنظیم شده و رنگ ستون‌ها نیز سبز تعیین شده است. در نتیجه، کد امکان مقایسه‌ی نحوه‌ی نمایش داده‌ها با تعداد bin مختلف را فراهم می‌کند.

اکنون که با نحوه کار هیستوگرام آشنا شدید بیاید نگاهی به هیستوگرام یک تصویر بیندازیم:

```
img = cv2.imread('images/bird.jpg',0)
if img is None:
    print('Error')

plt.figure(figsize=[10,3])
plt.subplot(121);plt.hist(img.ravel(),256, color = 'green');plt.title('Gray image histogram')
plt.subplot(122);plt.hist(img.ravel(),254,color = 'red',histtype='step');plt.title('Step
Histogram')
```

این کد یک تصویر خاکستری را بارگذاری کرده و سپس دو نوع هیستوگرام مختلف از آن رسم می‌کند. ابتدا تصویر با دستور `cv2.imread('images/bird.jpg',0)` خوانده می‌شود که پارامتر `0` به معنی خواندن تصویر به صورت سیاه‌وسفید است. اگر تصویر وجود نداشته باشد یا مسیر اشتباه باشد، پیام `'Error'` چاپ می‌شود. سپس یک شکل با اندازه‌ی افقی بیشتر ساخته شده و به دو بخش تقسیم می‌گردد. در بخش اول (`subplot(121)`) ، با استفاده از `plt.hist(img.ravel(),256,color='green')` یک هیستوگرام با **۲۵۶ سطح شدت خاکستری** رسم می‌شود که محور افقی شدت روشنایی (۰ تا ۲۵۵) و محور عمودی تعداد پیکسل‌ها را نشان می‌دهد. در بخش دوم (`subplot(122)`)، دوباره هیستوگرام تصویر رسم می‌شود اما این بار با **bin۲۵۴** ، رنگ قرمز، و نوع ترسیم `step` که به صورت خطوط پیوسته (بدون پرشدگی) نمایش داده می‌شود. در نهایت می‌توان مقایسه‌ای بین هیستوگرام معمولی و step هیستوگرام تصویر داشت.

نگاهی به متد ravel() :

متد `ravel()` در کتابخانه‌ی NumPy برای بازگرداندن یک آرایه‌ی چندبعدی به صورت یک آرایه‌ی یک‌بعدی (تخت‌شده) استفاده می‌شود. این متد در واقع یک نمای (view) از آرایه اصلی ایجاد می‌کند و در صورت امکان داده‌ها را بدون کپی‌کردن در حافظه خطی می‌سازد، بنابراین از نظر سرعت و مصرف حافظه بهینه است. تفاوت آن با متد `flatten()` در این است که `flatten()` همیشه یک کپی جدید می‌سازد، اما `ravel()` فقط زمانی که لازم باشد کپی می‌گیرد. در پردازش تصویر، معمولاً از `ravel()` برای تبدیل ماتریس دوبعدی تصویر به یک بردار یک‌بعدی استفاده می‌شود تا بتوان به راحتی روی آن عملیاتی مثل رسم هیستوگرام انجام داد.

در کد بالا ، تصویر خاکستری `img` یک آرایه‌ی دوبعدی از مقادیر شدت روشنایی پیکسل‌هاست (هر سطر و ستون یک پیکسل). اما تابع `plt.hist` انتظار دارد داده‌ها به صورت یک آرایه‌ی یک‌بعدی به آن داده شوند تا بتواند فرکانس وقوع هر مقدار را محاسبه کند. به همین دلیل از متد `ravel()` استفاده شده تا ماتریس دوبعدی تصویر به یک بردار یک‌بعدی تبدیل شود و همه‌ی مقادیر پیکسل‌ها پشت سر هم قرار گیرند. در نتیجه، `plt.hist(img.ravel(), 256, ...)` می‌تواند به درستی تعداد پیکسل‌ها در هر سطح روشنایی را بشمارد و هیستوگرام تصویر را رسم کند.

calcHist in open cv :

اما برای محاسبه هیستوگرام یک تصویر در کتابخانه open cv متدی تحت عنوان calcHist وجود دارد. تابع `cv2.calcHist()` در OpenCV برای محاسبه‌ی هیستوگرام تصویر به کار می‌رود و پنج آرگومان اصلی دارد:

1. **Images** لیستی از تصاویری که می‌خواهیم هیستوگرام آن‌ها را محاسبه کنیم. معمولاً به صورت `[img]` نوشته می‌شود.

2. **Channels** مشخص می‌کند هیستوگرام بر اساس کدام کانال محاسبه شود. مثلاً `[0]` برای کانال آبی در تصویر رنگی BGR یا تنها کانال خاکستری در تصاویر تک‌کاناله.
3. **Mask** یک ماسک اختیاری است که اگر داده شود، هیستوگرام فقط روی نواحی انتخاب‌شده (پیکسل‌هایی که مقدار ماسک آن‌ها غیر صفر است) محاسبه می‌شود. اگر `None` باشد، کل تصویر در نظر گرفته می‌شود.
4. **Bins** تعداد بخش‌هایی که محدوده‌ی شدت روشنایی به آن تقسیم می‌شود. مثلاً `[256]` برای ۲۵۶ سطح شدت.
5. **Range** بازه‌ی شدت مقادیر مورد نظر. برای تصاویر ۸ بیتی معمولاً `[0,256]` استفاده می‌شود.

ما می توانیم هیستوگرام تصویر قبلی را با استفاده از کتابخانه `open cv` نیز رسم کنیم

```
img = cv2.imread('images/bird.jpg',0)
if img is None:
    print('Error')
hist = cv2.calcHist([img],[0], None,[256], [0,256])
plt.figure(figsize=[5,3])
plt.plot(hist)
```

این کد یک تصویر خاکستری را بارگذاری کرده و در صورت موفقیت، با استفاده از تابع `cv2.calcHist` هیستوگرام شدت روشنایی آن را محاسبه می‌کند. در این تابع، تصویر ورودی به‌صورت لیست داده شده، کانال صفر (تنها کانال تصویر خاکستری) انتخاب شده، ماسک `None` است تا کل تصویر در نظر گرفته شود، تعداد bin برابر ۲۵۶ تعیین شده و بازه‌ی شدت روشنایی از ۰ تا ۲۵۶ مشخص شده است. خروجی این تابع آرایه‌ای است که در آن هر عنصر بیانگر تعداد پیکسل‌های تصویر با شدت روشنایی مشخصی است. در نهایت با استفاده از `plt.plot`، این مقادیر به‌صورت یک نمودار خطی رسم می‌شوند که محور افقی شدت روشنایی و محور عمودی فراوانی هر شدت را نمایش می‌دهد.

رسم هیستوگرام برای تصاویر رنگی:

ما همچنین می توانیم برای عکس های رنگی نیز از مفهوم هیستوگرام استفاده کنیم.

```
import matplotlib.pyplot as plt
import numpy as np
import cv2

# read images
dark_tones = cv2.imread('images/dark.png')
mid_tones = cv2.imread('images/mid.png')
light_tones = cv2.imread('images/light.png')

#plot imagea
```

```
plt.figure(figsize=[12,7])
plt.subplot(231);plt.imshow(dark_tones[...,::-1]);plt.title("dark tones image");
plt.subplot(232);plt.imshow(mid_tones[...,::-1]);plt.title("mid tones image");
plt.subplot(233);plt.imshow(light_tones[...,::-1]);plt.title("light tones image");

color = ('b', 'g', 'r')
#dark tones histogram
plt.subplot(234);
for i, col in enumerate(color):
    histogram = cv2.calcHist([dark_tones], [i], None, [256], [0, 256])
    plt.plot(histogram, color = col)

#mid tones histogram
plt.subplot(235);
for i, col in enumerate(color):
    histogram = cv2.calcHist([mid_tones], [i], None, [256], [0, 256])
    plt.plot(histogram, color = col)

#light tones histogram
plt.subplot(236);
for i, col in enumerate(color):
    histogram = cv2.calcHist([light_tones], [i], None, [256], [0, 256])
    plt.plot(histogram, color = col)
```

این کد سه تصویر با تن‌های مختلف (تیره، میانی و روشن) را از مسیر مشخص‌شده بارگذاری کرده و سپس آن‌ها را در یک شکل مشترک نمایش می‌دهد. به دلیل اینکه تصاویر در OpenCV به‌صورت BGR خوانده می‌شوند، هنگام نمایش با matplotlib ترتیب کانال‌ها با `[...,::-1]` معکوس شده تا تصویر به‌درستی در فضای رنگی RGB نشان داده شود. در بخش اول شکل، سه تصویر اصلی در کنار هم (به‌ترتیب تیره، میانی و روشن) نمایش داده می‌شوند تا امکان مقایسه‌ی بصری بین آن‌ها فراهم گردد.

در بخش دوم، برای هر تصویر یک هیستوگرام رنگی جداگانه رسم می‌شود. با استفاده از حلقه‌ای روی کانال‌های رنگی (آبی، سبز و قرمز)، تابع `cv2.calcHist` فراوانی شدت پیکسل‌ها را در بازه‌ی ۰ تا ۲۵۵ محاسبه می‌کند و سپس با رنگ متناظر ترسیم می‌گردد. در نتیجه، نمودارهای زیر هر تصویر نشان می‌دهند که شدت رنگ‌ها چگونه توزیع شده است. در تصویر تیره، هیستوگرام بیشتر در سمت مقادیر پایین متمرکز است، در تصویر میانی توزیع متعادل‌تری دارد و در تصویر روشن، بیشتر در سمت مقادیر بالا قرار می‌گیرد. این کار به‌خوبی تفاوت روشنایی و شدت رنگ در تصاویر مختلف را از دید آماری و بصری مشخص می‌سازد.

بررسی بیشتر حلقه for :

```
plt.subplot(236);
for i, col in enumerate(color):
    histogram = cv2.calcHist([light_tones], [i], None, [256], [0, 256])
    plt.plot(histogram, color = col)
```

این حلقه وظیفه‌ی رسم هیستوگرام سه کانال رنگی تصویر **dark_tones** را بر عهده دارد. ابتدا با `plt.subplot(234)` محل رسم نمودار در شکل مشخص می‌شود. سپس با دستور `for i, col in enumerate(color)` : روی سه رنگ اصلی یعنی آبی، سبز و قرمز پیمایش انجام می‌شود که در آن متغیر `i` شماره‌ی کانال (۰ برای آبی، ۱ برای سبز و ۲ برای قرمز) و متغیر `col` رنگ متناظر برای ترسیم نمودار است. در هر تکرار، تابع `cv2.calcHist` فراوانی شدت پیکسل‌ها در کانال انتخاب‌شده را در بازه‌ی ۰ تا ۲۵۵ محاسبه می‌کند و خروجی به صورت آرایه‌ای شامل تعداد پیکسل‌ها در هر سطح شدت ذخیره می‌شود. در نهایت، این مقادیر با `plt.plot(histogram, color=col)` به شکل نمودار خطی رسم شده و هر کانال با رنگ واقعی خودش نمایش داده می‌شود. نتیجه‌ی این حلقه یک هیستوگرام رنگی است که نشان می‌دهد در تصویر تیره، شدت هر کانال چطور توزیع شده است.

یکنواخت کردن هیستوگرام:

تابع `cv2.equalizeHist()` در OpenCV برای **بهبود کنتراست تصاویر خاکستری** به کار می‌رود. ایده‌ی اصلی این تابع استفاده از تکنیک **هیستوگرام مساوی‌سازی (Histogram Equalization)** است. در این روش، توزیع شدت روشنایی پیکسل‌ها در تصویر به گونه‌ای تغییر می‌کند که هیستوگرام تصویر تا حد امکان یکنواخت‌تر و گسترده‌تر شود. به عبارت دیگر، پیکسل‌های متراکم در یک محدوده‌ی شدت، بازتوزیع می‌شوند تا کل بازه‌ی شدت (۰ تا ۲۵۵) پوشش داده شود. نتیجه این است که بخش‌های تیره روشن‌تر می‌شوند و بخش‌های خیلی روشن نیز جزئیات بیشتری پیدا می‌کنند، در نتیجه کنتراست تصویر بهبود می‌یابد.

از این تابع معمولاً در تصاویری استفاده می‌شود که روشنایی یکنواختی ندارند یا کنتراست آن‌ها پایین است، مانند عکس‌هایی که در شرایط نور ضعیف گرفته شده‌اند. اما باید توجه داشت که `cv2.equalizeHist()` تنها روی **تصاویر تک‌کاناله (Grayscale)** کار می‌کند، زیرا مساوی‌سازی مستقیم روی تصاویر رنگی می‌تواند موجب تغییرات غیرواقعی در رنگ‌ها شود. برای تصاویر رنگی، روش‌های دیگری مثل **CLAHE (Contrast Limited Adaptive Histogram Equalization)** یا مساوی‌سازی جداگانه روی کانال روشنایی (L در فضای رنگی LAB یا Y در فضای YCrCb) به‌کار گرفته می‌شود. این باعث می‌شود کنتراست بهبود یابد بدون اینکه رنگ‌های تصویر غیرطبیعی شوند.

وقتی یک تصویر با استفاده از **هیستوگرام یکنواخت** یا سایر روش‌های بهبود کنتراست پردازش می‌شود، قله‌های هیستوگرام تغییر محسوسی پیدا می‌کنند. در تصویر اصلی با کنتراست پایین، هیستوگرام معمولاً در یک بازه‌ی محدود و باریک از شدت روشنایی متمرکز است، بنابراین یک یا چند قله‌ی بلند در همان محدوده دیده می‌شود. اما بعد از بهبود کنتراست، این پیکسل‌ها در بازه‌ی گسترده‌تری از شدت‌ها بازتوزیع می‌شوند.

در نتیجه، قله‌های بلند هیستوگرام شکسته و کوتاه‌تر می‌شوند و مقادیر شدت در کل بازه‌ی ۰ تا ۲۵۵ پخش می‌گردند. به بیان دیگر، هیستوگرام از حالت متراکم و متمرکز خارج شده و یکنواخت‌تر و گسترده‌تر می‌شود. این تغییر باعث می‌شود جزئیات بیشتری در نواحی تاریک و روشن تصویر دیده شود و تصویر نهایی طبیعی‌تر و با کنتراست بالاتر به نظر برسد.

```
image = cv2.imread('images/xray.jpg', 0)
result = cv2.equalizeHist(image)

plt.figure(figsize=[8,4])
plt.subplot(121);plt.imshow(image, cmap='gray');plt.title("Original X-ray image");
plt.subplot(122);plt.imshow(result, cmap='gray');plt.title("equalized histogram")
```

هر چند که استفاده از تابع `cv2.equalizeHist()` برای یکنواخت سازی هیستوگرام تصاویر رنگی مرسوم نیست اما میتوان ایت عملیات را انجام داد و نتیجه آن را مشاهد نمود:

```
import cv2
import matplotlib.pyplot as plt

image = cv2.imread('images/mid.png')
################
# Histogram Equalization
channels = cv2.split(image)
eq_channels = []
for ch in channels:
    eq_channels.append(cv2.equalizeHist(ch))
eq_image = cv2.merge(eq_channels)


#plot images
plt.figure(figsize=[12,7])
plt.subplot(221);plt.imshow(image[...,::-1]);plt.title("Original");
plt.subplot(223);plt.imshow(eq_image[...,::-1]);plt.title("Equalized");


color = ('b', 'g', 'r')
#Original image histogram
```

```
plt.subplot(222);
for i, col in enumerate(color):
    histogram = cv2.calcHist([image], [i], None, [256], [0, 256])
    plt.plot(histogram, color = col)

#Equualized image histogram
plt.subplot(224);
for i, col in enumerate(color):
    histogram = cv2.calcHist([eq_image], [i], None, [256], [0, 256])
    plt.plot(histogram, color = col)
```

این کد یک تصویر رنگی با تونالیته‌ی میانی (mid.png) را بارگذاری می‌کند و سپس روی هر سه کانال رنگی آن (آبی، سبز و قرمز) به طور جداگانه **هیستوگرام یکنواخت** (Histogram Equalization)اعمال می‌کند. ابتدا تصویر به سه کانال جداگانه با cv2.splitتقسیم می‌شود، سپس هر کانال با استفاده از cv2.equalizeHistپردازش می‌گردد تا کنتراست در آن افزایش یابد. بعد از انجام این عملیات روی هر سه کانال، آن‌ها دوباره با cv2.mergeادغام می‌شوند تا تصویر رنگی نهایی با کنتراست بهبودیافته ساخته شود.

در بخش رسم، ابتدا تصویر اصلی و تصویر Equalized کنار هم نمایش داده می‌شوند تا بتوان به‌طور بصری تغییرات کنتراست را مشاهده کرد. سپس با استفاده از cv2.calcHistو یک حلقه روی کانال‌های رنگی، هیستوگرام هر سه کانال در دو حالت اصلی و Equalized ترسیم می‌شود. در هیستوگرام تصویر اصلی، معمولاً توزیع شدت‌ها در محدوده‌ای محدود متمرکز است، در حالی که در هیستوگرام تصویرEqualized ، شدت‌ها گسترده‌تر و متعادل‌تر در بازه‌ی ۰ تا ۲۵۵ پخش شده‌اند. این مقایسه به خوبی نشان می‌دهد که چگونه عملیات equalizeHist قله‌های متمرکز هیستوگرام را بازتوزیع کرده و باعث بهبود کنتراست تصویر شده است.

یکنواخت سازی تطبیقی:

روش **CLAHE (Contrast Limited Adaptive Histogram Equalization)**نسخه‌ی پیشرفته‌تری از هیستوگرام مساوی‌سازی است که برای بهبود کنتراست تصویر به‌صورت محلی عمل می‌کند. در حالی که روش معمولی Equalization کل تصویر را یک‌جا پردازش کرده و ممکن است باعث اغراق یا نویز در برخی نواحی شود، CLAHEتصویر را به بلوک‌های کوچک (Tile)تقسیم می‌کند و در هر بلوک به‌طور جداگانه مساوی‌سازی هیستوگرام انجام می‌دهد. سپس این بلوک‌ها با هم ترکیب می‌شوند تا تصویر نهایی ایجاد شود. این روش کمک می‌کند تا جزئیات نواحی تاریک و روشن تصویر همزمان بهتر دیده شوند.

ویژگی مهم CLAHE این است که با استفاده از **Contrast Limiting**یا محدود کردن کنتراست، جلوی تقویت بیش‌ازحد نویز را می‌گیرد. در واقع اگر تعداد پیکسل‌ها در یک سطح شدت خیلی زیاد باشد (تشکیل یک قله‌ی بلند در هیستوگرام)، این قله بریده و مقادیر اضافی بین سطوح دیگر توزیع می‌شوند. این کار باعث می‌شود تصویر نهایی طبیعی‌تر به نظر برسد و نویز یا آرتیفکت‌ها

تقویت نشوند. به همین دلیل CLAHE به‌ویژه در تصاویر پزشکی (مثل عکس‌های رادیولوژی) و شرایطی که حفظ جزئیات اهمیت دارد، بسیار پرکاربرد است.

در OpenCV برای استفاده از روش **CLAHE** ابتدا باید یک شیء از آن بسازیم که با متد زیر انجام می‌شود:

```
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
```

این متد یک شیء **CLAHE** برمی‌گرداند که بعداً می‌توان روی تصویر خاکستری آن را اعمال کرد (مثلاً `clahe.apply(img)`). ورودی‌های مهم این متد عبارت‌اند از:

1. **`clipLimit`**: حد آستانه‌ی کنتراست. این مقدار میزان تقویت کنتراست را کنترل می‌کند. اگر مقدار خیلی کم باشد، اثر یکنواخت سازی ضعیف خواهد بود و اگر خیلی زیاد باشد، شبیه به equalizeHist معمولی عمل می‌کند و ممکن است نویز هم تقویت شود. مقدار پیش‌فرض معمولاً `2.0` است.
2. **`tileGridSize`**: اندازه‌ی تقسیم‌بندی تصویر به بلوک‌های کوچک (Tiles). این پارامتر یک زوج عددی است، مثلاً `(8,8)` یعنی تصویر به شبکه‌ای از بلوک‌های ۸×۸ تقسیم شود و هیستوگرام یکنواخت ساز در هر بلوک به صورت محلی اعمال گردد. اگر اندازه‌ی بلوک کوچک انتخاب شود، جزئیات بیشتری تقویت می‌شوند اما احتمال ایجاد آرتیفکت هم بیشتر است.

در نهایت با استفاده از `clahe.apply(image)`، تصویر ورودی (که باید **خاکستری** باشد) پردازش شده و نسخه‌ی بهبودیافته‌ی آن برگردانده می‌شود.

آرتیفکت (Artifact) در پردازش تصویر به **خطاها یا جلوه‌های ناخواسته‌ای** گفته می‌شود که در نتیجه‌ی پردازش، فشرده‌سازی یا انتقال تصویر ایجاد می‌شوند و کیفیت یا طبیعی بودن تصویر را کاهش می‌دهند. این پدیده معمولاً ناشی از محدودیت‌های الگوریتم یا داده است و در اصل جزئی از تصویر واقعی محسوب نمی‌شود.

برای مثال، در فشرده‌سازی JPEG ممکن است **بلوک‌بندی (Blockiness)** یا **لبه‌های مصنوعی** دیده شود، یا در ویدیوها به دلیل کاهش نرخ بیت، نویزهای رنگی و خطوط اضافی ظاهر شوند. در بهبود کنتراست مثل CLAHE هم اگر پارامترها درست انتخاب نشوند، ممکن است بخش‌هایی از تصویر بیش از حد روشن یا تاریک شوند و الگوهای مصنوعی به‌وجود بیایند. به طور کلی، آرتیفکت همان "اثرات جانبی ناخواسته" است که کیفیت بصری تصویر را پایین می‌آورد.

ما می توانیم از این متد بر روی عکس های رنگی نیز استفاده کنیم:

```
image = cv2.imread('images/statue.png')

# Histogram Equalization
channels = cv2.split(image)

#Adaptive method(CLAHE)
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
eq_channels = []
for ch in channels:
    eq_channels.append(clahe.apply(ch))
eq_clahe_image = cv2.merge(eq_channels)

#Global method
eq_channels = []
for ch in channels:
    eq_channels.append(cv2.equalizeHist(ch))
eq_image = cv2.merge(eq_channels)

#plot images
plt.figure(figsize=[15,15])
plt.subplot(321);plt.imshow(image[...,::-1]);plt.title("Original");
plt.subplot(323);plt.imshow(eq_clahe_image[...,::-1]);plt.title("After Adaptive histogram
equalization");
plt.subplot(325);plt.imshow(eq_image[...,::-1]);plt.title("After global histogram
equalization");


color = ('b', 'g', 'r')
#Original image histogram
plt.subplot(322);
for i, col in enumerate(color):
    histogram = cv2.calcHist([image], [i], None, [256], [0, 256])
    plt.plot(histogram, color = col)

#Equualized image histogram with Adaptive method
plt.subplot(324);
for i, col in enumerate(color):
    histogram = cv2.calcHist([eq_clahe_image], [i], None, [256], [0, 256])
    plt.plot(histogram, color = col)

#Equualized image histogram with Global method
plt.subplot(326);
```

```
for i, col in enumerate(color):
    histogram = cv2.calcHist([eq_image], [i], None, [256], [0, 256])
    plt.plot(histogram, color = col)
```

این کد یک تصویر رنگی از فایل statue.png بارگذاری کرده و با دو روش مختلف کنتراست آن را بهبود می‌دهد: روش **یکنواخت‌سازی هیستوگرام تطبیقی (CLAHE)** و روش **یکنواخت سازی هیستوگرام سراسری Global Histogram Equalization**. ابتدا تصویر به سه کانال رنگی (آبی، سبز و قرمز) با cv2.split تقسیم می‌شود تا هر کانال به‌طور جداگانه پردازش گردد. در روش تطبیقی، یک شیء CLAHE با پارامترهای clipLimit=2.0 و tileGridSize=(8,8) ساخته می‌شود. سپس روی هر کانال اعمال می‌گردد و نتایج در قالب یک تصویر رنگی جدید ادغام می‌شوند. این تصویر همان نسخه‌ی CLAHE تصویر اصلی است که کنتراست آن به صورت محلی و کنترل‌شده بهبود یافته است.

در ادامه برای مقایسه، روش سراسری نیز روی همان سه کانال انجام می‌شود. در این روش از cv2.equalizeHist استفاده شده که کل تصویر را یک‌جا پردازش می‌کند و پیکسل‌ها را در بازه‌ی شدت ۰ تا ۲۵۵ بازتوزیع می‌کند. نتیجه‌ی این روش نیز پس از ادغام کانال‌ها یک تصویر رنگی جدید است. بنابراین در پایان سه نسخه از تصویر در اختیار داریم: نسخه‌ی اصلی، نسخه‌ی بهبود یافته با CLAHE و نسخه‌ی بهبود یافته با روش سراسری.

برای نمایش نتایج، از شش پنجره‌ی subplot استفاده شده است. در ستون سمت چپ سه تصویر اصلی، CLAHEو Global کنار هم نشان داده می‌شوند تا تفاوت‌های بصری در کنتراست مشخص شود. در ستون سمت راست نیز برای هر نسخه از تصویر، هیستوگرام رنگی (آبی، سبز، قرمز) رسم می‌شود. این هیستوگرام‌ها نشان می‌دهند که در تصویر اصلی شدت رنگ‌ها در یک محدوده‌ی خاص متمرکز است، در تصویر Global شدت‌ها به‌طور یکنواخت‌تری در کل بازه پخش شده‌اند و در تصویر CLAHE نیز بازتوزیع شدت‌ها به صورت محلی و کنترل‌شده اتفاق افتاده است. این مقایسه هم به لحاظ بصری و هم آماری نشان می‌دهد که روش‌های مختلف مساوی‌سازی هیستوگرام چه تأثیری بر کنتراست تصویر دارند.

```
for i, col in enumerate(color):
    histogram = cv2.calcHist([image], [i], None, [256], [0, 256])
    plt.plot(histogram, color = col)
```

این بخش از کد برای بهبود کنتراست تصویر با روش **CLAHE**به‌کار می‌رود. ابتدا با دستور cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))یک شیء CLAHE ساخته می‌شود که پارامتر clipLimitمیزان محدودسازی کنتراست (برای جلوگیری از تقویت بیش‌ازحد نویز) و tileGridSizeاندازه‌ی بلوک‌های تقسیم‌بندی تصویر (۸×۸) را تعیین می‌کند. سپس با یک حلقه، هر یک از کانال‌های رنگی تصویر جداگانه پردازش می‌شود؛ به این صورت که متد applyروی هر کانال اعمال شده و نسخه‌ی بهبودیافته‌ی آن در

لیست eq_channels ذخیره می‌گردد. در نهایت، کانال‌های پردازش‌شده با دستور cv2.merge دوباره به هم ادغام می‌شوند تا تصویر رنگی جدیدی به دست آید که کنتراست آن به صورت محلی و تطبیقی افزایش یافته است.