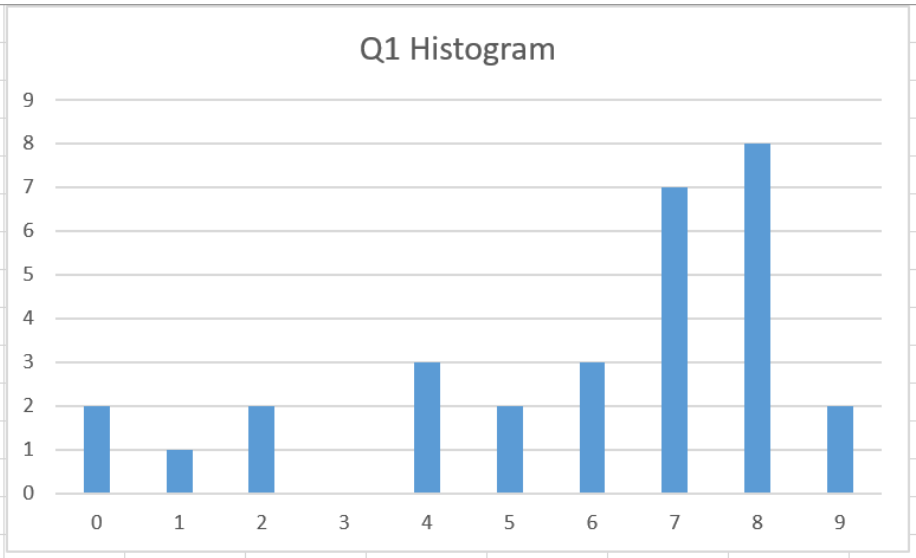
۱.



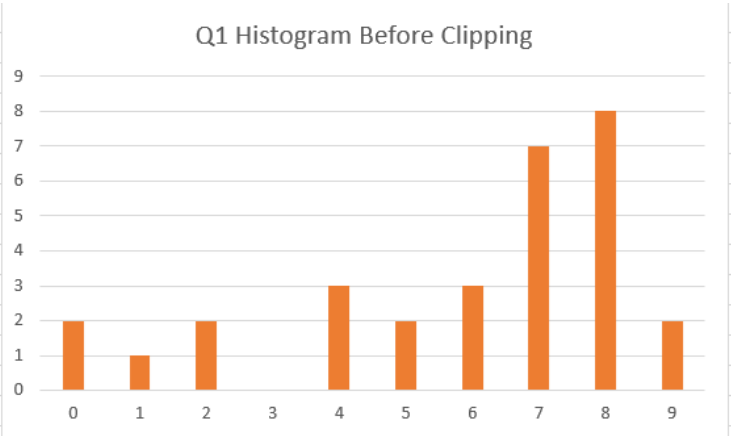
2	1	2	0	3	2	3	7	8	2
0	1	2	3	4	5	6	7	8	9

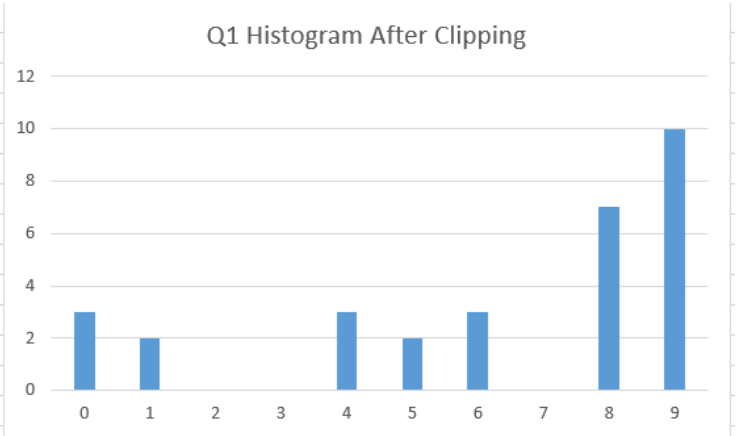
هیستوگرام تصویر اصلی

رابطه برش هیستوگرام با ۱۰٪ مولفه‌ها: $\left(\frac{f(x,y)-f_{10}}{f_{90}-f_{10}}\right)(MAX-MIN)+MIN$

$$10\% \times 30 = 3 \Longrightarrow \begin{cases} f_{10} = 1 \\ f_{90} = 8 \end{cases}$$

$$\left(\frac{f(x,y)-1}{8-1}\right)(9-0)+0 = \frac{9}{7}(f(x,y)-1)$$





تعداد هر رنگ در تصویر اصلی	2	1	2	0	3	2	3	7	8	2
مقدار رنگ	0	1	2	3	4	5	6	7	8	9
بعد از برش	-1.2857	0	1.28571	2.57143	3.85714	5.14286	6.42857	7.71429	9	10.2857
بعد از برش و محدودسازی مقادیر	0	0	1.28571	2.57143	3.85714	5.14286	6.42857	7.71429	9	9
بعد از برش و محدودسازی و رندکردن	0	0	1	3	4	5	6	8	9	9
تعداد هر رنگ بعد از برش	3	2	0	0	3	2	3	0	7	10
مقدار رنگ	0	1	2	3	4	5	6	7	8	9

تصویر برش‌یافته:

8	8	9	9	9	9
1	0	4	4	4	9
8	0	5	5	1	9
9	0	6	9	9	8
9	8	6	6	8	8

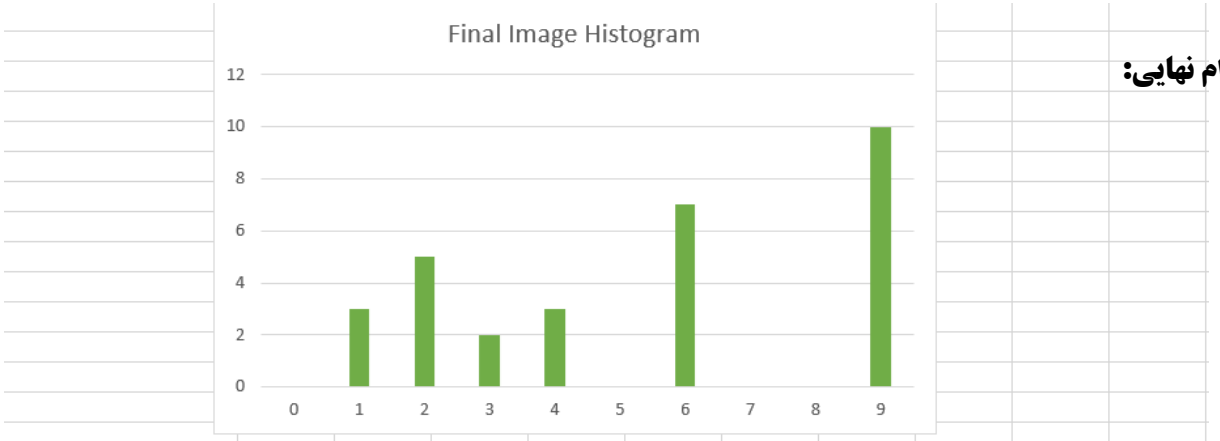
عمل متعادل‌سازی روی هیستوگرام تصویر برش‌یافته:

k	0	1	2	3	4	5	6	7	8	9
n_k	**3**	**2**	**0**	**0**	**3**	**2**	**3**	**0**	**7**	**10**
$\sum_{j=0}^{k} n_j$	**3**	**5**	**5**	**5**	**8**	**10**	**13**	**13**	**20**	**30**
$\sum_{j=0}^{k} \frac{n_j}{n}$	$\frac{3}{30}$	$\frac{5}{30}$	$\frac{5}{30}$	$\frac{5}{30}$	$\frac{8}{30}$	$\frac{10}{30}$	$\frac{13}{30}$	$\frac{13}{30}$	$\frac{20}{30}$	$\frac{30}{30}$
$(L-1)\sum_{j=0}^{k} \frac{n_j}{n}$	**0.9**	**1.5**	**1.5**	**1.5**	**2.4**	**3**	**3.9**	**3.9**	**6**	**9**
Round	**1**	**2**	**2**	**2**	**2**	**3**	**4**	**4**	**6**	**9**

تصویر نهایی:

6	6	9	9	9	9
2	1	2	2	2	9
6	1	3	3	2	9
9	1	4	9	9	6
9	6	4	4	6	6

هیستوگرام نهایی:



تعداد رنگ در تصویر نهایی	0	3	5	2	3	0	7	0	0	10
مقدار رنگ	0	1	2	3	4	5	6	7	8	9

مقدار رنگ در تصویر نهایی	1	2	2	2	2	3	4	4	6	9
تعداد رنگ در تصویر برش‌یافته	3	2	0	0	3	2	3	0	7	10
مقدار رنگ در تصویر برش‌یافته	0	1	2	3	4	5	6	7	8	9

۲.الف.

ابتدا تعداد موجود از هر رنگ در تصویر را شمردم و در *cnt[color]* ذخیره کردم سپس توزیع تجمعی رنگ‌ها را در *cdf* محاسبه و ذخیره کردم.

مقادیر بدست‌آمده در *cdf* را ضربدر *L-1* و تقسیم بر *n* کردم و بعد از رندکردن به نزدیک‌ترین عدد صحیح در *equalized_vals* ریختم. حالا باید همه‌ی رنگ‌های *k* در تصویر به *equalized_vals[k]* تبدیل بشوند که در نهایت با انجام این کار *output_image* را ساختم.

```
############
# Your code
# Start

L = 256 # Number of colors
height = image.shape[0]
width = image.shape[1]
n = height * width # Number of pixels

cnt = np.zeros((L,), dtype=int)

for line in image:
    for pixel in line:
        cnt[pixel] += 1

cdf = np.zeros((L,), dtype=int)
cdf[0] = cnt[0]
for color in range(1, L):
    cdf[color] = cdf[color - 1] + cnt[color]

equalized_vals = np.rint(cdf * (L - 1) / n).astype(np.int32)

output_image = np.zeros((height, width), dtype=np.uint8)

for i in range(height):
    for j in range(width):
        output_image[i][j] = equalized_vals[image[i][j]]
# End
```

```
# START
equ = hist_equ(img)
# END

res = np.hstack((img, equ)) #stacking images side-by-side

plt.figure(figsize=(16, 16))
plt.imshow(res, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x154c9ce9e48>
```

خروجی حاصل از تابع متعادل‌سازی نوشته شده (تصویر سمت راست) در کنار تصویر اصلی (تصویر سمت چپ)

```
# START
equ = cv2.equalizeHist(img)
# END

res = np.hstack((img, equ)) #stacking images side-by-side

plt.figure(figsize=(16, 16))
plt.imshow(res, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x154c9d672c8>
```

خروجی حاصل از تابع متعادل‌سازی *opencv* (تصویر سمت راست) در کنار تصویر اصلی (تصویر سمت چپ)

از مقایسه خروجی صفحه قبل و خروجی بالا می‌توان دید تفاوت چندانی بین خروجی تابع نوشته شده توسط من و خروجی تابع *equalizeHist* خود *OpenCV* وجود ندارد که احتمالا بخاطر ساز و کار مشابه متعادل‌سازی در هر دو است.

۲.ب.

```
# START
clh = CLAHE(img, (8, 8), 5)
# END

res = np.hstack((img, clh)) #stacking images side-by-side

plt.figure(figsize=(16, 16))
plt.imshow(res, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x1c2442bdfc8>
```

نتیجه بدست‌آمده دارای جزئیات بسیار بیشتری نسبت به خروجی قسمت قبل (متعادل‌سازی) است که بدلیل بهبود کنتراست هر بخش از تصویر متناسب با روشنایی همان قسمت از تصویر در توابع متعادل‌سازی سازگار مثل *CLAHE* است. مثلا در مورد شاخه‌های درخت، بازتابشان در رودخانه، بوته‌ها و سنگچین پل این بهبود جزئیات کاملا مشهود است.

۲.پ.

خروجی تابع متعادل‌سازی خودم **تصویر اصلی**

خروجی تابع متعادل‌سازی *OpenCV* **تصویر اصلی**

مشابه نتیجه بدست‌آمده روی تصویر *River* در اینجا هم تفاوت چندانی بین تابع متعادل‌سازی نوشته شده و تابع خود *OpenCV* دیده نمی‌شود و هر دو، کنتراست تصویر را بهبود داده‌اند.

خروجی تابع *CLAHE* **تصویر اصلی**

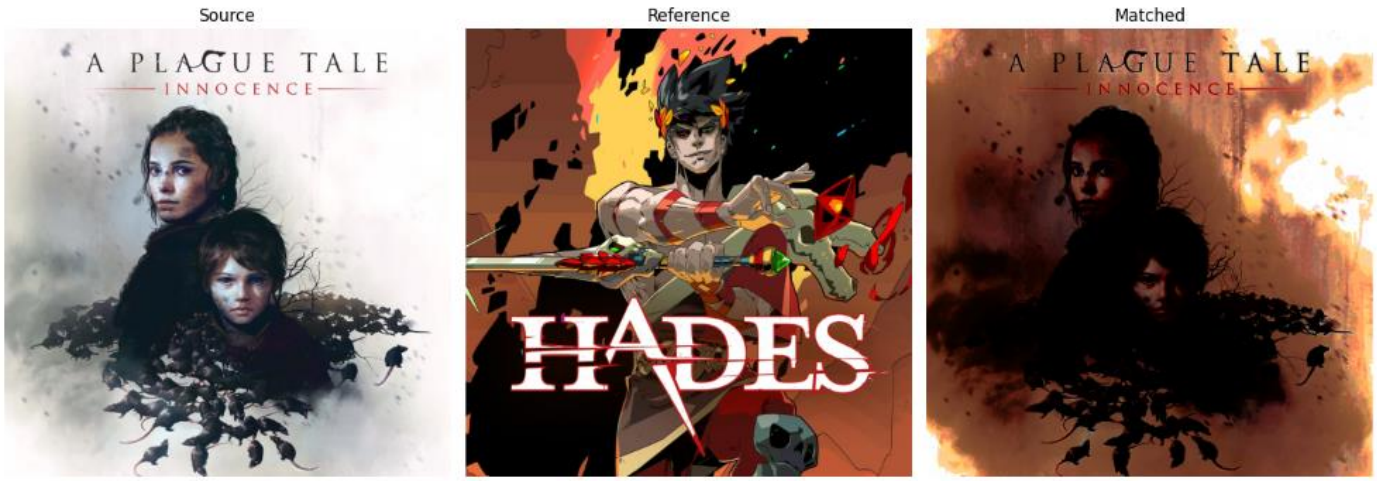
کنتراست تصویر نه‌تنها نسبت به تصویر اصلی بهبود یافته است بلکه نسبت به خروجی متعادل‌سازی صفحه قبل هم بهتر شده است که از مقایسه ساختمان‌های کنار و حاشیه قلعه اصلی کاملا مشخص است.

۲.ت. خیر، اگر متعادل‌سازی را روی هر کانال جداگانه انجام دهیم تناسب بین نسبت رنگ سبز و قرمز و آبی هر پیکسل را به هم می‌زنیم و بجای بهبود روشنایی ممکن است ماهیت خود رنگ‌های موجود در تصویر را به هم بزنیم و در برخی نقاط رنگ‌های غیرواقعی ایجاد کنیم. یک راه‌حل جایگزین می‌تواند تبدیل تصویر از فضای رنگی *RGB* به فضای رنگی دیگری باشد که شدت روشنایی را از مولفه‌های رنگی جدا کند تا بتوانیم فقط روی شدت روشنایی متعادل‌سازی انجام دهیم و سپس تصویر را دوباره به فضای *RGB* برگردانیم و از آن استفاده کنیم.

نمونه‌هایی از فضای رنگی که شدت روشنایی را از خود رنگ جدا نگه می‌دارند:

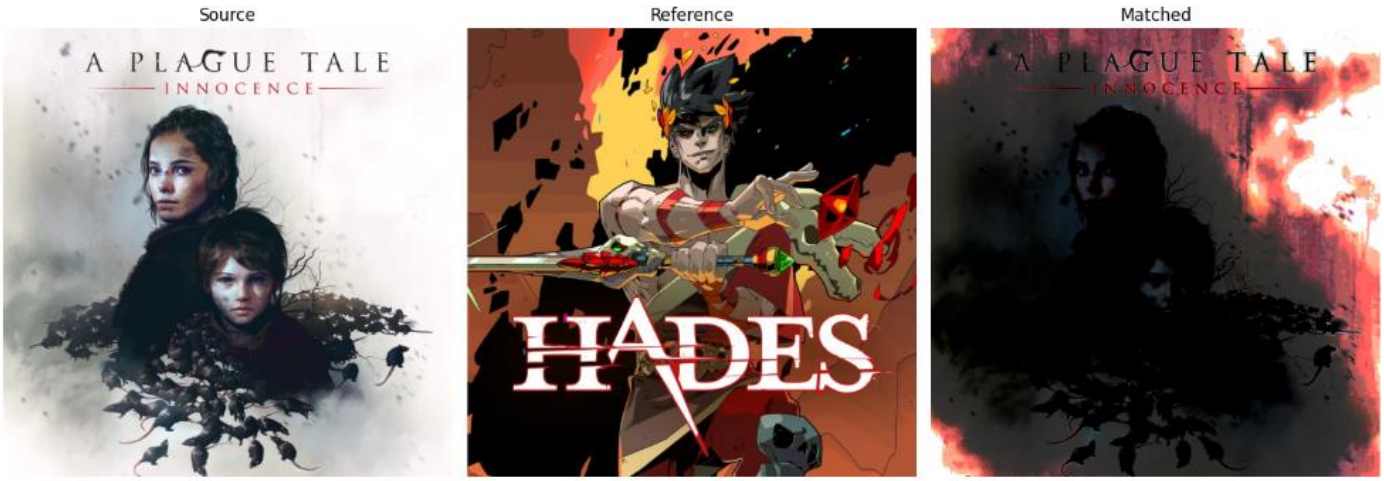
- HSV/HLS
- YUV
- YCbCr

۳.الف.



چون در تصویر مرجع، فراوانی رنگ‌های مایل به قرمز بسیار زیاد است در تصویر نهایی هم، تطبیق هیستوگرام باعث شده بسیاری از نقاط تصویر به طیف قرمز تبدیل شوند. با این وجود قسمت‌های مختلف تصویر همچنان تقریبا قابل تشخیص از همدیگرند.

۳.ب.



نتیجه بازهم قرمزگون شده اما قدری با تصویر بدست‌آمده قسمت قبل تفاوت دارد و این تفاوت بخاطر جزئیاتی که در پیاده‌سازی تطبیق هیستوگرام وجود دارد مخصوصا برای تصاویر رنگی قابل انتظار است. در تابعی که خودم پیاده‌سازی کردم سه کانال را جداگانه تطبیق هیستوگرام دادم ممکن است در تابع آماده اینگونه پیاده‌سازی نشده باشد.

تفاوت دیگری که وجود دارد سرعت بسیار بالاتر تابع آماده نسبت به تابعی است که خودم نوشتم. که احتمالا بدلیل پیاده‌سازی بهینه‌تر تطبیق هیستوگرام است.

۳.پ.

خروجی تابع آماده تطبیق هیستوگرام

خروجی تابع تطبیق هیستوگرام خودم

منابع:

سوال ۲.الف: numpy.zeros — NumPy v1.23 Manual

1.4.2. Numerical operations on arrays — Scipy lecture notes (scipy-lectures.org)

Round elements of the array to the nearest integer in Numpy (tutorialspoint.com)

python - Can numpy.rint to return an Int32? - Stack Overflow

numpy.ndarray.dtype — NumPy v1.23 Manual

OpenCV: Histograms - 2: Histogram Equalization

سوال ۲.ب: OpenCV: Histograms - 2: Histogram Equalization

سوال ۲.ت: Histogram Equalization Of RGB Images – Perpetual Enigma (prateekvjoshi.com)

سوال ۳.الف: Histogram matching — skimage v0.19.2 docs (scikit-image.org)