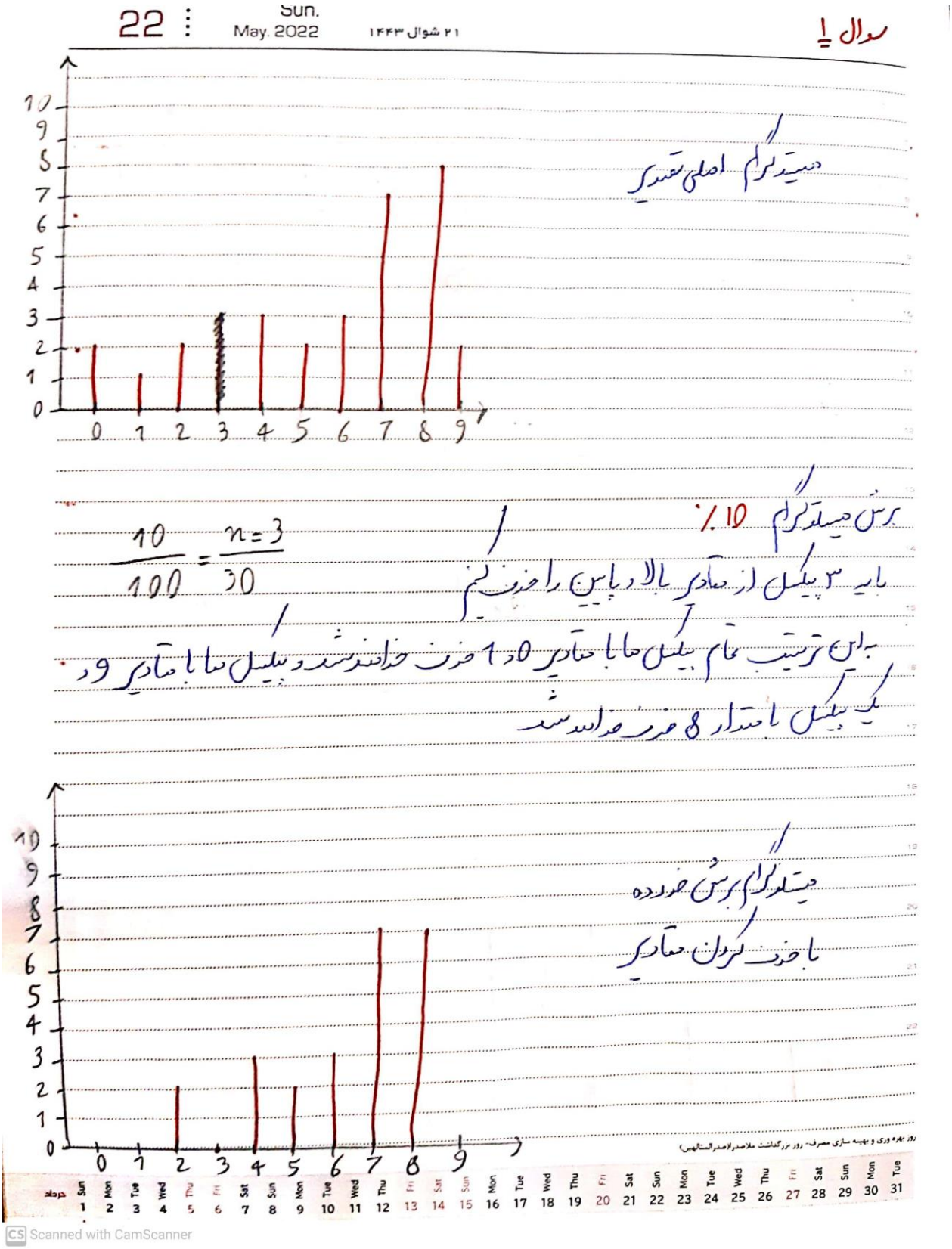
گزارش تمرین سوم بینایی ماشین

نام تهیه‌کننده: ملیکا نوبختیان

شماره دانشجویی: ۹۷۵۲۲۰۹۴

نسخه: ۱

۱- سوال اول

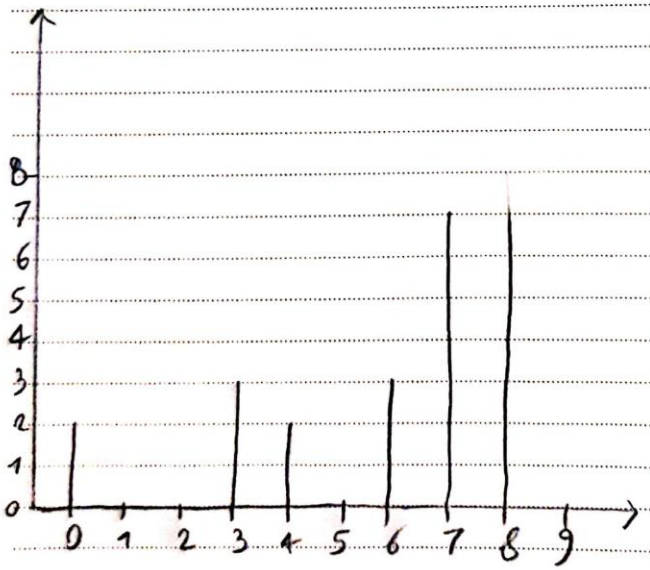


$$g(x,y) = clip[f(x,y)] = \left(\frac{f(x,y) - f_{10}}{f_{90} - f_{10}}\right)(Max - MIN) + MIN$$

$Max = 9 \quad Min = 0$

$f_{10} = 2 \quad f_{90} = 8 \quad g(x,y) = \frac{9}{6} \times (f(x,y) - 2)$

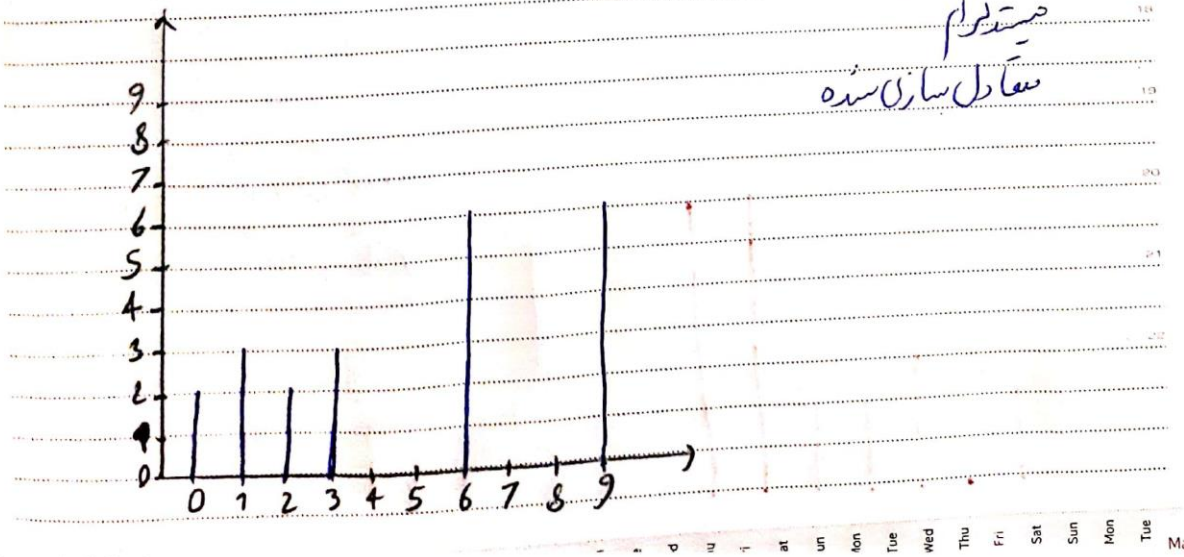
$f(x,y)$	0	1	2	3	4	5	6	7	8	9
$g(x,y)$	-	-	0	1.5	3	4.5	6	7.5	9	-
round	-	-	0	1	3	4	6	7	9	-



۲۲ شوال ۱۴۴۳ May. 2022 : 23

k	0	1	2	3	4	5	6	7	8	9
n_k	0	0	2	0	3	2	3	7	7	0
$\sum_{j=0}^{k} n_j$	0	0	2	2	5	7	10	17	24	24
$\sum_{j=0}^{k} \frac{n_j}{n}$	0	0	$\frac{1}{12}$	$\frac{1}{12}$	$\frac{5}{24}$	$\frac{7}{24}$	$\frac{5}{12}$	$\frac{17}{24}$	1	1
$(L-1)\sum_{j=0}^{k} \frac{n_j}{n}$	0	0	$\frac{3}{4}$	$\frac{3}{4}$	1.875	2.1625	3.75	6.1375	9	9
Round	0	0	0	0	1	2	3	6	9	9

هیستوگرام
معادل سازی شده



۲- سوال دوم

برای پیاده‌سازی متعادل‌سازی هیستوگرام ابتدا لازم است بدانیم که هر کدام از پیکسل‌های تصویر چه مقداری دارند (۰ تا ۲۵۶) و چه تعداد از آن‌ها موجود است. هم چنین مقدار تجمعی آن‌ها را نیز می‌خواهیم. در قطعه کد زیر nk_dict نشان می‌دهد هر کدام از ۰ تا ۲۵۶ مقدار چه تعداد پیکسل را به خود اختصاص می‌دهند و nk_sum_dict مقدار تجمعی آن‌ها را نشان می‌دهد:

```
max_value = 256
    for i in range(max_value):
        nk_dict[i] = np.count_nonzero(image == i)
        nk_sum_dict[i] = nk_sum_dict[i-1] + nk_dict[i]
```

سپس لازم است احتمال هر رنگ را به دست آوریم که برای این کار باید مقادیر به دست آمده در nk_sum_dict را به تعداد کل پیکسل‌های تصویر تقسیم کنیم. بعد از آن باید در مقدار $L-1$ که همان ۲۵۵ است ضرب کنیم و در آخر مقادیر به دست آمده را گرد کنیم:

```
for i in range(1, max_value):
        nk_sum_dict[i] /= image.size
        nk_sum_dict[i] *=  (max_value - 1)
        nk_sum_dict[i] = round(nk_sum_dict[i])
```

حالا لازم است مقادیر جدید پیکسل‌ها را با توجه به مقدارهایی که به دست آوردیم تغییر دهیم. Output_image تصویر نهایی ما را در خود خواهد داشت:

```
output_image = np.zeros_like(image)

    for i in range(max_value):
        output_image = np.where(image != i, output_image, nk_sum_dict[i])
```

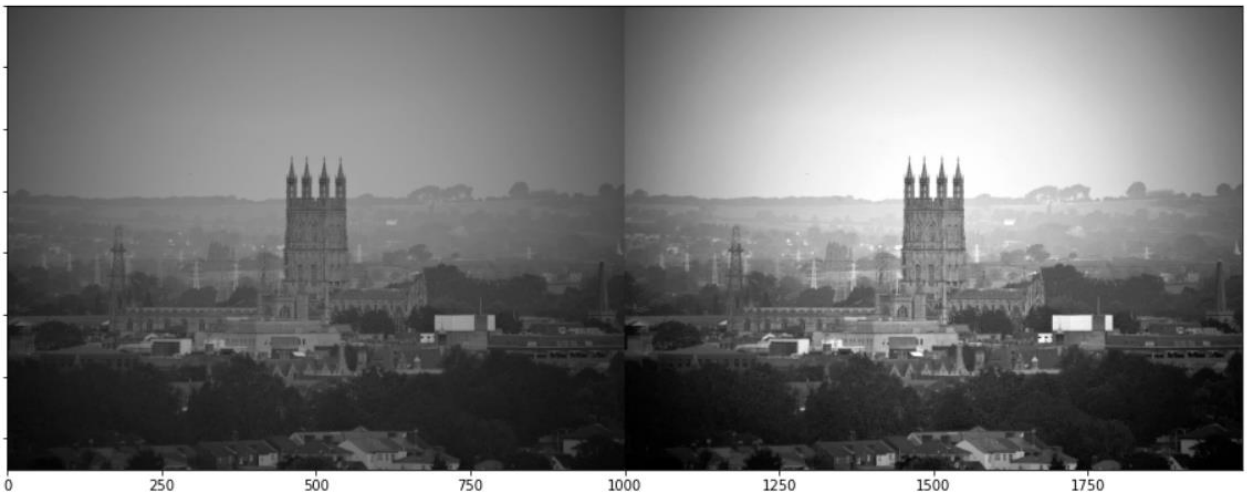
نتیجه اعمال تابع بر روی تصویر به صورت زیر خواهد بود:

حالا با استفاده از تابع پیاده‌سازی شده opencv این کار را انجام می‌دهیم:

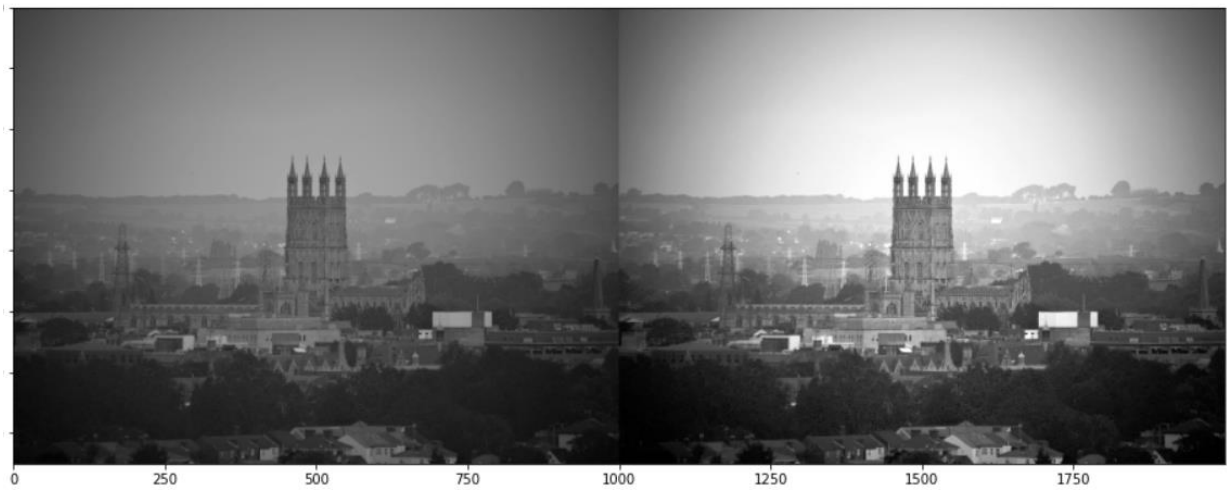
در هر دو حالت کیفیت تصویر افزایش پیدا کرده و طیف رنگی بیشتری را شامل شده‌است. نتیجه اعمال با استفاده از تابع آماده opencv و تابع پیاده‌سازی شده با numpy تفاوت خاصی ندارد و نتیجه هر دو به نظر یکسان است.
حالا CLAHE را روی تصویر اعمال می‌کنیم. برای اعمال آن از clip limit به اندازه ۲ و grid_size به شکل ۸*۸ استفاده می‌کنیم نتیجه به شکل زیر خواهد بود:

همان طور که واضح است تصویری که در این قسمت مشاهده می‌کنیم کیفیت بهتری نسبت به تصویر حالت قبل دارد. برای مثال می‌توان به جزئیات شاخه‌ها در تصاویر اشاره کرد. در تصویری که با استفاده از CLAHE بهبود یافته‌است، به طور واضح تری می‌توانیم شاخه‌ها را مشاهده کنیم و مانند تصویری که با استفاده متعادل‌سازی هیستوگرام بهبود دادیم شاخه‌ها حالت تار و محود ندارند. CLAHE از روش‌های ارتقا محلی است و هم چنین میزان تقویت کنتراست را محدود می‌کند و باعث می‌شود کیفیت تصویر افزایش یابد.

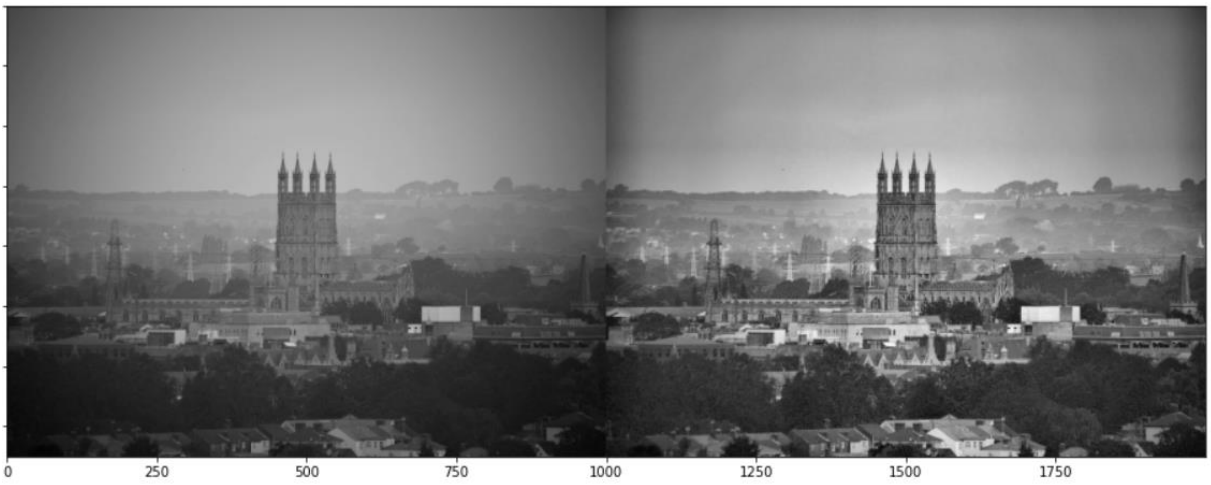
حالا به سراغ تصویر City.png می‌رویم و مراحل بالا را برای آن هم اعمال می‌کنیم.
نتیجه اعمال تابع پیاده‌سازی شده متعادل سازی روی تصویر به شکل زیر است:



نتیجه اعمال تابع متعادل سازی هیستوگرام در opencv به شکل زیر است:



و نتیجه اعمال CLAHE نیز به شکل زیر خواهد بود:



اگر بخواهیم متعادل‌سازی هیستوگرام را روی تصاویر رنگی اعمال کنیم، شاید اولین چیزی که به ذهن می‌رسد اعمال متعادل‌سازی به طور جداگانه روی سه کانال تصویر است اما این روش مشکلاتی دارد. وقتی ما متعادل‌سازی را به طور جداگانه روی هر کانال اعمال می‌کنیم، در واقع داریم ارتباط این سه کانال را نادیده می‌گیریم. در صورتی که رنگ‌های موجود در تصویر با ارتباط این سه کانال ایجاد شده‌اند و اعمال جداگانه متعادل‌سازی روی این سه کانال ممکن است color balance تصویر را به هم بریزد و رنگ‌های متفاوتی به ما بدهد. به عبارت دیگر لازم است تصویر ما در فضایی باشد که مقادیر intensity را از مقادیر color components جدا کند تا این متعادل‌سازی مولفه‌های رنگی تصویر را به هم نریزد.

یک روش برای حل مشکل این است که ابتدا تصویر رنگی خود را به فضاهای HSV/HSI ببریم و سپس تابع متعادل‌سازی را روی مقادیر luminance اعمال کنیم. در این صورت مقادیر hue و saturation تصویر بدون تغییر باقی خواهند ماند.

منابع:

https://hypjudy.github.io/2017/03/19/dip-histogram-equalization/
https://prateekvjoshi.com/2013/11/22/histogram-equalization-of-rgb-images/

۳- سوال سوم

برای تطبیق هیستوگرام Hades بر Plague Tale به شکل زیر عمل می‌کنیم:

```
reference = plt.imread('A Plague Tale.jpg')
source = plt.imread('Hades.jpg')

### YOUR CODE ###
# START
matched = match_histograms(source, reference, multichannel=True)
# END
```

نتیجه این تطبیق به صورت زیر خواهد بود:

همان طور در تصویر نهایی واضح است طیف رنگ‌های موجود در تصویر Hades روشن‌تر شده‌اند و از رنگ‌های تیره‌تر مثل قرمز، قهوه‌ای، سیاه و .... به سمت رنگ‌های روشن‌تر مانند سفید، آبی و ... رفته‌اند. این به این دلیل است که در تصاویر plague tale بر خلاف تصویر Hades تعداد زیادی از پیکسل‌ها سفید و روشن و بقیه سیاه هستند و چون تطبیق را به آن انجام داده‌ایم طیف رنگی تصویر Hades هم به این شکل تغییر کرده‌است.

حالا به سراغ پیاده‌سازی تطبیق هیستوگرام با استفاده از python و numpy می‌رویم. ابتدا لازم است هیستوگرام تصاویر source و reference را برای هر کانال آن‌ها به دست آوریم. برای این کار در اینجا از np.histogram استفاده می‌کنیم:

```
bins_count = 255
  channels_count = src_image.shape[-1]
  for channel in range(channels_count):
    src_hist, bins = np.histogram(src_image[..., channel].flatten(), bins_
count)
    ref_hist, bins = np.histogram(ref_image[..., channel].flatten(), bins_
count)
```

در قدم بعد لازم است مقادیر تجمعی هیستوگرام‌ها را محاسبه کنیم. برای این کار از np.cumsum استفاده می‌کنیم. در نهایت هم برای نرمال‌سازی مقادیر بر تعداد کلی پیکسل‌ها تقسیم می‌کنیم:

```
# cumulative sum
  cdfsrc = src_hist.cumsum()
  cdfsrc = 255 * cdfsrc // cdfsrc[-1]

  cdftint = ref_hist.cumsum()
  cdftint = 255 * cdftint // cdftint[-1]
```

حالا لازم است با توجه به مقادیر جدیدی که به دست آوردیم هیستوگرام تصویر source را تغییر دهیم که نتیجه آن Src_equ خواهد بود. حالا که این هیستوگرام را برای تصویر source به دست آوردیم باید آن را به هیستوگرام تصویر reference تطبیق دهیم. مقادیر نرمال‌شده‌ای که برای تصویر ref به دست آوردیم را مرجعی برای این تطبیق در نظر می‌گیریم. در نهایت هم مقادیر جدید را برای کانال موردنظر جایگذاری می‌کنیم:

```
src_equ = np.interp(src_image[:,:,channel].flatten(),bins[:-1],cdfsrc)
    matched_channel = np.interp(src_equ,cdfref, bins[:-1])

    matched[...,channel] = matched_channel.reshape((src_image.shape[0],src
_image.shape[1]))
```

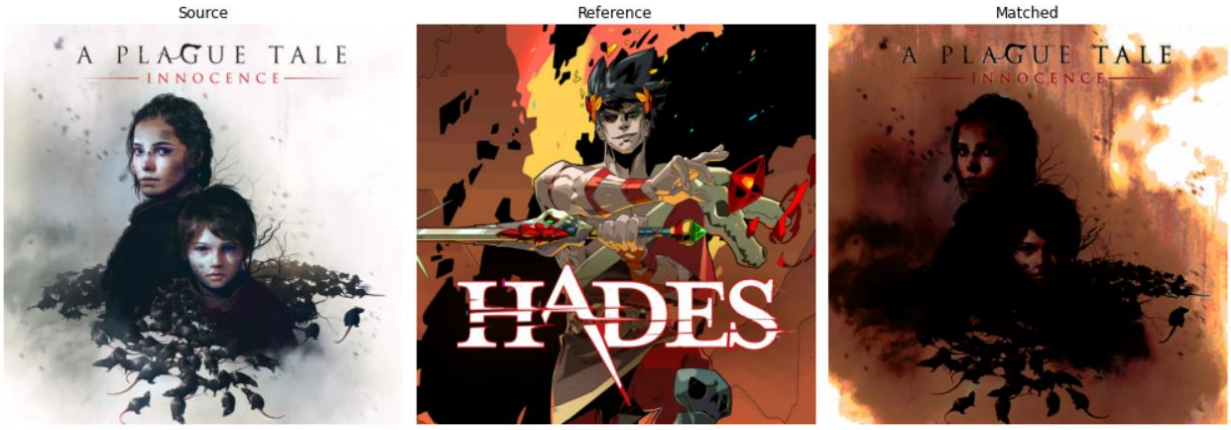
در نهایت پس از انجام این کار به تصویر تطبیق شده خواهیم رسید. نتیجه برای این حالت هم به شکل زیر خواهد بود:

نتیجه این حالت هم مانند حالت قبل است که با استفاده از match_histogram این کار را انجام دادیم و نکات قسمت قبل برای آن برقرار است.

حالا جای تصاویر source و reference را تغییر می‌دهیم. نتیجه اعمال تابع آماده به شکل زیر خواهد بود:

در اینجا حالتی عکس قسمت قبل داریم. در اینجا تصویر مرجع ترکیب رنگ‌های تیره زیادی داره که در حدود قرمز، قهوه‌ای و ... هستند. تطبیق هیستوگرام باعث می‌شود این ویژگی‌های رنگی به تصویر source نیز انتقال یابد و همان طور که در تصویر نهایی نیز مشخص است رنگ‌های قهوه‌ای زیادی به تصویر اضافه‌شده‌اند و رنگ‌های سفید و روشن را تغییر داده‌اند.

نتیجه اعمال تابع پیاده‌سازی شده با python و numpy هم به شکل زیر خواهد بود:



که مانند حالت قبل است و توصیفات حالت قبل برای آن برقرار است.

منبع:

https://vzaguskin.github.io/histmatching1/