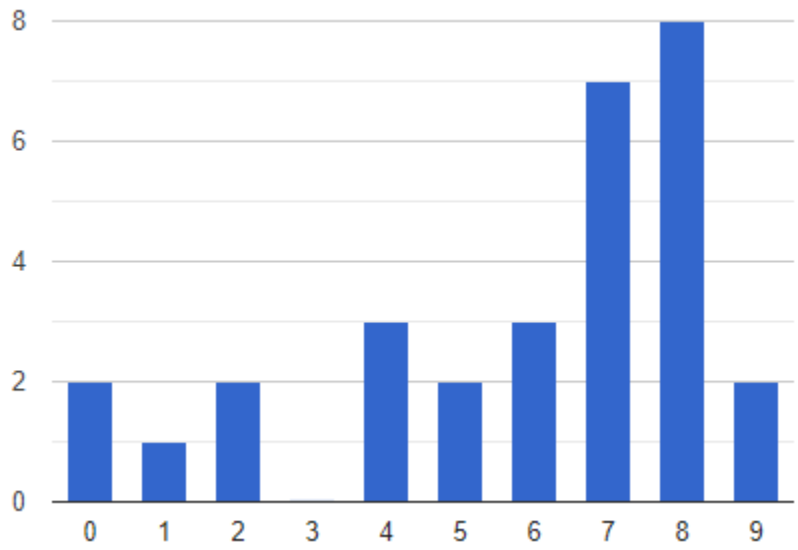
تمرین 3 بینایی کامپیوتر

سینا اسکندری 97521054

-1

هیستوگرام اولیه تصویر به صورت زیر می باشد.



اگر مقادریر پیکسل ها را sort کنیم مقادیر زیر بدست می آید

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

و 10 درصد پایین و بالا(چون 30 پیکسل داریم 3 پیکسل بالا و پایین) حذف کنیم حاصل می شود:

2, 2, 4, 4, 4, 5, 5, 6, 6, 6, 7, 7, 7, 7, 7, 7, 7, 8, 8, 8, 8, 8, 8, 8

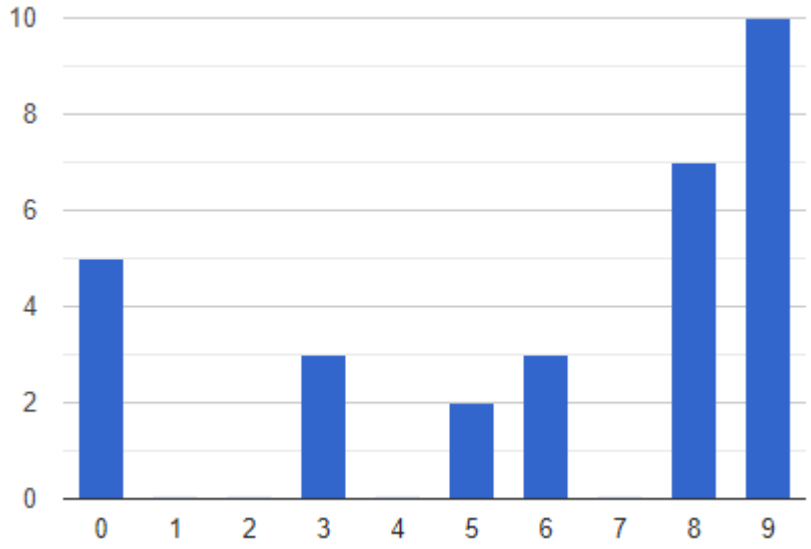
که در این حالت f_{min} برابر 2 و f_{max} برابر 8 است. اگر فرمول برش هیستوگرام را اعمال کنیم تصویر به صورت زیر می شود.

7.5	7.5	9	9	9	9
0	-1.5	3	3	3	9
7.5	-3	4.5	4.5	0	9
9	-3	6	10.5	10.5	7.5
9	7.5	6	6	7.5	7.5

چون مقادیر منفی و بزرگتر از 9 ممکن نیست این مقادیر را به ترتیب به 0 و 9 map می کنیم و بقیه مقادیر نیز گرد می شوند.

8	8	9	9	9	9
0	0	3	3	3	9
8	0	5	5	0	9
9	0	6	9	9	8
9	8	6	6	8	8

هیستوگرام تصویر الان به این شکل می شود.



2-

الف)

برای پیاده‌سازی متعادل‌سازی هیستوگرام مراحل اسلاید جلسه سوم صفحه 30 را طی می‌کنیم. ابتدا باید هیستوگرام تصویر را با دستور np.histogram بدست آوریم. سپس باید مقدار تجمعی هیستوگرام محاسبه شود که برای این کار از دستور np.cumsum استفاده می‌کنیم و سپس متعادل‌سازی می‌کنیم و در نهایت در L-1 که 255 است ضرب می‌نماییم تا cdf بدست آید. در آخر نیز یک آرایه خالی با ابعاد تصویر ایجاد می‌کنیم و هر نقطه از تصویر اصلی را به نقطه متناظر در cdf مپ می‌کنیم. مراحل به صورت زیر است:

```python
def hist_equ(image):
    '''
    input:
    image (ndarray): input image
    output:
    output_image (ndarray): enhanced image
    '''

    ############
    # Your code
    # Start
    histogram = np.histogram(image.flatten(), bins=256, range=(0, 255))[0]
    cumulative_sum = np.cumsum(histogram)
    cdf = np.floor(255 * (cumulative_sum / cumulative_sum[-1]))
    output_image = np.zeros(image.shape)
    for i in range(256):
        output_image[image == i] = cdf[i]
    # End
    return output_image
```

نتیجه حاصل شده:

همانطور که قابل مشاهده است متعادل سازی به خوبی انجام شده است و رنج زیادی از روشنایی‌ها قابل مشاهده است.

تابع آماده موجود در opencv، equalizeHist است که عکس را ورودی می‌گیرد.

```
img = cv2.imread('River.jpg', 0)

### YOUR CODE ###
# 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')
```

نتیجه:

همان‌طور که مشخص است نتیجه خیلی شبیه به حالتی است که من پیاده‌سازی کردم و در اینجا نیز به خوبی متعادل‌سازی شده است.

ب)

برای استفاده از تابع CLAHE ابتدا باید با استفاده از createCLAHE یک اینستنس از CLAHE درست کنیم و با استفاده از متد apply تغییرات را اعمال می‌کنیم. createCLAHE 2 ورودی می‌گیرد که یکی میزان threshold برای محدودسازی مقدار تقویت است و دیگری سایز ناحیه‌بندی برای متعادل‌سازی هیستوگرام مشخص می‌کند.

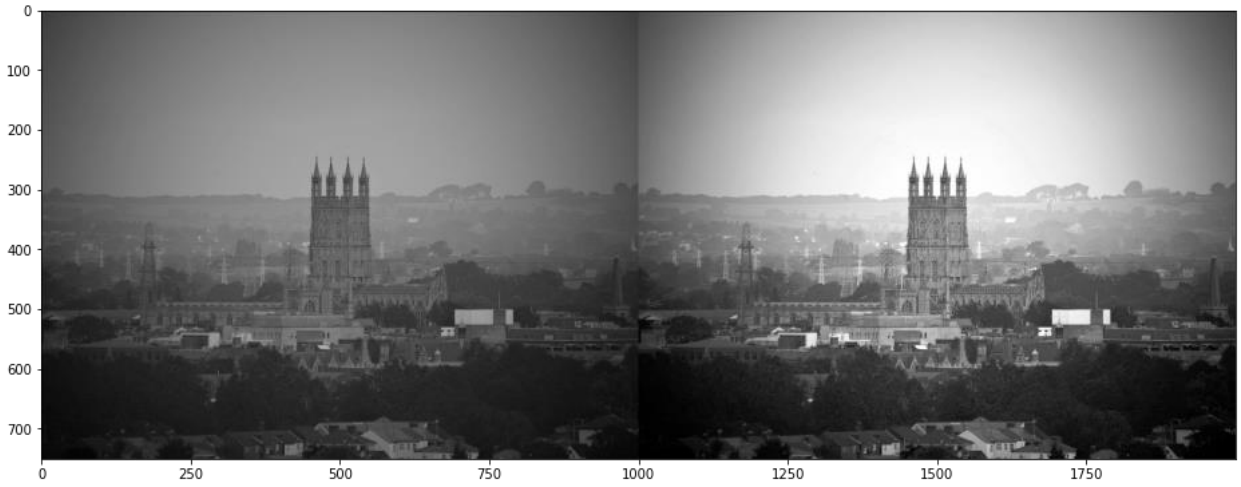
```
img = cv2.imread('River.jpg', 0)

### YOUR CODE ###
# START
clahe = cv2.createCLAHE(clipLimit=2.0)
clh = clahe.apply(img)
# END

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

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

نتیجه:

همانطور که مشخص است از حالت قبل بهتر است زیرا با محدودیت کنتراست تغییر کرده است.

پ)

متعادل‌سازی هیستوگرام پیاده سازی شده:



متعادل‌سازی هیستوگرام آماده:

CLAHE:

ت)

خیر چون در 3 کانال با هیستوگرام کار داریم و رنگ نهایی حاصل ترکیب این 3 کانال است، متعادل‌سازی هیستوگرام ممکن است رنگ‌ها را تغییر بدهد. برای بهتر شدن می‌توان میانگین 3 کانال را در نظر گرفت و هیستوگرام‌ها را به آن مپ کرد.

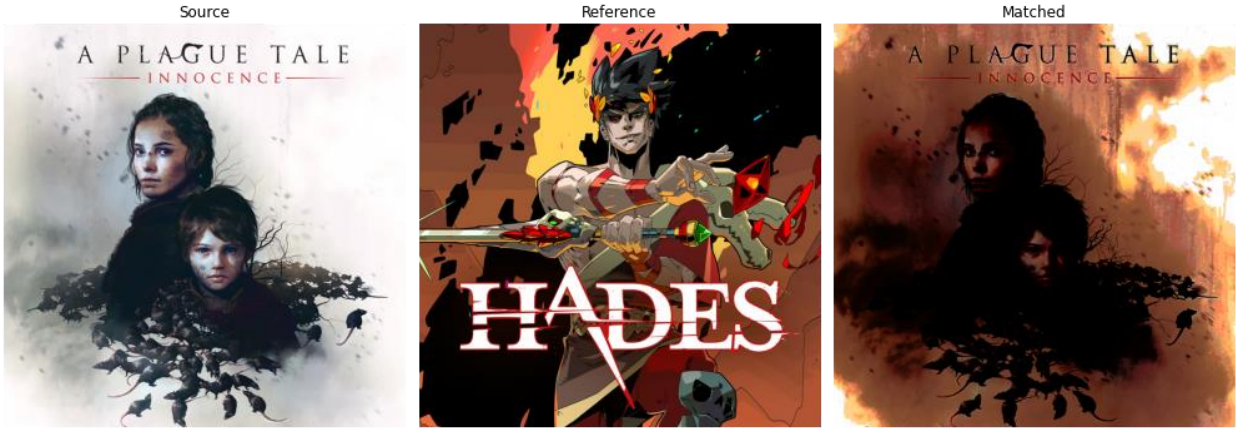
3-

الف)

تابع match_histograms در کتابخانه skimage 3 ورودی می‌گیرد. تصویر مبدا، تصویر مرجع و محور تعداد کانال تصویر که در اینجا آخرین بعد تصویر یعنی 3 می‌باشد.

```
### YOUR CODE ###
# START
matched = match_histograms(source, reference, channel_axis=-1)
# END
```

تصویر پس از تطبیق هیستوگرام:



همانطور که قابل مشاهده است رنگ های تصویر مبدا به خوبی به رنگ های تصویر 2 منطبق شده اند ولی فقط رنگ ها منطبق شده اند و کلیات تصویر از بین نرفته که هدف ما در تطبیق هیستوگرام همین می باشد.

ب)

برای تطبیق در 3 کانال،عملیات را جدا برای هر کدام اعمال می کنیم. ابتدا عین سوال اول cdf هر هیستوگرام را محاسبه می کنیم. و به ازای هر المان در cdf تصویر مبدا، مقدار برابر یا نزدیکترین را در cdf تصویر مرجع پیدا می کنیم که برای اینکار می توان از تابع np.interp استفاده کرد و در نهایت هیستوگرام را به آن نقاط مپ می کنیم.

```
### YOUR CODE ###
# START
matched = np.zeros(src_image.shape)
for i in range(3):
  src_histogram = np.histogram(src_image[..., i].flatten(), bins=256, range=(0, 255))[0]
  ref_histogram = np.histogram(ref_image[..., i].flatten(), bins=256, range=(0, 255))[0]

  src_cumsum = np.cumsum(src_histogram)
  ref_cumsum = np.cumsum(ref_histogram)

  src_cdf = src_cumsum / src_cumsum[-1]
  ref_cdf = ref_cumsum / ref_cumsum[-1]

  interp_t_values = np.interp(src_cdf, ref_cdf, np.arange(256))
  matched[..., i] = interp_t_values[src_image[..., i].flatten()].reshape(src_image[..., i].shape)
# END
```

نتیجه حاصل شده:

تصویر بدست آمده مانند حالت اول نیست و به این معناست که به دقت تابع آماده نیست ولی قابل مشاهده است که برخی نقاط منطبق شده اند.

پ)

استفاده از تابع آماده:

استفاده از تابع پیاده سازی شده:

منابع:

https://docs.opencv.org/4.x/d5/daf/tutorial_py_histogram_equalization.html

https://scikit-image.org/docs/dev/auto_examples/color_exposure/plot_histogram_matching.html

https://numpy.org/doc/stable/reference/generated/numpy.histogram.html?highlight=histogram#numpy.histogram

https://numpy.org/doc/stable/reference/generated/numpy.cumsum.html?highlight=cumsum#numpy.cumsum

https://stackoverflow.com/questions/32655686/histogram-matching-of-two-images-in-python-2-x