

-۱

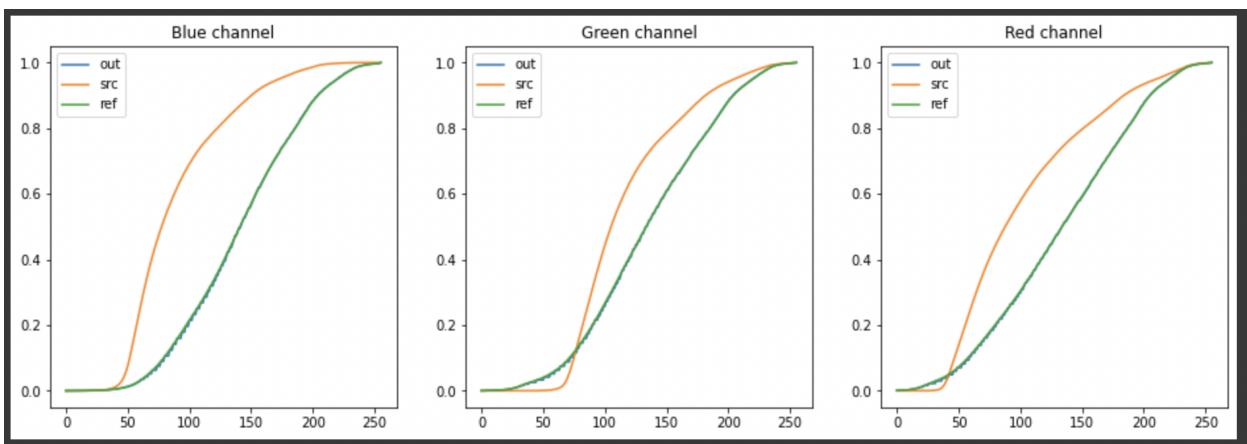
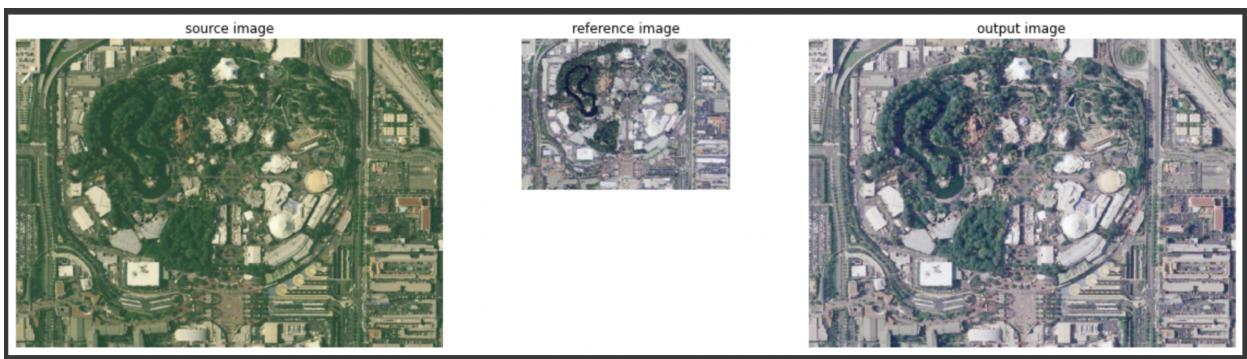
یک حلقه `for` تعریف میکنیم و روی بعد سوم ماتریس تصویر حرکت میکنیم که در واقع کانال های تصویر است.
برای هر بعد تصویر `src` و `ref` محاسبات زیر را انجام می دهیم:
هیستوگرام تصویر و سپستابع توزیع تجمعی آن هیستوگرام را محاسبه میکنیم و آن را نرمالیزه میکنیم تا تابع `cdf` به دست آید.

سپس با استفاده از دو حلقه `for` تو در تو روی پیکسل های تصویر حرکت میکنیم و عملیات زیر را انجام میدهیم:
مقدار پیکسل در تصویر `src` را با آرایه `cdf` تصویر `ref` مقایسه میکنیم و نزدیک ترین مقدار را پیدا میکنیم و با استفاده از متدهای `argmin` آن را بر می گردانیم.
این اندیس در واقع مقدار منتظر این پیکسل در تصویر `ref` است که با مقدار فعلی آن جایگزین میکنیم

```
# Your code
for i in range(src_image.shape[2]):
    ref_hist = calc_hist(ref_image[:, :, i])
    cum_ref=np.cumsum(ref_hist)
    cdf_ref=cum_ref/cum_ref[-1]

    src_hist = calc_hist(src_image[:, :, i])
    cum_src=np.cumsum(src_hist)
    cdf_src=cum_src/cum_src[-1]

    for row in range(src_image.shape[0]):
        for col in range(src_image.shape[1]):
            output_image[row,col,i]=np.argmin(np.abs(cdf_ref-cdf_src[src_image[row,col,i]]))
# Start
```



۲-الف

r_k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
n_k	1	20	25	100	50	40	15	4	0	0	0	0	0	0	1	0
$\sum_{j=0}^k n_k$	1	21	46	146	196											
$\sum_{j=0}^k \frac{n_k}{n}$	$\frac{1}{256}$	$\frac{21}{256}$	$\frac{46}{256}$	$\frac{146}{256}$	$\frac{196}{256}$											
$(L - 1) \sum_{j=0}^k \frac{n_k}{n}$					11.48											
Round					11											

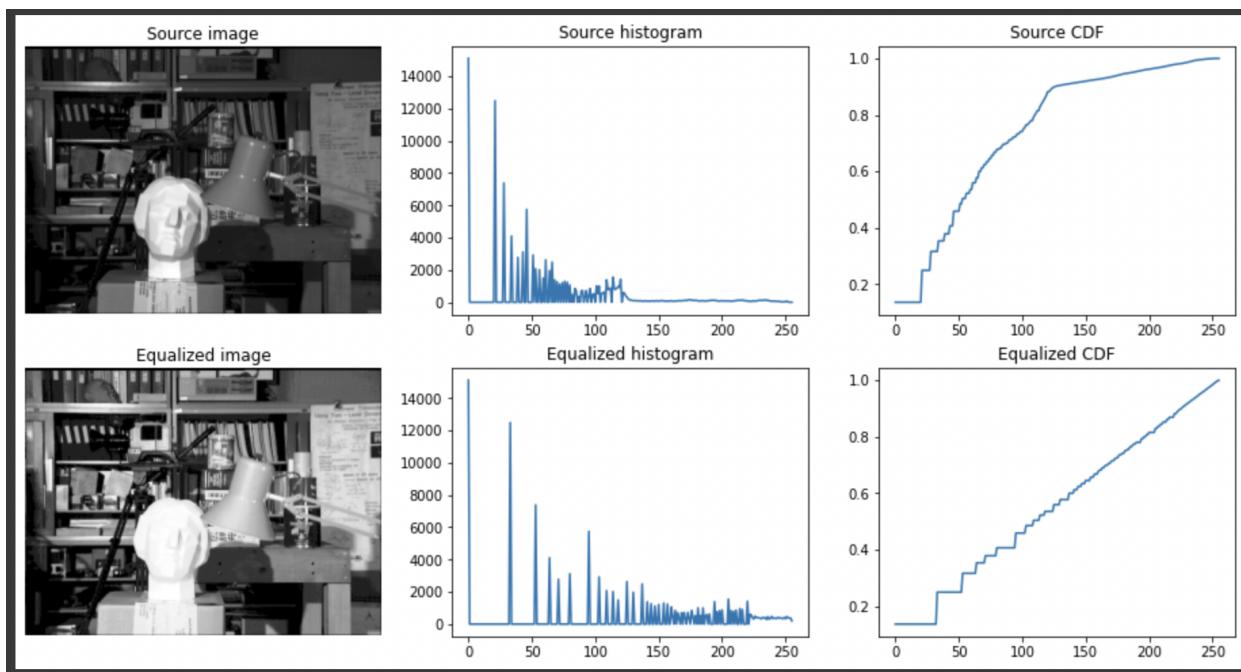
۲-ب

```
# Start
cv2.equalizeHist(image,output_image)
# End
```

۲-پ

خیر

کیفیت تصویر در نقاط تیره بهتر شد ولی در نقاط روشن کیفیت کنتراست تصویر کاهش یافت زیرا این الگوریتم با همه نواحی تصویر یکجور رفتار میکند.



باید روی پیکسل های تصویر حرکت کنیم و به ازای هر کدام روی پیکسل های اطراف آن در محدوده یک مربع با عرض مشخص محاسباتی را انجام دهیم، از آنجا که پیکسل های مرزی از یک طرف و یا در نقاط گوشی ای از دو طرف فاقد پیکسل های همسایه هستند، با استفاده از متده copyMakeBorder از کتابخانه opencv برای تصویر حاشیه میسازیم. با استفاده دو حلقه for تو در تو روی پیکسل های تصویر حرکت کنیم و روی پیکسل های همسایه آن که در مربعی با عرض مشخص شده توسط gridSize قرار دارند، تابع equalize_hist را اعمال می کنیم و مقداری که این تابع برمیگرداند را با مقدار فعلی پیکسل جایگزین میکنیم.

```

output = image.copy()
#your code here
# Start
borderSize_h = gridSize[0] // 2;
borderSize_v = gridSize[1] // 2;
borderType = cv2.BORDER_REPLICATE;
image2 = cv2.copyMakeBorder(image,borderSize_v,borderSize_h,borderSize_h,borderType)
for i in range(borderSize_h,image2.shape[0]-borderSize_h):
    for j in range(borderSize_v,image2.shape[1]-borderSize_v):
        point=equalize_hist(image2[i-10:i+10, j-10:j+10])
        output[i-borderSize_h,j-borderSize_v]=point[borderSize_h,borderSize_v]
# End

```



تابع `get_clipped_hist`، هیستوگرام تصویر وردی را محاسبه میکند، بر اساس مقدار `limit` تعیین شده، آن را برش می زند و مجموع مقادیر برش زده شده را به کل هیستوگرام اضافه میکند.

به این صورت که روی المان های آرایه هیستوگرام حرکت میکند و مقدار آن المان را با `limit` مقایسه میکند، اگر از `limit` بیشتر بود، آن المان را برابر مقدار `limit` قرار میدهد و مابه تفاوت آن را با متغیر `sum` جمع میکند. در آخر مقدار `sum` را بین آرایه های هیستوگرام توزیع میکند.

```
# Start

def get_clipped_hist(grid, clip):
    grid_hist = calc_hist(grid)

    clipped_hist_grid = np.zeros_like(grid_hist)

    var = 0
    for index, i in enumerate(grid_hist):
        if (i - clip) > 0:
            clipped_hist_grid[index] = clip
            var += (i - clip)
        else:
            clipped_hist_grid[index] = i

    return clipped_hist_grid + (var // 256)
```

مشابه تابع `ahe` برای تصویر حاشیه میسازیم، روی پیکسل های تصویر حرکت می کنیم و به ازای هر پیکسل، عملیات زیر را انجام میدهیم:

تابع `get_clipped_hist` را روی محدوده ای مربعی، با عرض `grid_size` اطراف پیکسل را اعمال می کنیم، بر روی هیستوگرامی که این تابع برミ گرداند، تابع `cdf` را مشابه سوال اول محاسبه می کنیم و در آخر با استفاده از این تابع `cdf` مقدار جدید را برای آن پیکسل محاسبه میکنیم، به این صورت که مقدار فعلی پیکسل را به تابع `cdf` می دهیم و نتیجه را با مقدار فعلی پیکسل جایگزین میکنیم.

```
borderSize_h = gridSize[0] // 2
borderSize_v = gridSize[1] // 2

borderType = cv2.BORDER_REPLICATE

image2 = cv2.copyMakeBorder(image, borderSize_v, borderSize_v, borderSize_h, borderSize_h, borderType)

for i in range(borderSize_h, image2.shape[0]-borderSize_h):
    for j in range(borderSize_v, image2.shape[1]-borderSize_v):
        grid = image2[i-borderSize_h:i+borderSize_h+1, j-borderSize_v:j+borderSize_v+1]

        new_region_hist = get_clipped_hist(grid, clip_limit)

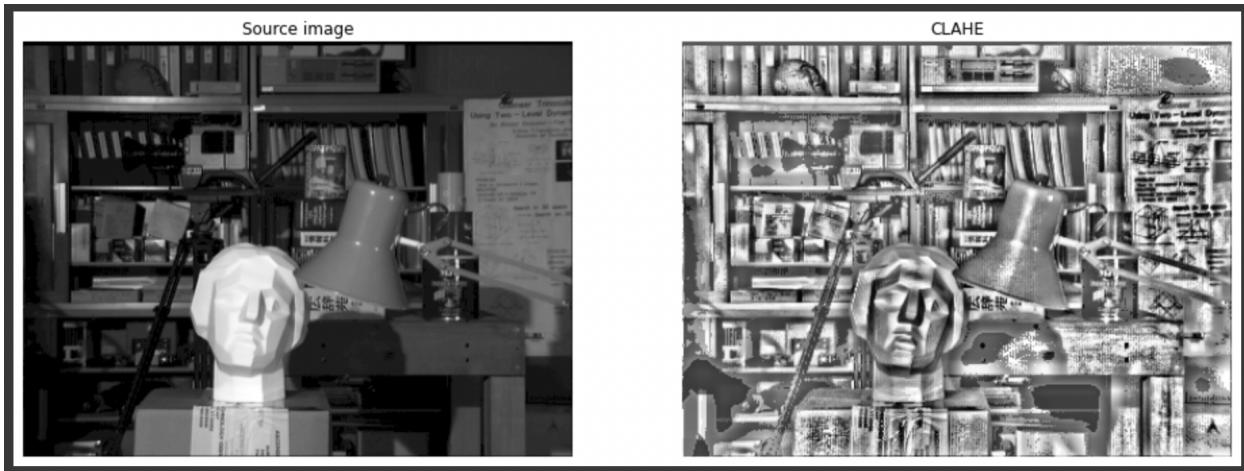
        cum = np.cumsum(new_region_hist)
        cdf = cum / cum[-1]
        final_equal = [int(elem) for elem in 255 * cdf]

        output[i-borderSize_h, j-borderSize_v] = final_equal[image[i-borderSize_h, j-borderSize_v]]
```

End

```
return output
```

نتیجه به این صورت خواهد بود:



-ب

برای ساخت کرنل گاوی

ابتدا با استفاده از متد `np.linspace` آرایه ای میسازیم که به تعداد `size` تا المان داشته باشد
مقدار ابتدا و انتهای آن را از $(-size/2)$ تا $(size/2)$ قرار میدهیم
با استفاده از `np.exp` و `np.square` تابع گاوی را میسازیم
و سپس با استفاده از `np.outer` آن را در خودش ضرب خارجی میکنیم تا ماتریسی دو بعدی به دست آید
در آخر آن را بر مجموع مقادیرش تقسیم میکنیم

```
#your code here
# Start
l= size[0]
ax = np.linspace(-(l - 1) / 2., (l - 1) / 2., l)
gauss = np.exp(-0.5 * np.square(ax) / np.square(std))
kernel = np.outer(gauss, gauss)
return kernel / np.sum(kernel)
# End
```

-الف

با استفاده از متد `filter2d` کرنل ساخته شده را بر روی تصویر اعمال میکنیم

```
# Start
output = cv2.filter2D(src=image, ddepth=-1, kernel=kernel)
# End
```

نتیجه به این صورت خواهد بود:



۴-پ

با توجه به فرمول گواس:

$$G(r, \sigma) = K e^{-\frac{r^2}{2\sigma^2}}$$

هرچه قدر مقدار σ را بیشتر کنیم، فیلتر به سمت فیلتر متوسط گیر می‌رود یعنی مقدار بلور کردن آن بیشتر می‌شود

$$\lim_{\sigma \rightarrow \infty} e^{-\frac{r^2}{2\sigma^2}} = e^{\sigma \rightarrow \infty -\frac{r^2}{2\sigma^2}} = e^0 = 1$$

1	1	1
1	1	1
1	1	1

و هرچه قدر مقدار σ را کمتر کنیم، فیلتر به سمت فیلتر همانی می‌رود یعنی مقدار بلور کردن آن کمتر می‌شود

$$\lim_{\sigma \rightarrow 0} e^{-\frac{r^2}{2\sigma^2}} = e^{\sigma \rightarrow 0 -\frac{r^2}{2\sigma^2}} = e^{-\infty} = 0$$

0	0	0
0	1	0
0	0	0

Refrences:

<https://learnopencv.com/image-filtering-using-convolution-in-opencv/>

<https://stackoverflow.com/questions/29731726/how-to-calculate-a-gaussian-kernel-matrix-efficiently-in-numpy>