Computer vision

Homework 3

Parsa Eissazadeh-97412364

سوال 1

در ابتدا فرکانس هر سطح روشنایی را مینویسیم .

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

در جدول بالا ، ردیف دوم اعداد هیستوگرام را نشان می دهد .

{ شکل هیستوگرام و اینجا اضافه کن، با کاغذ بکش عکسشو بذار اینجا، یا با یه کد پایتون بکش }

برای برش 5 درصد از اطلاعات بالا و پایین عکس اعداد هیستوگرام را sort میکنیم :

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

اگر 10 درصد از بالا و پایین را حذف کنیم این اعداد باقی می مانند :

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

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	0.08	0.08	0.2	0.29	0.41	0.7	1	1

$(L-1)\sum_{j=0}^{k}\frac{n_j}{n}$	0	0	0.72	0.72	1.8	2.7	3.6	6.3	9	9
round	0	0	1	1	2	3	3	6	9	9

در نهایت به شکل زیر میشه :

6	6	9	9	9	9
1	0	2	2	2	9
6	0	3	3	1	9
9	0	3	9	9	6
9	6	3	3	6	6

سوال 2

الف)

همانطور که در سوال قبل هم نشان داده شد ، مراحل متعادل سازی هیستوگرام به شکل زیر است :

1. محاسبه چگالی احتمال
2. محاسبه توزیع تجمعی
3. محاسبه T

برای پیاده سازی به ترتیب این ها را پیاده سازی می کنیم .

محاسبه چگالی احتمال :

```
for pixel in np.nditer(image) :
  pdf[np.round_(pixel)] += 1
```

از متد nditer برای پیمایش numpy array استفاده می کنیم .

محاسبه توزیع تجمعی :

```
cdf = np.zeros(BRIGHTNESS_LAYERS)
cdf[0] = pdf[0]
for pixel_iterator in range(1,BRIGHTNESS_LAYERS) :
  cdf[pixel_iterator] = cdf[pixel_iterator-1] + pdf [pixel_iterator]
```

که تابع پیچیده ای نیست و صرفا مقدار pdf را با جمع مقدار قبلی cdf جمع می کند . با متد np.cumsum هم می شد این جمع انباشته ای را محاسبه کرد .

در مرحله آخر تابع تبدیل را میسازیم .

```
height , width = image.shape
n = height * width
tr = (cdf / n) * (BRIGHTNESS_LAYERS - 1)
```

سپس عکس خروجی را میسازیم .

```
output_image = np.zeros((1,height * width))

output_index = 0
for pixel in np.nditer(image) :
  output_image[0,output_index]= tr[np.round_(pixel)]
  output_index += 1

output_image = output_image.reshape(image.shape)
assert image.shape == output_image.shape
# End

return output_image
```

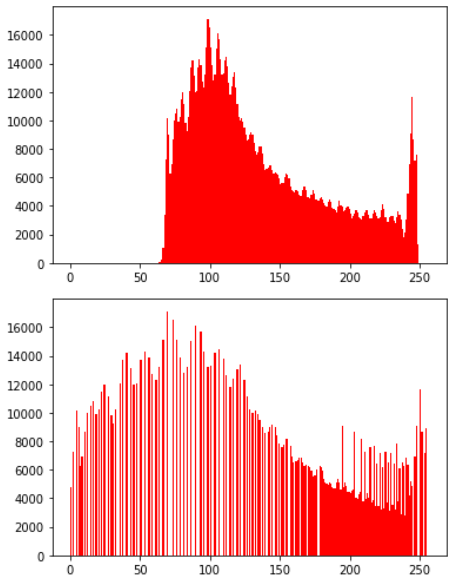
نتیجه به شکل زیر شد :

همانطور که مشاهده می شود کیفیت عکس بالا رفته است .

با استفاده از قطعه کد زیر histogram این دو عکس را رسم کردم :

```
plt.hist(equ.flatten(),256,[0,256], color = 'r')
plt.show()
```

هیستوگرام عکس از بالا به پایین تغییر کرد ، دیده می شود که رنگ ها پخش تر شده اند و از ۰ شروع شده اند و به تقریبا ۲۵۶ رسیده اند :



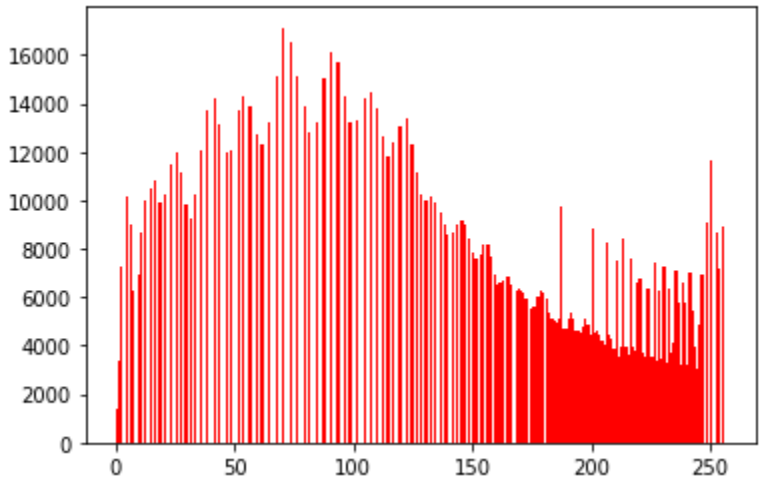
برای استفاده از متد آماده openCV برای histogram equalization ، از لینک زیر کمک گرفتم :

و کد به شکل زیر بود :

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

و نتیجه به شکل زیر :

هیستوگرام عکس جدید :



که شباهت بالایی با هیستوگرام پیاده سازی خودم دارد .

ب)

برای استفاده از متد آماده openCV برای CLAHE ، از لینک زیر کمک گرفتم :

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

و کد به شکل زیر بود :

```
### YOUR CODE ###
# START

clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
clh = clahe.apply(img)
# END
```

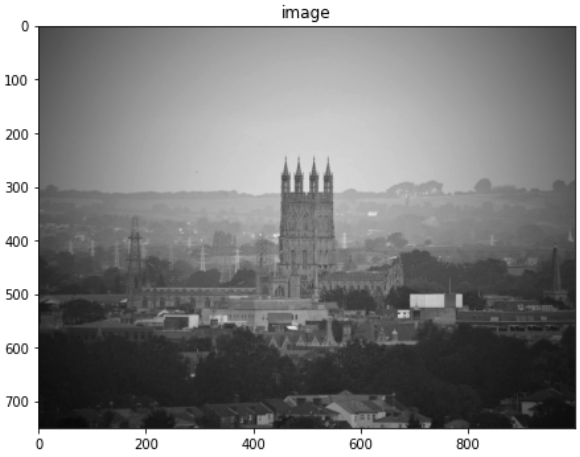
و نتیجه به شکل زیر بود :

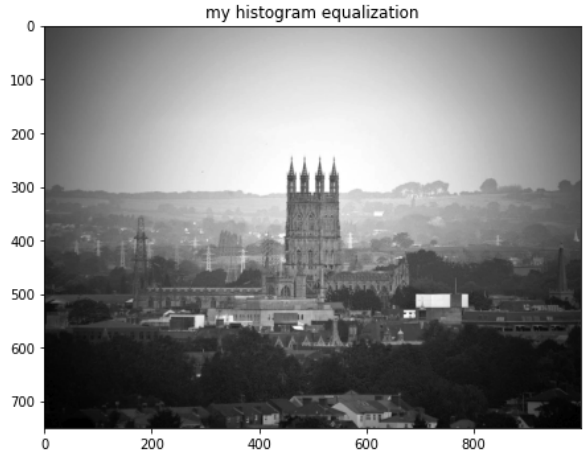
کیفیت عکس به مراتب افزایش یافت زیرا همه عکس به شکل یکدست تاریک یا روشن نشد ، بلکه بر حسب روشنایی قبلی عکس روشنایی ها به شکلی محلی تغییر کرده در نتیجه در تونل زیر پل ، عکس روشن تر شده است .

پ)

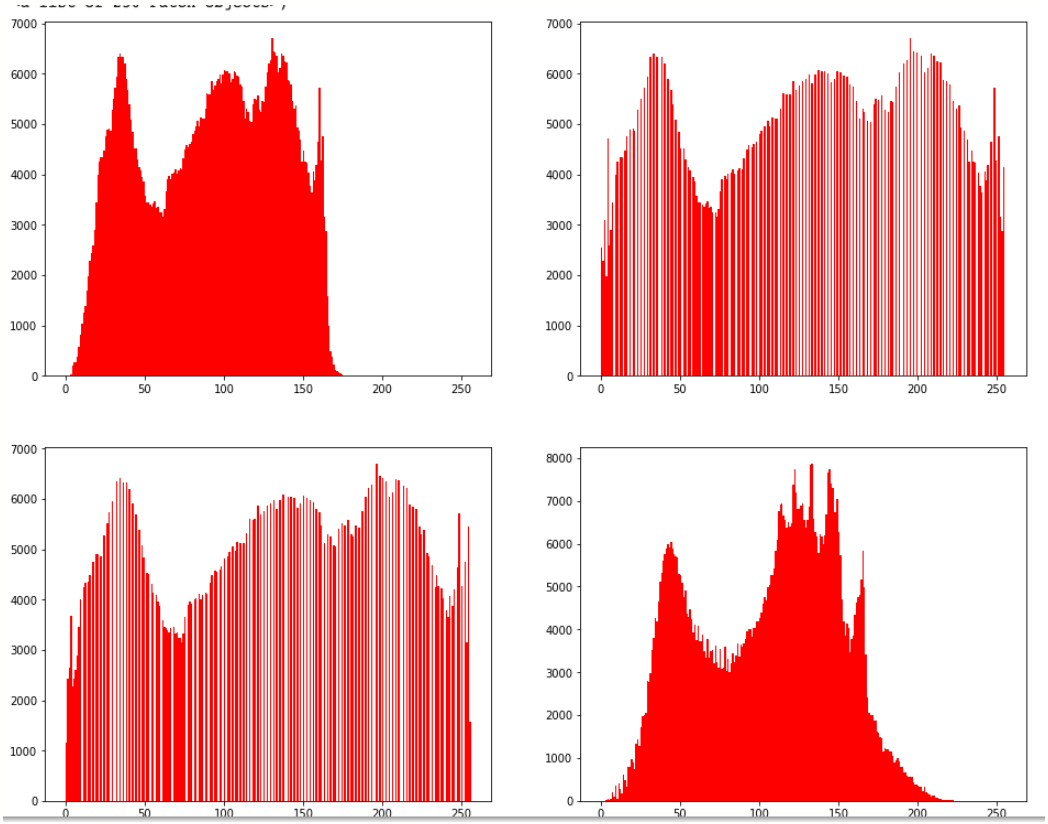
در نهایت هر سه متد را روی عکس City پیاده کردم و به کمک متد subplot آن ها را نمایش دادم :

{عکس در صفحه بعد }





هیستوگرام هر ۴ عکس را به همین ترتیب رسم کردم :



در عکس کلاهه ، رنگ ها به اندازه دو عکس دیگر باز نشده اند هر چند در عکس کلاهه اصلا هدف این نیست که همه رنگ ها به شکلی یک دست باز شوند .

سوال ۳

الف)

مم

ب)

ما در ابتدا به تبدیل متعادل سازی هیستوگرام هر عکس نیاز داریم .

به عکس دوم ، تابع متعادل سازی هیستوگرامش را اعمال می کنیم . سپس معکوس تبدیل عکس اول را بر روی آن اعمال می کنیم . تبدیل برابر است با

$$(L-1) * cdf$$

از آنجایی که L-1 ثابت است ، ما فقط cdf هر دو عکس را حساب می کنیم . برای اینکه معکوس اعمال شود ، چک میکنیم که cdf ها کجا برابر هستند ، اگر مثلا در عکس اول cdf رنگ i برابر بود با cdf عکس دوم در j ، همه ی j ها در عکس دوم را برابر با i میگذاریم .

قبل از محاسبه cdf ها عکس ها را به سه کانال تبدیل می کنیم . با هر یک از این کانال ها همانند یک عکس gray scale رفتار میکنیم .

متد split_channels را نوشتم :

```
def split_channels(img):

  channels_count = img.shape[2]
  channels = [img[:,:,i] for i in range(channels_count)]

  return channels_count,channels
```

که تعداد کانال ها و خودشان را بر میگرداند . برای هر دو عکس این متد را فراخوانی کردم و سپس روی کانال ها حلقه for زدم .

```
channels_count , src_channels = split_channels(src_image)
channels_count , ref_channels = split_channels(ref_image)

output_image = []
for i in range(channels_count):
  src_channel = src_channels[i]
  src_cdf = compute_cdf(src_channel)

  ref_channel = ref_channels[i]
  ref_cdf = compute_cdf(ref_channel)

  transition = match_histogram_channels(src_cdf,ref_cdf)
  output_image.append(transition[src_channel])
```

هر کانال هر عکس اول وارد متد compute_cdf میشه که به شکل زیر هستش :

```
BRIGHTNESS_LAYERS = 256
def compute_cdf (image):
  pdf = np.zeros(BRIGHTNESS_LAYERS)

  for pixel in np.nditer(image) :
    pdf[np.round_(pixel)] += 1

  cdf = np.cumsum(pdf)

  return cdf
```

اینجا پیاده سازی تقریبا شبیه پیاده سازی سوال قبل هست با این تفاوت برای محاسبه خود cdf از متد cumsum کتابخانه numpy استفاده کردم که مخفف cumulative sum هست که جمع انباشته ای رو حساب میکنه . بعد از این که cdf ها رو محاسبه کردیم ، وارد متد match_histogram_channel میشن که به شکل زیر هست :

```
def match_histogram_channels(src , ref):

  transition= np.zeros((BRIGHTNESS_LAYERS))
  for i in range(1,BRIGHTNESS_LAYERS):
    for j in range(1,BRIGHTNESS_LAYERS):
      if ref[i] <= src[j]:
        transition[j] = i
        break

  transition[BRIGHTNESS_LAYERS-1] = BRIGHTNESS_LAYERS-1
  return transition
```

از اونجایی که cdf یک عکس صعودی هست ، ما در حلقه for اولین باری که به جایی می رسیم که cdf یک عکس از اون یکی کمتره میفهمیم که cdf توی این دو نقطه از این دو عکس به هم نزدیک هستن ، پس به محض اینکه به اونجا برسیم transition همون پیکسل تو عکس اول رو مقدار دهی میکنیم و بعد break می کنیم .

در نهایت تبدیل رو return می کنیم . numpy این قابلیت و داره که با یه آرایه مثل یه متدی رفتار کنه که ورودی هاش index ها هستن و خروجی هاشون مقدار آرایه تو اون index . در نتیجه ما کل کانالی که از عکس اول به دست اوردیم رو بهش پاس میدیم و خودش میاد دونه دونه عناصر اون آرایه تو اون تابع می ذاره .

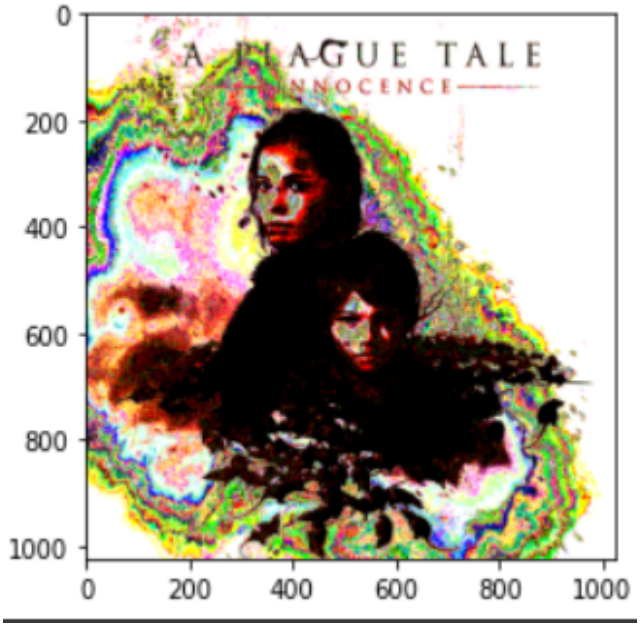
این شکلی به کانال جدید می رسیم و همه رو به هم اضافه می کنیم و در نهایت با متد dstack اون ها رو تو 3 بعد عکس پخش می کنیم .

```
output_image = np.dstack((output_image[0],output_image[1],output_image[2]))

assert output_image.shape == (1024,1024,3)
```

و یک چکی هم میذاریم برای عکس تا shape درستی داشته باشه و اگر این گونه نبود برنامه ارور بده .

عکس را نمایش دادم و در ابتدا به شکل زیر شد :



کیفیت عکس خیلی از بین رفت .

پ)

مم