تمرین اول – بخش عملی

سوال دوم – گزارش کار

مهرشاد فلاح اسطلخزیر 401521462

منطق کد:

در ادامه ابتدا به پیاده‌سازی هر تابع و دلایل استفاده از هر تابع numpy می‌پردازم.

تابع calc_hist:

```python
def calc_hist(image):
    '''
    Do not use libraries
    calculate image histogram
    input(s):
      image (ndarray): input image
    output(s):
      hist (ndarray): computed input image histogram
    '''
    # hist = np.zeros(256,dtype=int)

    # for i in range(image.shape[0]):
    #     for j in range(image.shape[1]):
    #         hist[image[i][j]] += 1

    #Without Using Loop
    hist, _ = np.histogram(image, bins = 256, range = (0, 256))

    return hist
```

برای این تابع می‌توان گفت دو کد موجود است. کد اول مثل سوال یک حلقه تو در تو در عکس می‌زند و در خانه‌های آرایه مقدار بدست آمده را ذخیره می‌کند. در حالت دوم از تابع histogram در NumPy استفاده می‌کنیم. پارامتر bins را برای گسسته‌سازی مقادیر خروجی در هیستوگرام استفاده کردیم و رنج هم مشخصا بازه اعداد ما را نشان می‌دهد. خروجی این تابع هیستوگرام و مقادیر bin هست که به آن نیازی نداریم و فقط در تابع calc_hist مقدار هیستوگرام را برمی‌گردانیم.

تابع calc_cdf:

```python
def calc_cdf(channel):
    '''
    Do not use libraries
    calculate image cdf
    input(s):
        channel (ndarray): input image channel
    output(s):
        cdf (ndarray): computed cdf for input image channel
    '''
    # cdf = np.zeros(256, dtype=float)
    # n = channel.shape[0] * channel.shape[1]
    histogram = calc_hist(channel)
    # current = 0

    # for i in range(256):
    #     current += histogram[i]
    #     cdf[i] = current / n

    # Without using Loop
    cdf = np.cumsum(histogram) / np.sum(histogram)

    return cdf
```

وظیفه این تابع این است که مقدار تابع توزیع تجمعی را برای ما محاسبه کنید برای این کار هیستوگرام عکس را از تابع calc_hist بدست می‌آوریم و پس از آن با استفاده از تابع cumsum مقدار جمع تجمعی را بدست آورده و در نهایت برای نرمالیزه شدن آن را بر جمع کل مقادیر هیستوگرام تقسیم می‌کنیم. بدین ترتیب مقدار cdf هر رنگ هم بدست می‌آید.

تابع hist_matching:

```python
def hist_matching(src_image, ref_image):
  '''
  don't use libraries
  input(s):
    src_image (ndarray): source image
    ref_image (ndarray): reference image
  output(s):
    output_image (ndarray): transformation of source image so that its histogram matches histogram of reference image
  '''
  output_image = src_image.copy()
  channels = [(0, 'Blue channel'), (1, 'Green channel'), (2, 'Red channel')]
  for channel, title in channels:
    src_cdf = calc_cdf(src_image[:, :, channel])
    ref_cdf = calc_cdf(ref_image[:, :, channel])

    table = np.zeros(256, dtype=int)
    ref_inverse = np.zeros(256, dtype=int)

    # for i in range(256):
    #   ref_inverse[int(ref_cdf[i] * 255)] = i
    # Without Using Loops

    ref_inverse[(ref_cdf * 255).astype(int)] = np.arange(256)

    ref_inverse = np.maximum.accumulate(ref_inverse) # For lost values

    table = ref_inverse[(src_cdf * 255).astype(int)]


    # for i in range(256):
    #   table[i] = ref_inverse[int(src_cdf[i] * 255)]

    output_image[:, :, channel] = table[src_image[:, :, channel]]

  return output_image
```

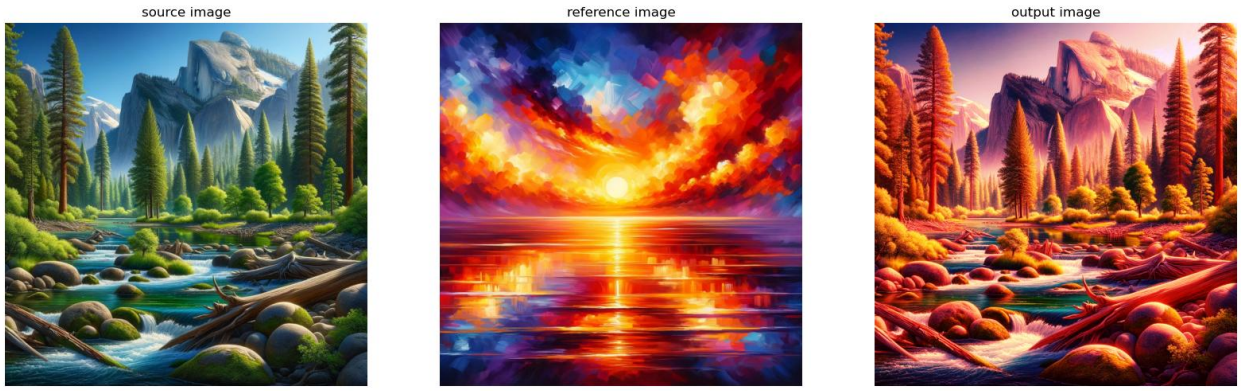
وظیفه این تابع تطبیق هیستوگرام است و برای این کار از هر سه کانال رنگی آبی و سبز و قرمز این کار را به صورت جداگانه انجام می‌دهد. ابتدا یک حلقه برای حرکت در کانال‌ها می‌زنیم و بعد cdf هر دو عکس ورودی و هدف را با کمک تابع calc_cdf محاسبه می‌کنیم. در مرحله بعد دو آرایه تعریف می‌کنیم. آرایه table برای این است که نگاشت مقادیر جدید رنگ عکس ورودی را در آن ذخیره کنیم و آرایه ref_inverse هم برای این است که مقدارهای cdf را با هم تطبیق بدهیم. در مرحله بد مقادیر ref_inverse را با استفاده از cdf عکس هدف بدست آوریم دلیل ضرب در عدد 255 برای این است که مقدار نرمال شده cdf به 0 تا 255 برگردد و حتما باید عدد صحیح باشد. np.arange هم یک آرایه 256 تایی از 0 تا 255 ایجاد می‌کند. عملا همان معکوس cdf عکس هدف را بدست آوردیم. صرفا بعضی مقادیر مقدار 0 دارند که خب این درست نیست برای همین در خط بعد از np.maximum.accumulate استفاده کردیم که مقادیر صفر گمشده را با مقدار قبلی جایگزین می‌کند. در نهایت برای تشکیل table هم کافیست به ازای تمامی مقادیر 0 تا 255 ابتدا cdf آن‌ها را گذاشته و در ref_inverse بدهیم و به این ترتیب رنگ جدید به ازای هر رنگ قبلی پیدا می‌شود و در آخر کار هم عکس خروجی را با کمک table نمایش می‌دهیم.

نکته: در تمامی توابع بالا از تکنیک vectorization استفاده شده و از حلقه استفاده نشده اگر چه کامنت‌ها کد با حلقه درست است.

در آخر کار هم کدهایی که به صورت اولیه داده شده وظیفه نمایش عکس ورودی و هدف و در نهایت عکس خروجی را دارد.

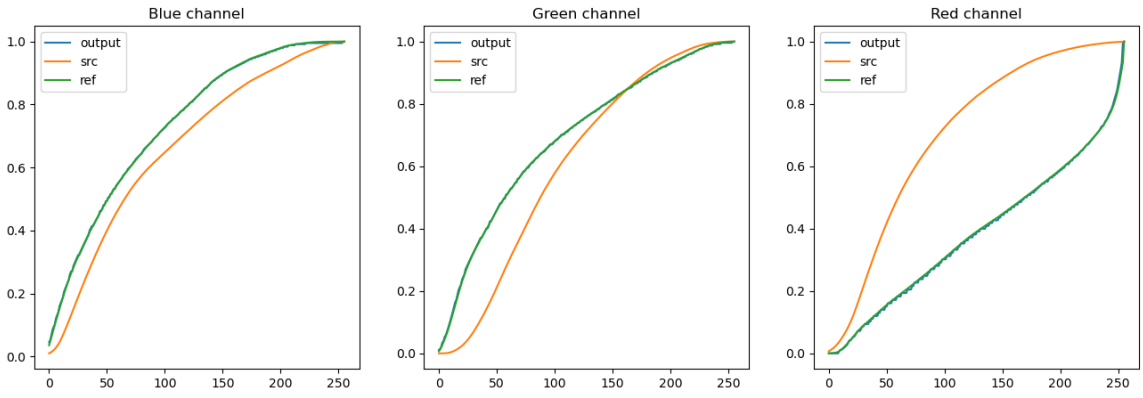
تحلیل نتایج:

عکس خروجی:



همانطور که در شکل بالا که همان خروجی کد است پیداست عکس سمت چپ با عمل تطبیق هیستوگرام در هر سه کانال رنگی قرمز و سبز و آبی از نظر رنگی مشابه عکس هدف که عکس وسط باشد شده و تقریبا رنگ‌های سبز به سمت رنگ قرمز میل کرده و کل عکس تقریبا تم نارنجی در آن غالب شده و عمل تطبیق هیستوگرام با موفقیت انجام شده.

نمودارهای خروجی:



همانطور که در نمودار بالا معلوم است خط سبز مقدار توزیع تجمعی عکس هدف، خط نارنجی مقدار توزیع تجمعی عکس ورودی و خط آبی که دقیقا مشخص نیست و تقریبا با سبز هم‌راستا شده مقدار توزیع تجمعی عکس خروجی است و همانطور که مشخص است نمودار ورودی به نمودار خروجی نزدیک شده در هر سه کانال رنگی.