به نام خدا

اصول پردازش تصویر

تمرین سری اول

محمدعلی علما – ۹۸۱۰۰۴۹۷

استاد درس

دکتر مصطفی کمالی تبریزی

نیمسال اول ۱۴۰۱–۱۴۰۰

نکات تکمیلی :

۱. برای ران کردن کد ها ، نیاز به لایبرری های OpenCV ، Numpy و Matplotlib داریم.
۲. عکس های ورودی را در پوشه resources قرار دهید.
۳. بعد از اجرای هر کد ، نتایج آن در پوشه Result قرار میگیرد. نتایج اصلی در همین پوشه و در صورتی که نتایج دیگری نیز ایجاد شده باشد، در زیرپوشه های دیگری به تفکیک هر تمرین قرار گرفته است.
۴. در لینک زیر نیز میتوانید نتایج را مشاهده کنید :

https://mega.nz/folder/CpBgFBDb#txWC1Ca-sS0aL-5YgjNP7Q

۵. نتایج res01 تا res11 در پوشه اصلی لینک بالا قرار گرفته اند. در صورتی که برای تمرینی، نتایج و خروجی های دیگری نیز درست شده باشد، آنها را به تفکیک هر تمرین در زیر پوشه ها قرار داده ام.
۶. در اکثر تمرین ها، برای انکه نتیجه محاسبات دقیقتر باشد، فرمت عکس ورودی (که uint8 هست) را تبدیل به float میکنم.

مشورت های انجام شده :

سوال ۳ :

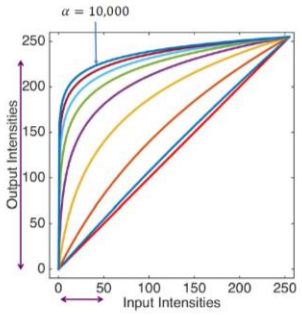
۱. سید امیرمحمد سادات شکوهی، سید عرفان موسویان (از دانشجویان سال قبل) برای نحوه مقایسه و مچ کردن تصاویر(پیشنهاد برای استفاده از مچ کردن edge ها)
۲. علی شفیعی : ایده ی اولیه ی نحوه کراپ کردن

سوال ۶ :

۱. رضا پیشکو : پیشنهاد نحوه محاسبه تقریبی تابع وارون (که در توضیحات سوال ۶ آن را به طور کامل شرح داده ام.)

سوال اول)

پس از لود کردن عکس ، برای آنکه محاسبات ما دقیق تر باشد، تایپ عکس را به float64 **تغییر میدهیم. سپس با اعمال تابع لگاریتمی زیر، کیفیت تصاویر را کمی بهبود میبخشیم :**



$$y = \frac{255\ \log(1+\alpha\ x)}{\log(1+255\ \alpha)}$$

که پارامتر α **را برابر** 0.05 **قرار میدهیم.**

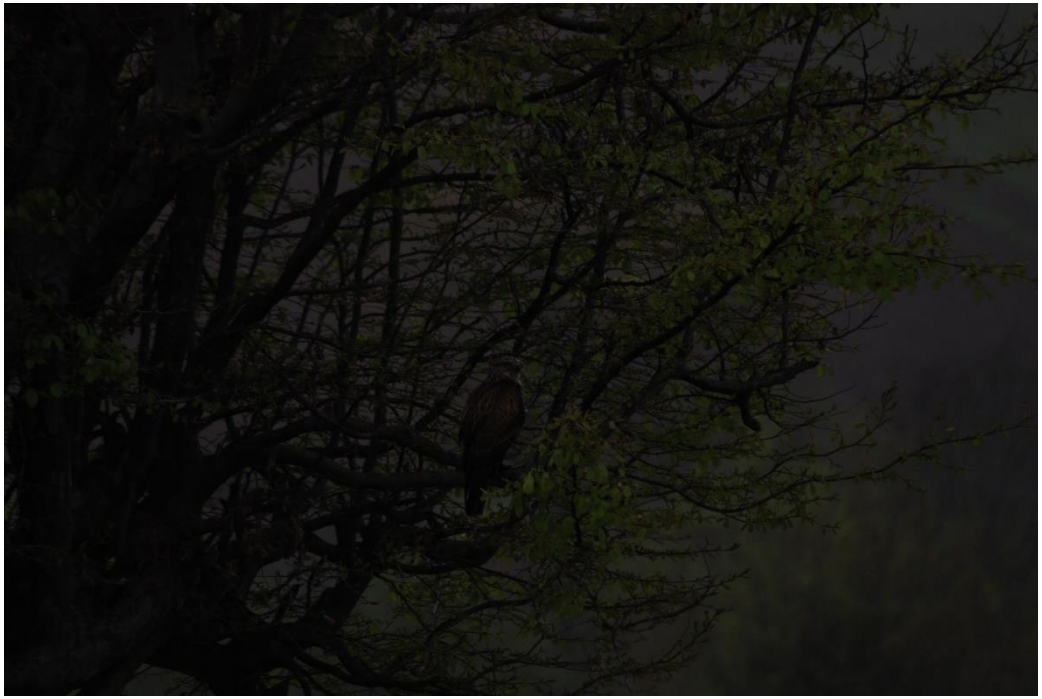
$\alpha = 0.001, 0.01, 0.1, 1, 10, 100, 1{,}000, 10{,}000$

با این عمل، عکس ما به صورت زیر در می آید :

این عکس را میتوانید در پوشه q1 **در لینک زیر مشاهده کنید .(یا پس از اجرای کد در مسیر** Result/q1/res1_JustLogFunction.jpg **آن را مشاهده کنید.)**

بعد از آن، عکس را به فضای رنگی HSV **میبریم و سپس مقادیر** V **تمام پیکسل ها را به اندازه ی** 15 **واحد اضافه میکنیم. این کار باعث افزایش روشنایی تصویر میشود و نتیجه نهایی مانند تصویر روبرو خواهد شد :**

مقایسه تصاویر اولیه و نهایی :



سوال ۲)

ابتدا عکس را لود میکنیم. سپس عکس را به فضای رنگی HSV میبریم. بعد بر روی کانال v ، یک تابع لگاریتمی با پارامتر ۰.۱ مانند زیر اعمال میکنیم :

$$Y = \frac{255Log(1+\alpha X)}{Log(1+255\alpha)}$$

و عکس بدست آمده را ذخیره میکنیم.

عکس اولیه :

عکس نهایی :

سوال سوم)

نام عکس را در متغیر image_name (خط 145) قرار دهید و برنامه را اجرا کنید. برنامه به صورت خودکار یک پوشه برای خروجی های این عکس در مسیر Result/q3/ ایجاد میکند و نتایج مراحل مختلف را ذخیره میکند.

روش کلی کار به این صورت که ابتدا عکس را در در متغیر image لود میکنیم. سپس در ۵ لایه (از لایه ۴ تا لایه ۰ که عکس اصلی است) شروع به منطبق کردن لایه ها میکنیم. (پیرامید)

سایز هایی که در هر لایه بر روی آنها کار میکنیم بدین صورت است :

- لایه 0 : عکس اصلی
- لایه 1 : سایز عکس به ۵۰٪ کاهش می یابد. (½)
- لایه ۲ : سایز عکس به ۲۵٪ کاهش می یابد. (¼)
- لایه ۳ : سایز عکس به ۱۲٪ کاهش می یابد.(1/8)
- لایه ۴ : سایز عکس به ۶ ٪ کاهش می یابد.(1/16)

سپس در یک حلقه فور ، در هر مرحله، ابتدا با استفاده از تابع myResize() ، سایز عکس خود را کو چک میکنیم . سپس در خطوط ۱۶۸ الی ۱۷۳ ، طول و عرض عکس حاصل و همچنین سه کانال آبی و سبز و قرمز را به دست می آوریم. سپس با استفاده از تابع matcher() ، به طور جداگانه ابتدا کانال سبز را بر کانال آبی منطبق میکنیم و سپس کانال قرمز را بر کانال آبی منطبق میکنیم.(توضیحات تابع matcher() در ادامه داده میشود.) با انجام هرکدام از این عملیات ها ، ۴ متغیر زیر بدست می آیند :

- j_for_g_channel
- i_for_g_channel
- j_for_R_channel
- i_for_r_channel

این چهار متغیر، میزان تغییراتی را نشان خواهد داد که در انتها لازم است پیکسل بالاچپ هرکدام از کانال های سبز و قرمز را از انجا بر روی کانال آبی منطبق کنیم. (متغیر های j در راستای عمودی و متغیر های i در راستای افقی عمل خواهند کرد. چون این تغییرات نسبی است، فرض میکنیم مختصات بالاچپ ترین پیکسل کانال آبی 0 , 0 هست.

سپس در خطوط ۱۷۷ الی ۱۹۲ ، نتیجه ی نهایی تصویر حاصل شده در آن مرحله(عکس خالص) را بدست می آوریم و سیو میکنیم.(با اسم bestLayer{i}.jpg)

بعد در خطوط ۱۹۴ تا ۲۰۷ ، حاشیه عکس خالص بدست آمده در آن مرحله ، را کراپ میکنیم و نتیجه را با نام layer{i}finalCrop.jpg ذخیره میکنیم.(توضیحات این کار در ادامه)

در نهایت در خطوط ۲۰۹ الی ۲۲۰ ، رنج تغییراتی را که لازم است در مرحله بعد مورد آزمایش قرار دهیم، را محاسبه میکنیم. از آنجا که در هر مرحله، سایز عکس دو برابر میشود، به طور حدودی پیش بینی میکنیم که مقادیر مورد قبول هر یک از چهار متغیر بالا نیز دو برابر شود و برای اطمینان یک بازه ی (-5 , +5) اطراف انها را در نظر میگیریم.

تابع matcher() :

در این تابع به عنوان ورودی، کانال آبی و یکی از کانال های سبز یا قرمز به عنوان channel، بازه تغییرات در راستای افقی (xmin , xmax) بالاچپ ترین پیکسل channel و بازه تغییرات در راستای عمودی(ymin , umax) بالا چپ ترین پیسکل channel را به ورودی میدهیم . سپس یک متغیر leastvalue میسازیم که در ابتدا مقدار آن مثبت بی نهایت است.(از این متغیر برای پیدا کردن یک سازگاری با کمترین اختلاف استفاده میکنیم.)

سپس در دو حلقه تو در تو، ابتدا در خطوط ۱۲۴ الی ۱۲۷ ، بازه های که برای کانال آبی و کانال channel نیاز داریم را می یابیم.(توضیحات اینکار را در ادامه خواهم داد.)

سپس در خطوط ۱۲۹ و ۱۳۰ ، یک زیرآرایه از کانال آبی و channel انتخاب میکنیم. در ادامه به وسیله تابع edgeDetector() مشتق اول دو زیر آرایه را بدست می آوریم.(در واقع زیرآرایه ای بدست می آوریم که در آن edge های تصویر مشخص شده اند.) سپس دو زیر آرایه بدست آمده را از یکدیگر کم کرده

و مجموع قدر مطلق تمام درایه های آنها را بدست می آوریم. در صورتی که این مقدار کمتر از leastvalue باشد میفهمیم که یک تطابق بهتر پیدا کرده ایم و مشخصاتی که لازم داریم را ذخیره میکنم.

دلیل اینکه چرا از تطابق مرز ها استفاده میکنیم این است که در مشاهدات تجربه ، نتیجه گرفتم که اختلاف دو زیر آرایه حاوی مقادیر رنگی، گزینه خوبی نیست و تطابق خوبی بدست نمیدهد.(به دلیل اختلاف گاها زیاد در مقادیر دو کانال) ، اما با استفاده از پیدا کردن مرز ها ، چون مرز ها در هر دو آرایه به حد خوبی شبیه به یکدیگر میشوند. میتوان با مچ کردن مرز های دو کانال نتیجه بسیار بهتری بدست آورد.

تابع edgeDetector() :

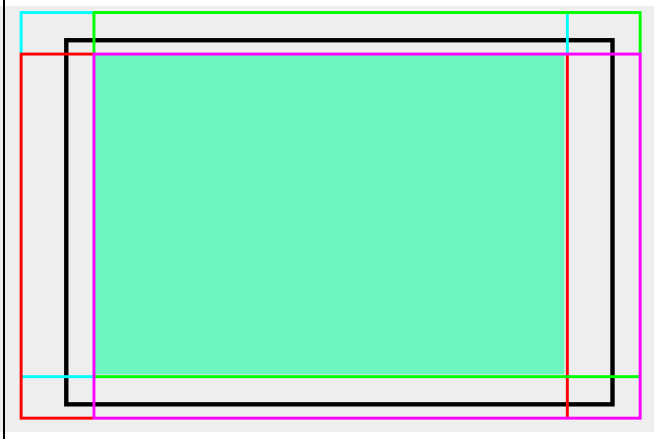
در این تابع ابتدا یک کرنل به صورت [[1] , [-1]] میسازیم. این کرنل برای شناسایی مرز های افقی کاربرد خواهد داشت. سپس با استفاده ز توابع آماده در cv2 ، آرایه ی حاصل از اعمال این کرنل را بدست می آوریم و خروجی میدهیم.(در صورتی که بخواهیم چنین کاری را به صورت دستی برای آرایه A انجام دهیم، کافی است دو زیر آرایه [: , 0:height-1]A و [: , 1:height]A را بدست آوریم و سپس حاصل تفریق آن ها را خروجی دهیم

$$Result = A[1:height\ ,\ :\ ]\ -\ A[\ 0:height-1\ ,\ :\ ]$$

تابع myResize() :

به عنوان ورودی، عکس و مقداری که میخواهیم عکس ما به ان سایز تغییر کند را میگیرید، سپس با استفاده از توابع آماده cv2 ، عکس مورد نظر را ریسایز میکند و خروجی میدهد.

توضیحات تکمیلی :



به شکل روبرو دقت کنید. فرض کنید مستطیل مشکی رنگ، کانال آبی است که آن را ثابت نگه داشتیم. و کانال دوم(سبز یا قرمز) میتواند در محدوده مستطیل های با کادر رنگی جا به جا شود.

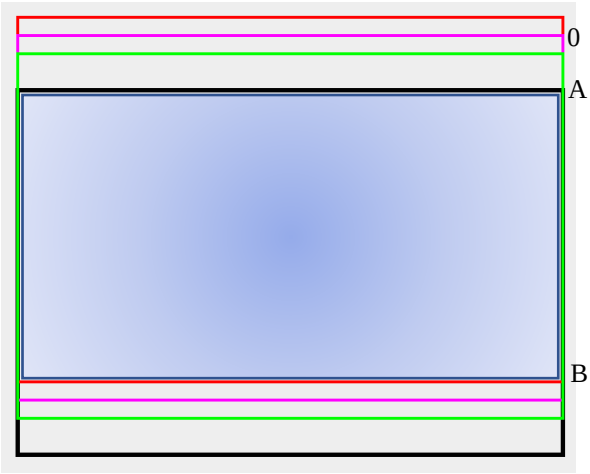
نکته مهم این هست که در تمام این حالت ها ، ناحیه رنگی وسط شکل، در تمام این مستطیل ها مشترک است و این همان ناحیه ای است که ما باید تمام برای تمام حالت ها(تغییرات رنج افقی و عمودی) میزان تطابق را بسنجیم و حالتی که بیشترین تطابق را دارد را پیدا کنیم.

توابع VsamerangerDetector() و تابع HsamerangerDetector() :

با استفاده از این دو تابع، به ترتیب محدوده ی افقی و محدوده ی عمودی کانال channel را که باید مقایسه کنیم بدست می آوریم. این ناحیه با توجه به مقادیر i (میزان جابه جایی افقی) ، j (میزان جا به جایی عمودی) و همچنین xmin , xmax (رنج تغییرات i) و ymin , ymax (رنج تغییرات j) بدست می آوریم. برای مثال محدوده مورد نظر در راستای عمودی (تابع VsamerangerDetector()) طبق حالت های مختلف به این صورت است :

*در هر سه شکل، مستطیل مشکی رنگ ، کل مستطیل کانال آبی(که قرار است ثابت بماند)، مستطیل قرمز نمایانگر حالتی که کانال channel در ابتدای بازه(ymin) باشد، مستطیل سبز نمایانگر حالتی که کانال channel در انتهای بازه(ymax) باشد و مستطیل صورتی بیانگر یک حالت دلخواه بین دو حالت قبلی(ymin تا ymax) است. (ما فقط حالت انتقال عمودی را بررسی میکنیم و انتقال افقی نیز به صورت مشابه خواهد بود)

حالت اول) ymin<0 و ymax<0

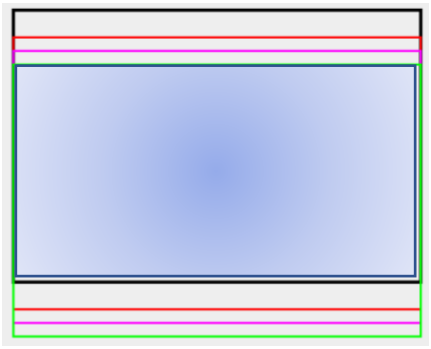


در این حالت ، ناحیه اشتراک تمام مستطیل ها برابر است با مستطیل روبرو است.

با فرض اینکه طول بلاچپ ترین پیکسل مستطیل فعلی ما(صورتی رنگ) در موقعیت 0-j قرار گرفته باشد(0 را به صورت پایه ای برابر موقعیت عمودی بالاچپ ترین پیکسل کانال channel درنظر میگیریم) آنگاه ناحیه ای که باید از مستطیل صورتی رنگ در نظر بگیریم، دارای ارتفاع h – ymin است که در آن h برابر با ارتفاع کانال رنگی ماست . همچنین داریم :

- A = 0-j
- B = h + ymin -j

دو حالت دیگر نیز به مشابه این حالت محاسبه میکنیم :



ymax>0 , ymin>0

ymin<0 , ymax>0

تابع HsamerangerDetector() نیز به طریق مشابه تعریف میشود.

توابع VbaseRange() و HbaseRange() :

همانطور که باید ناحیه اشتراک را برای کانال CHANNEL بدست بیاوریم، باید این ناحیه را برای کانال آبی که ثابت است(و آن را Base مینامیم) نیز پیدا کنیم که این ۲ تابع با عملکردی شبیه به دو تابع قبلی این کار را برای ما انجام میدهند.

کراپ کردن عکس :

نحوه کراپ کردن عکس را برای بالای تصاویر توضیح میدهیم. برای راست و چپ و پایین تصویر به طریق مشابه است.

بعد از انطباق عکس ، کانال r , g, b عکس حاصل را جداگانه استخراج میکنیم. سپس به طور جداگانه روی هر کدام این کار را انجام میدهیم :

- یک متغیر i0 میسازیم.
- تا زمانی که i0 کوچکتر از ۱۰٪ ارتفاع عکس باشد :
  - میانگین ردیفی که در آن هستیم را درنظر میگیریم.
  - اگر این مقدار به طور حدودی نزدیک به سیاه (مثلا کمتر از ۴۰۰۰) و یا به طور حدودی نزدیک به سفید (مثلا بیشتر از ۶۰۰۰۰) بود، این ردیف را در زمره ردیف های حذف شده قرار میدهیم.
- بعد از انجام مرحله بالا ، این حرکت را برای چپ و راست و پایین عکس نیز انجام میدهیم.
- چهار عدد بدست آمده از مراحل قبل(تعداد سطر و ستون های مورد نیاز برای حذف از بالا و پایین و چپ و راست) را خروجی میدهیم.

بعد از اینکه برای هر کانال رنگی این چهار عدد را بدست آوردیم، کاری میکنیم که تصویر نهایی، در اشتراک تمام این تصاویر باشد، یعنی برای مثال برای بالای عکس اگر سه عدد a,b,c برای سه کانال r,g,b بدست آمده باشد ، max(a,b,c) را در نظر میگیریم.

نتایج :

نتایج هر لایه از پیرامید برای عکس T را میتوانید در پوشه ی Result/q3/T مشاهده کنید.عکس اولیه لایه i ام به صورت bestLayer{i}.jpg و نسخه کراپ شده آن عکس با نام layer{i}finalCrop.jpg ذخیره شده است. عکس های لایه 0 عکس هایی با سایز اصلی هستند.

برای تصویر امیر بخارا :

```
********************************************************************
Layer 4
best for R chanel : i =3  and  j=6
best for G chanel : i =2  and  j=3
********************************************************************
Layer 3
best for R chanel : i =5  and  j=13
best for G chanel : i =3  and  j=6
********************************************************************
Layer 2
best for R chanel : i =10  and  j=26
best for G chanel : i =6  and  j=12
********************************************************************
Layer 1
best for R chanel : i =20  and  j=52
best for G chanel : i =12  and  j=25
********************************************************************
Layer 0
best for R chanel : i =41  and  j=106
best for G chanel : i =23  and  j=50
101.21237993240356

Process finished with exit code 0
```

تصویر قطار :


```
********************************************************************
Layer 4
best for R chanel : i =2  and  j=5
best for G chanel : i =0  and  j=3
********************************************************************
Layer 3
best for R chanel : i =4  and  j=10
best for G chanel : i =0  and  j=5
********************************************************************
Layer 2
best for R chanel : i =7  and  j=21
best for G chanel : i =0  and  j=10
********************************************************************
Layer 1
best for R chanel : i =14  and  j=43
best for G chanel : i =0  and  j=20
********************************************************************
Layer 0
best for R chanel : i =28  and  j=84
best for G chanel : i =-2  and  j=40
89.40288782119751
```

تصویر آرامگاه شاه سمرقند :

```
*****************************************************************
Layer 4
best for R chanel : i =0  and  j=6
best for G chanel : i =0  and  j=3
*****************************************************************
Layer 3
best for R chanel : i =0  and  j=14
best for G chanel : i =0  and  j=6
*****************************************************************
Layer 2
best for R chanel : i =0  and  j=30
best for G chanel : i =-1  and  j=13
*****************************************************************
Layer 1
best for R chanel : i =-1  and  j=61
best for G chanel : i =-1  and  j=28
*****************************************************************
Layer 0
best for R chanel : i =-2  and  j=124
best for G chanel : i =-3  and  j=56
87.77485418319702

Process finished with exit code 0
```

سوال چهارم)

در این تمرین، بعد از لود کردن عکس، ابتدا آن را به فضای رنگی HSV میبریم. سپس با کمک تابع range_chooser() ، پیکسل هایی که نیاز به تغییر رنگ دارند را می یابیم.

رنج پیکسل هایی که سرچ میکنیم :

H	S	V
133<=H<=179	S>50	V > 200
0 <= H <=10	S > 150	V > 150
	80 < S < 200	80 < V < 200
	70 < S < 130	170 < V < 210
	135 < S < 215	100 < V < 160

بعد از یافتن تمام پیکسل ها، ابتدا ماتریس temp را میسازیم که در آن تمام پیکسل های پیدا شده، به رنگ زرد تغییر میکنند.(این کار با تغییر مقادیر کانال H آنها به یک مقدار زرد رنگ و بدون تغییر دادن دو کانال S,V انجام میشود.)

بعد از آن ، عکس ورودی را به کمک تابع cv2.blur با یک کرنل به اندازه ۵*۵ بلور میکنیم و در متغیر new_image قرار میدهیم و در نهایت پیکسل های تغغیر رنگ داده شده (که در متغیر temp قرار دارند) را مجددا در ماتریس new_image جایگذاری میکنیم تا حاصل آن یک عکس با پل های زرد و پس زمینه بلور باشد.

نتیجه را با اسم res6.jpg در پوشه ی /Result ذخیره میکنیم.

نکته : به دلیل اینکه مقادیر کانال های رنگی برخی شاخه در رنج هایی که برای پیدا کردن پیکسل های صورتی میگردیم وجود دارند، به طور ناخواسته تعدادی از این پیکسل ها نیز زرد میشوند.(هرچند تا حد امکان سعی شده تعداد چنین پیکسل هایی کم باشد). چنین پیکسل هایی عموما در قسمت پایین چپ عکس قرار دارند.

ما یک عکس دیگر با عنوان res6_enhanced.jpg ذخیره میکنیم که در آن یک رنج خاص از عکس (که به صورت درصدی از طول و عرض عکس هستند) را به حالت قبل از جایگذاری پیکسل های رنگی برمیگردانیم تا خطاهای عکس کمتر شده باشد.

مقایسه :

نتیجه نهایی بهبود یافته

نتیجه نهایی

عکس اصلی

سوال ۵)

بعد از لود کردن عکس، به ترتیب از تابع آماده ، از لوپ و عملیات ماتریسی استفاده میکنیم .

تابع آماده :

از تابع اماده cv2.blur استفاده میکنیم.

لوپ :

به جز سطر و ستون اول و آخر (برای $q=1$ ، اگر q بزرگتر باشد، q سطر و ستون اول و آخر) میتوانیم فیلتر را روی بقیه پیکسل ها اعمال کنیم.(برای پیکسل های در سطر و ستون های بالا ، چون قسمتی از فیلتر خارج از عکس قرار میگیرد نمیتوان فیلتر اعمال کرد.)

بنابراین بر روی تمام پیکسل ها در ناحیه مجاز ذکر شده ، یک باکس فیلتر به ابعاد $(2q+1 , 2q+1)$ در نظر میگیریم. مطابق سوال داریم $q=1$ بنابراین یک مربع ۳*۳ در اطراف آن پیکسل در نظر میگیریم، میانگین تمام این ۹ پیکسل را بدست می آوریم و به عکس جدید اضافه میکنیم.

عملیات ماتریسی :

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

اگر به عملیات بالا دقت کنیم ، میبینیم که برای اعمال یک باکس فیلتر به شکل روبرو :

خانه سورمه ای رنگ بر روی تمام ستون ها به جز دو ستون آخر، و بر روی تمام سطرها به جز دو سطر آخر اعمال میشود. (به طریق مشابه بقیه خانه ها نیز بر به جز ۲ سطر و ۲ ستون، بر روی بقیه سطر و ستون ها اعمال میشوند.)

در نتیجه کافی است ۹ زیر ماتریس از ماتریس اصلی کراپ کنیم. هر کدام را در 1/9 ضرب کنیم و سپس با هم جمع کنیم (یا همه را با هم جمع کرده و سپس تقسیم بر ۹ کنیم.)

در روش سوم، ابعاد ماتریس نهایی ۲ سطر و ۲ ستون از عکس اصلی کمتر دارد.

برای اینکه در هر سه عکس نتیجه یکسان باشد، ماتریس های حاصل از دو روش اول را نیز اندکی کراپ میکنیم تا هر سه هم اندازه شوند.

بعد از اجرای برنامه، در کنسول تعداد خانه با مختصات یکسان از هر سه روش نمایش داده میشود که نشان میدهد نتیجه هر سه روش یکسان است.

```
0.07181167602539062
opencv_convolve finished
62.486921548843384
loop_convolve finished
1.385871410369873
matrix convolve finish
```

نتیجه :

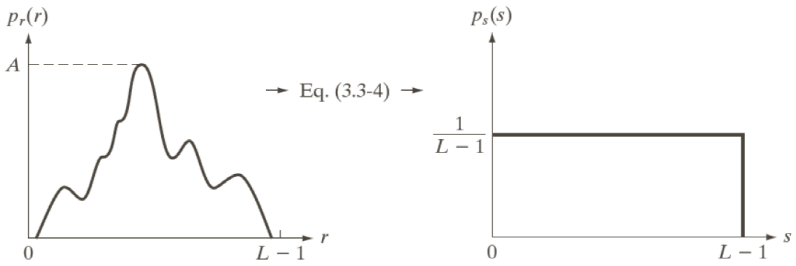
که میتوان نتیجه گرفت، عملیات حلقه بر روی عکس، بسیار فرایند زمانگیری است و بهتر است تمام اعمال را با توابع آماده یا با استفاده از روش های ماتریسی پیاده سازی کنیم.

سوال ششم)

Histogram Equalization

قصد داریم که histogram specification انجام دهیم. یک روش برای انجام این کار این است که ابتدا ، هیستوگرام هر دو عکس را equalize کنیم تا هر دو مشابه شوند، یعنی ابتدا هیستوگرام دو عکس را طبق شکل روبرو تغییر دهیم :

(با فرض اینکه L=256 هست.)



a b

FIGURE 3.18 (a) An arbitrary PDF. (b) Result of applying the transformation in Eq. (3.3-4) to all intensity levels, r. The resulting intensities, s, have a uniform PDF, independently of the form of the PDF of the r's.

بدین ترتیب هیستوگرام هرردو عکس، شبیه به یکدیگر میشود.

فرض کنید هیستوگرام عکس اول و دوم ،به ترتیب تحت تبدیلهای F و G به هیستوگرام یونیفرم تبدیل شده باشند.کاری که ما باید بکنیم، استفاده از تابعی مانند $y = G^{-1}(F(x))$ است. یعنی میخواهیم ببینیم که اینتنسیتی x در عکس اول، باید به چه مقدار y در عکس دوم مپ شود. از انجا که میدانیم $G(y) = F(x)$ میتوان نتیجه گرفت که : $y = G^{-1}(F(x))$

اما در عمل ممکن است با دو مشکل مواجه شویم:

۱. ممکن است تابع G یک به یک و در نتیجه وارون پذیر دقیق نباشد. در عمل برای رفع چنین مشکلی تمام مقادیر x را به کوچکترین مقدار y میبریم که $G(y) = F(x)$.
۲. اما اشکال دوم که میتواند پیچیده تر باشد، این است که ممکن است هیچ y پیدا نشود که $G(y) = F(x)$. در این صورت ما باید نزدیک ترین مقدار به y مانند y' را پیدا کنیم که $G(y') = F(x)$ و انجام چنین کاری میتواند پیچیده باشد.

اما برای حل این سوال، از یک روش تقریبی دیگر استفاده میکنیم:

توابع توزیع تجمعی عکس های اول و دوم را بدست می آوریم و آنها را به ترتیب C1 و C2 مینامیم. حال کافی است به ازای هر اینتنسیتی x در عکس اول، کوچکترین اینتنسیتی مانند y در عکس دوم پیدا کنیم به صورتی که :

$$min \;\; y : C2(y) \geq C1(x) \quad **$$

و این کار دقیقا همان اشکال دوم (پیدا کردن بهترین و نزدیک ترین مقدار ممکن) را برای ما انجام خواهد داد.(یک دلیل دیگر برای اینکار این است که این تابع مپ کردن، باید حالتی صعودی داشته باشد یعنی

$$w < r \rightarrow G^{-1}(F(w)) \leq G^{-1}(F(r))$$

بنابراین در این اول روش مینیمم y را پیدا میکنیم که خاصیت ** را داشته باشد.)

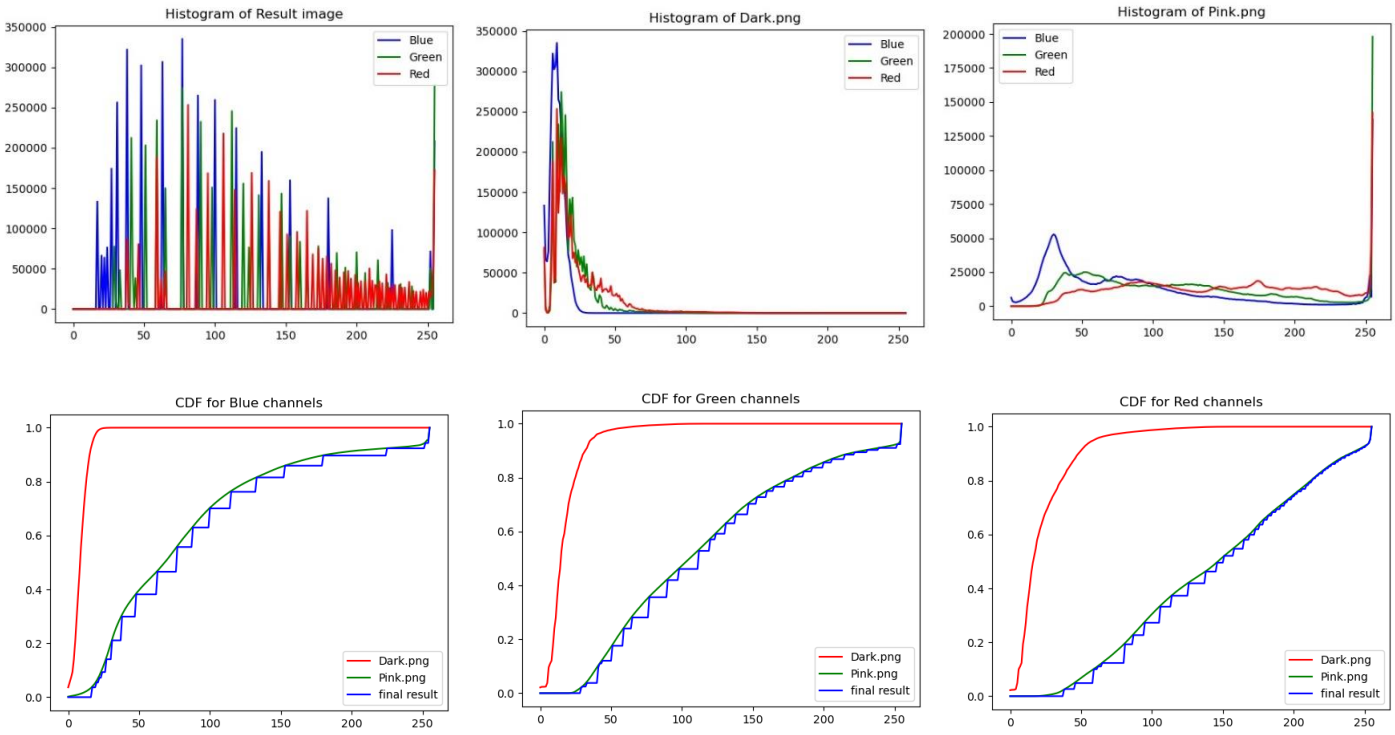
روش کار :

ابتدا هر دو عکس را لود میکنیم. سپس توابع هیستوگرام و توزیع تجمعی آنها را بدست می آوریم.(از توابع آماده cv2 استفاده میکنیم.)

بعد با استفاده از تابع match() اقدام به مچ کردن دو عکس میکنیم.(این کار را به صورت مجزا بر روی هر سه کانال رنگی rgb انجام میدهیم.)

در این تابع ، به ازای هر x در عکس اول ، نزدیک ترین y در عکس دوم که در شرایط بالا (روش تقریبی) صدق کند را پیدا میکنیم. بعد از اتمام این مرحله نیز ، پیکسل های عکس جدید را با استفاده از مقادیر y پیدا شده به جای x ، میسازیم و خروجی میدهیم.

کلیه نتایج اعم از هیستوگرام های هر سه کانال رنگی برای هر دو عکس، هیستوگرام های تجمعی هر عکس، cdf های تجمعی هر سه عکس و ... را میتوانید در فولدر مگا یا در پوشه ی Result/q6/ (بعد از اجرای کد) مشاهده کنید.



عکس نهایی :