**بسمه تعالی**

**دانشگاه صنعتی شریف**

**دانشکده علوم کامپیوتر**

**Image processing – دکتر مصطفی کمالی**

**محمد حسین مسلمی – 97102463**

**HW1 report**

## سوال یک – Enhancement :

برای این سوال از ترکیب دو روش استفاده شده ، در ابتدا تصویر را با استفاده log transformation بهبود بخشیده ایم ؛ به اینصورت که نقاط تاریک تصویر را روشن کرده ایم ، زیرا همانطور که در درس آمده بود این تبدیل میتواند فاصله intensity را برای نقاط تاریک بیشتر کند ، همچنین پارامتر $\alpha$ برای این تبدیل به صورت تجربی و با استفاده از امتحان کردن مقادیر مختلف و برابر با 0.01 انتخاب شده است ( همچنین انتخاب تبدیل log نیز به صورت تجربی انتخاب شده ، همچنین نحوه اعمال این تبدیل در سوال یک و دو کاملا مانند هم است که در سوال دو توضیح داده شده است). سپس در مرحله بعدی برای بدست آمدن نتیجه بهتر اقدام به افزایش contrast عکس شده است به اینصورت که چون عکس رنگی است ابتدا آنرا به hsv تبدیل کرده و کانال سوم آن یعنی value که نشان‌دهنده brightness است را در نظر گرفتیم و hist آنرا بدست آوردیم ، حال برای افزایش contrast کافیست عمل histogram normalization را انجام دهیم ، به اینصورت که $H_{target}$ برابر با یک تابع uniform به اندازه $\frac{1}{255}$ است ، لذا مقدار cumulative آن برابر با یک خط با شیب $\frac{1}{255}$ است ، لذا برای این‌که هیستوگرام تصویر را نرمال کنیم کافیست که اگر مقدار یک پیکسل برابر با r باشد این مقدار را برابر با $255*H_1(r)$ قرار دهیم ، که در اینجا $H_1$ تابع cumulative برای تصویر اصلی است که از hist آن بدست آمده و ضرب در 255 در واقع همان $H^{-1}_{target}$ است ، زیرا $H_{target}$ برابر یک خط باشیب عکس 255 بود. پس از انجام این تغییرات بر روی کانال value دوباره عکس را به صورت BGR میکنیم و ذخیره میکنیم .

## سوال دو – Enhancement :

این سوال نیز شبیه سوال قبل است با این تفاوت که پارامتر تبدیل log برابر با 0.075 انتخاب شده است ، برای اعمال این تبدیل عکس رنگی که سه کانال دارد را ؛ کانال هایش را از هم جدا میکنیم و این تبدیل را بر روی هر کانال به صورت جداگانه اعمال میکنیم و در نهایت دوباره این سه کانال را در کنار هم قرار میدهیم . در ابتدا نوع ماتریس کانال را از uint8

به float64 میبریم که محاسبات را بتوان به درستی انجام داد ، سپس برای مثال بر روی کانال B به اینصورت تبدیل به دست آمده :

```
img_b2 = np.uint8(np.log10(a * img_b + 1) * (255 / np.log10(1 + 255 * a)))
```

که در آخر نیز برای اینکه بتوان عکس را نمایش داد دوباره ماتریس را به نوع uint8 تبدیل کرده ایم .

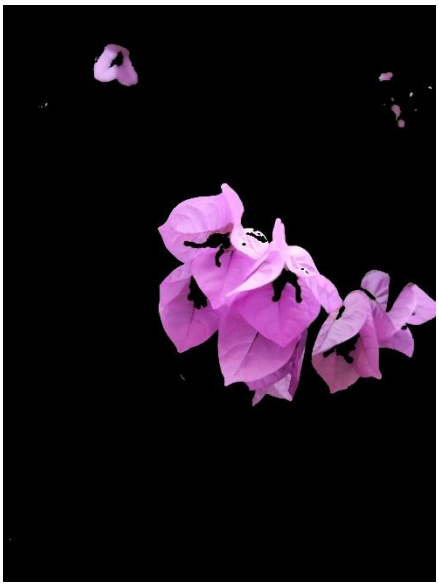
**سوال چهار – Color Processing and Blurring :**

برای این سوال دو mask میسازیم ، mask1,mask2 که mask1 برابر با نواحی مطلوب است و mask2 برابر نواحی نامطلوبی است که توسط mask1 به اشتباه درآمده ، حال برای mask نهایی از mask2 یک not گرفته و درایه های متناظر را با mask1 ، and میکنیم که حاصل mask نهایی ما میشود.

برای بدست آمدن این maskها ابتدا عکس را به hsv برده و سپس با آزمون و خطا دو کران بالا و پایین برای hsv1 , hsv2 بدست میآوریم ، سپس با استفاده از این دو کران و تابع cv2.inRange ، mask خود را میسازیم ، خروجی تابع cv2.inrange یک ماتریس است که از دو مقدار 0 و 255 تشکیل شده پس ما خروجی را تقسیم بر 255 میکنیم تا mask ما حالت باینری 0و1 پیدا کند.

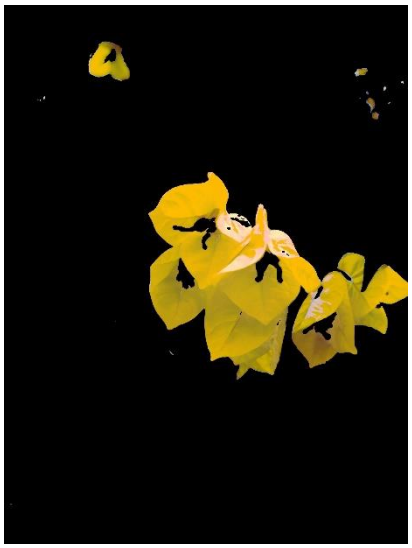
در نهایت هم این mask بدست آمده با با یک فیلتر میانگین گیر با اندازه 5 میانگیری میکنیم که نتیجه بهتری میدهد.

در مرحله بعد مقدار این mask را در سه کانال عکس ضرب میکنیم که فقط قسمت های صورتی انتخاب شوند ، ( متغیر img2 ) دو مقدار mask وimg2 در زیر آمده اند :



در قدم بعدی از عکس اصلی یک ورژن بلور شده با یک avg_filter با اندازه 21 میسازیم به نام img_blur سپس این نسخه بلور شده از عکس را ضربدر 1-mask میکنیم که حاصل یعنی تمام قسمت هایی از عکس که رنگ غیر صورتی داشته اند بلور شده اند و قسمت هایی که صورتی بوده اند فعلا خالی هستند یعنی: ( عکس زیر برابر img_blur است)

حال در آخرین مرحله img2 که همان قسمت های صورتی عکس بود را میخواهیم زرد کنیم به اینصورت که ابتدا آنرا به hsv تبدیل کرده و سپس از کانال hue مقدار 80 واحد کم میکنیم که باعث میشود صورتی تبدیل به زرد شود :



حال این عکس که همان img2 است را با img_blur جمع میکنیم که نتیجه دلخواه بدست میآید و ذخیره میکنیم .

**سوال پنج – Convolution :** *( همه قسمت ها را با* mode = ‘valid’ *انجام میدهیم )*

در این سوال به سه روش می‌خواهیم فیلتر میانگین گیر با اندازه 3 را عکس اعمال کنیم ، روش اول استفاده از توابع آماده است که ابتدا یک kernel 3 در 3 با درایه های 1 میسازیم و سپس این kernel را با تابع scipy.signal.convolve2d بر هر کانال عکس اعمال میکنیم و سپس حاصل را تقسیم بر 9 میکنیم تا درنهایت برابر با اعمال یک فیلتر 3 در 3 با درایه های 1/9 یا همان میانگین شود ، علت این کار اینست که اگر مستقیما با فیتر 3 در 3 با درایه های 1/9 بکنیم حاصل به دلیل نوع پیاده سازی کتابخانه اندکی با خروجی دو قسمت بعد تفاوت میکند ولی با کانوالو کردن با ماتریس با درایه های یک و سپس تقسیم بر 9 حاصل با قسمت های دیگر دقیقا برابر میشود .

برای روش دوم با استفاده از دو حلقه تو در تو میخواهیم میانگین گیر را پیاده کنیم ، به اینصورت که i , j را از 0 تا h-3 و w-3 تغییر میدهیم و هربار یک پنجره 3 در 3 از عکس در img [i + 1, j + 1] در نظر میگیریم و میانگین مقادیر این پنجره را برابر مقدار پیکسل عکس حاصل در نقطه result[i + 1, j + 1] قرار میدهیم که حاصل برابر اعمال فیلتر میانگین گیر بر تصویر میشود .

در روش سوم ، ابتدا یک تابع shift(image,tx,ty) میسازیم که این تابع با استفاده از ماتریس translation که تدریس شد عکس را به اندازه tx به راست و به اندازه ty به پایین حرکت میدهد و جاهای خالی را صفر میگذارد ، یعنی اگر یک عکس 100×100 را به اندازه 10 واحد به پایین شیفت دهیم با این تابع ، حاصل برابر یک ماتریس 100×100 میشود که 10 سطر بالای آن صفر است و 90 سطر زیر آن را از 90 سطر اول عکس اصلی پر میکند ، یعنی بخشی از پایین عکس از بین میرود ، چون میخواهیم در mode=’valid’ حاصل فیلتر را حساب کنیم مشکلی رخ نمیدهد. حال برای بدست آوردن مقدار یک پیسکل در تصویر بلور شده کافیست مقدار عکس اصلی در آن پیسکل را با 8 شیفت یافته عکس یعنی : {بالا ، پایین ، راست ، چپ ، بالا-راست ، بالا-چپ ، پایین-راست ، پایین-چپ} جمع کرده و تقسیم بر 9 کنیم و سپس فقط بخشی از تصویر حاصل که مورد نیاز است را در نظر میگیریم یعنی از هر طرف یک سطر یا ستون را در نظر نمیگیرم.

حاصل هر سه این روش دقیقا مثل هم است و اگر از هم منها کنیم و abs حاصل را جمع کنیم صفر میشود. همچنین زمان اجرای این سه روش به صورت زیر است :

| Method 1 | Method 2 | Method 3 |
|---|---|---|
| 0.7265 s | 75.119 s | 0.513 s |

## سوال شش – Histogram Specification :

در این سوال نیز به شیوه سوال یک عمل میکنیم ولی در اینجا $H_2$ یک تابع خطی با شیب ثابت 255 نیست بلکه تابعی است که از هیستوگرام عکس target بدست آمده . برای اینکار هر دو عکس اصلی و target را به hsv تبدیل کرده و سپس کانال value یا همان brightness آنها را در نظر گرفته و مقادیر hist و از آنها مقدار cumulative آن یعنی $H_1$ ,$H_2$ را در نظر میگیریم .سپس اگر مقدار یک پیکسل برابر با r باشد آنگاه مقدار آنرا برابر با $H^{-1}{}_2(H_1(r))$ قرار میدهیم ، با این کار هیستوگرام تصویر اصلی را به هیستوگرام تصویر هدف تبدیل میکنیم ، همچنین باید دقت شود که عکس ها رنگی هستند ، لذا هیستوگرام هر کانال از عکس source را باید با هیستوگرام از همان کانال از تصویر target لحاظ کرد. با استفاده از np.histogram و np.cumsum مقدار cumulative از هستوگرام را در تابع hist_spec حساب میکنیم ، سپس با دوبار استفاده از تابع np.interp مقدار $H^{-1}{}_2(H_1(r))$ را بدست میآوریم ، که بار اول مقدار $H_1(r)$ و بار دوم با جابجا کردن مقدار x,y در تابع np.interp عملا inverse را حساب کرده و $H^{-1}{}_2$ محاسبه میشود. در آخر نیز نمودار هیستوگرام را برای هر سه کانال عکس بدست آمده بدست میآوریم و در کنار نمودار hist اولیه جهت مقایسه رسم میکنیم.

## سوال سه :

در این سوال سه تابع وجود دارد که با توضیح آنها عملا نحوه انجام سوال را توضیح داده ایم ، تابع اصلی main است که با گرفتم اسم فایل ورودی و اسم فایل خروجی کارهای لازم را انجام داده و خروجی را ذخیره میکند ، تابع find_crop یک عکس میگیرد که دور آن کادرهای سیاه و یا رنگی ایجاد شده است و این کادرها را crop میکند ، تابع سوم find_shift دو وردی به نام in1,in2 دارد که in1 را ثابت در نظر گرفته و مقدار شیفت مورد نیاز که in2 بر in1 منطبق شود را در خروجی میدهد. حال به صورت دقیق تر این توابع را بررسی میکنیم .

***main -1*** : با گرفتن نام ورودی ابتدا ورودی را به صورت cv2.IMREAD_UNCHANGED باز میکند ( چون تصاویر 16 بیتی هستند) سپس طول و عرض حاصل را در نظر گرفته و از طول سه قسمت میکنیم که سه عکس مربوطه به سه کانال بدست آید ، عکس بالا im1 ، وسط im2 و پایین im3 مینامیم . سپس im2 را ثابت در نظر گرفته و با استفاده از تابع find_shift مقدار جابجایی لازم برای im1,im3 که منطبق بر im2 شوند را پیدا میکنیم و سپس im1,im3 را با توجه به مقداری که find_shift بدست میدهد شیفت میدهیم تا بر im2 منطبق شوند ، حال اگر همینجا عکس ها را روی هم بگذاریم خروجی با این که بر هم منطبق شده یکسری کادر رنگی دور آن ایجاد شده که بدلیل وجود یکسری کادر سیاه یا سفید در کناره های هر کدام از im1,im2,im3 است .برای حل این مشکل ابتدا im1,im3 را که شیفت داده

بودیم از قبل ، حال از تابع find_crop(image) استفاده میکنیم. خروجی این تابع به صورت [up, down, left, right] است که یعنی مقداری که از هر چهار طرف عکس باید crop شود که این مقادیر برای im1,im2,im3 متفاوت است ، اما برای اینکه باید پس از crop شدن انداز هرسه برابر شود ، ما مقدار بیشینه برای میزان crop شدن از هر طرف را بدست میآوریم ، یعنی اگر مقدار مورد نیاز برای crop کردن از بالای عکس im1,im2,im3 برابر با $up_1,up_2,up_3$ باشد ، بین این سه مقدار بیشینه را در نظر گرفته و از هر سه im1,im2,im3 از سمت بالا به تعداد این بیشینه از سطر ها را crop میکنیم. حال اگر این کار را برای هر چهار طرف انجام دهیم این کادرهای سیاه از بین رفته و خروجی نهایی دیگر کادر رنگی در کناره هایش ندارد ، در نهایت این سه کانال را روی هر قرار داده و خروجی را از 16 بیتی به 8 بیتی تبدیل میکنیم و به صورت jpg ذخیره میکنیم.

2- *find_shift(in1, in2)* : این تابع ورودی in1 را ثابت فرض کرده و مقدار شیفت مورد نیاز برای in2 که بر in1 منطبق شود را خروجی میدهد. برای این کار در قدم اول فقط 15 درصد از ناحیه وسط in1,in2 را درنظر میگیرد تا دقت وسرعت بهینه شود. در بتدا عکس ها را به scale 0.1 میکنیم و سپس tx,ty را از 10- تا 9 حرکت میدهیم (در کل 400 مقدار شیفت مختلف را بررسی کرده ایم ) که متناظر با 100- تا 90 در scale اولیه است ، سپس به ازای هر tx,ty که in2 اسکیل شده را جابجا کردیم فاصله $l_1$ دو ماتریس را حساب کرده و به صورت [l1 , (x,y)] ، در انتهای ارایه distances میگذرایم ، در نهایت به ازای همه این tx,ty ها در این بازه ای که گفته شد مقدار l1 را حساب کردیم و در distances ریخته ایم و لذا کوچترین مقدار l1 را از بین این مقادیر پیدا کرده و مقدار (x,y) آن را به عنوان شیفتی که دو عکس در scale 0.1 بهترین انطباق را بر هم دارند گزارش میکنیم ، حال این مقدار را ضربدر 5 کرده و در متغیر shift میریزیم ، علت اینکه ضربدر 5 کردیم این بود که در مرحله اول در scale 0.1 محاسبات را انجام دادیم ، در مرحله بعدی میخواهیم در scale 0.5 بهترین جواب را پیدا کنیم ، پس 1 واحد شیفت در scale=0.1 برابر 5 واحد شیفت در scale=0.5 است . این مقدار شیفت انتخاب شده را $T_{1x}$ , $T_{1y}$ مینامیم.

در scale=0.5 بازه شیفت هایی که بررسی میشود برابر $T_{x1} \pm 2$ و $T_{y1} \pm 2$ میباشند دوباره مثل قبل بهترین جواب را از فاصله $l_1$ پیدا میکنیم و این بار برای اینکه به scale=1 ببریم باید ضربدر دو کنیم که میشود $T_{2x}$ , $T_{2y}$ ( در این قسمت 25 شیفت مختلف حول جواب قسمت قبل بررسی میشود ) سپس در آخر و در scale=1 با ازای $T_{x2} \pm 1$ و $T_{y2} \pm 1$ مقادیر $l_1$ را بررسی میکنیم ( یعنی 9 مقدار مختلف در کل ) و بهترین جواب را به عنوان S1=( $T_{3x}$ , $T_{3y}$) خروجی میدهیم که مقدار شیفت مورد نیاز برای in2 است که بر روی in1 بیفتد.

3- تابع find_crop فقط یک عکس در ورودی میگیرد و چهار مقدار در خروجی میدهد که نشان میدهد از هر طرف عکس چقدر باید crop شود تا دیگر کادر سفید یا سیاه نداشته باشد . این تابع در ابتدا میانگین مقادیر 100 سطر وسط

عکس ورودی را حساب میکند ( فرقی نمیکند سطر یا ستون این عدد تقریبا در هر دو حالت یکی میشود ) و آنرا avg مینامیم، سپس از اولین ردیف از هر چهار طرف عکس شروع کرده و به جلو میرویم ( تا حداکثر 300 پیکسل ) ؛ هر ردیفی که میانگین آن تقسیم بر avg کمتر از 0.6 و یا بیشتر از 1.4 باشد به عنوان ردیف نا مطلوب در نظر گرفته میشود ، در نهایت آخرین مقدار ردیف های نامطلوب از هر طرف را در متغیر مربوطه ذخیره کرده و خروجی میدهیم

به طور میانگین پردازش هر عکس حدود 10 ثانیه یا اندکی بیشتر طول میکشد همچنین این کد بدون هیچ گونه نیاز به تغییر در هیچ پارامتری برای هر عکس از این نوع Prokudin-Gorskii Images کار میکند و هیچ نیازی به fine-tune کردن برای عکس جدید ندارد و صرفا باید نام فایل آنرا و نامی که میخواهیم تحت آن ذخیره شود را به main بدهیم .