به نام خدا

امیرمحمد کمیجانی ۹۹۵۲۲۰۳۲

گزارش کُد تمرین سری دوم بینایی کامپیوتر

**** سوالات : ۱ الف ، ۲ الف ، ۵ و ۶ که تشریحی بودند در فایل دیگری همراه با این فایل هستند**

۱)

توضیح کد :

تابع **calc_hist** با استفاده از کتابخانه opencv و تابع calcHist به دست آمده است.

تابع stretch hist مطابق با فرمول موجود در جزوه و بدون استفاده از کتابخانه ای و با استفاده از حلقه ها به دست آمده است.

تابع clip_hist ، این تابع را نیز با استفاده از فرمول داخل جزوه پیاده سازی کردیم.همچنین اگر مقدار f(x,y) کمتر از min_value باشد مقدار خروجی در آن پیکسل از تصویر جدید را برابر صفر قرار میدهیم. اگر مقدار f(x,y) – minVal / maxVal – minVal بزرگتر از ۱ شد چون بعد از آن در ۲۵۵ ضرب میکنیم مقدار خروجی بیشتر از ۲۵۵ میشود که قابل قبول نیست در نتیجه آنرا به ۲۵۵ مَپ میکنیم.

$$g(x,y) = clip[f(x,y)] = \left(\frac{f(x,y) - f_1}{f_{99} - f_1}\right)(MAX - MIN) + MIN$$

$$g(x,y) = stretch[f(x,y)] = \left(\frac{f(x,y) - f_{min}}{f_{max} - f_{min}}\right)(MAX - MIN) + MIN$$

قسمت ج)

بعد از انجام کشش هیستوگرام تغییر خاصی در تصویر مشاهده نمیکنیم به این دلیل است که بعضی نقاط وجود دارند که مقادیر بسیار پایین یا بسیار بالایی دارند و در فرآیند کشش هیستوگرام تغییر خاصی نمیکنند(یا خیلی روشنند یا خیلی تاریک) که این مورد باعث میشود هیستوگرام ما دچار کشش خاصی نشود و فقط در جایی هیستوگرام و مقادیر زیادی داریم از نقطه ای به نقطه ای دیگر مَپ شوند. و همان تراکم باقی میماند که مطلوب ما نیست

در روش برش هیستوگرام ، بخشی از دیتا که مطلوب ما نیستند را حذف میکنیم و محاسبات را در قسمت مطلوب انجام میدهیم. که این روش باعث میشود هیستوگرام توزیع بهتری داشته باشد در نتیجه کنتراست تصویر افزایش میابد و کیفیت تصویر همانگونه که مشخص است بهتر میشود.

مقادیر minVal = 70 و maxVal = 180 را در نظر گرفتم که با دیدن هیستوگرام تصویر و البته کمی آزمون و خطا این مقادیر را قرار دادم.

۲)

توضیح کد و قسمت ب) تحلیل نتایج : * در این سوال فقط از نامپای استفاده کردم.

با استفاده از کتابخانه نامپای و تابع هیستوگرام آن هیستوگرام را در بخش اول بدست آوردیم

سپس در تابع calc_cdf ابتدا هیستوگرام تصویر را به دست آوردیم سپس با استفاده از تابع cumsum جمع تجمعی این هیستوگرام را محاسبه کردیم و در قسمت آخر normalize کردیم.

در قسمت تطبیق هیستوگرام ابتدا cdf مربوط به تصویر source , reference را به دست آوردیم سپس با استفاده از تابع interp نقاطی که شرط cdf1(x1) = cdf2(x2) میشد را بررسی کردیم و در انتها خروجی نهایی را بر اساس شدت روشنایی های جدید محاسبه کردیم.

در خروجی این سوال ، جزئیات تصویر را باید حفظ میکردیم ولی شدت روشنایی را باید از تصویر reference میگرفتیم که همانگونه هم شد.

در تصویر نمودار ها، نمودار output , ref بسیار منطبق بر هم هستند که یعنی خواسته ما برآورده شده است و ما شدت روشنایی های تصویرref را به تصویر خروجی منتقل کرده ایم.

۳)

الف)

ابتدا با استفاده از تابع equalizeHist عملیات متعادل سازی هیستوگرام را انجام دادیم.
مشکل این روش در این است که به نقاط محتلف تصویر کاری ندارند و یک فرمول را به کل تصویر اعمال میکنند. در مورد تصویری که در این سوال است به طور مثال باعث شده است که بعضی از جزئیات مانند نوشته های بورد پشت سر مجسمه بهتر شوند اما باعث شده است جزئیات مجسمه بدتر شوند. علت این است که برخی پیکسل ها بسیار تاریکند و برخی بسیار روشن.به طور کلی این روش در همه جا خوب جواب نمیدهد.

ب)

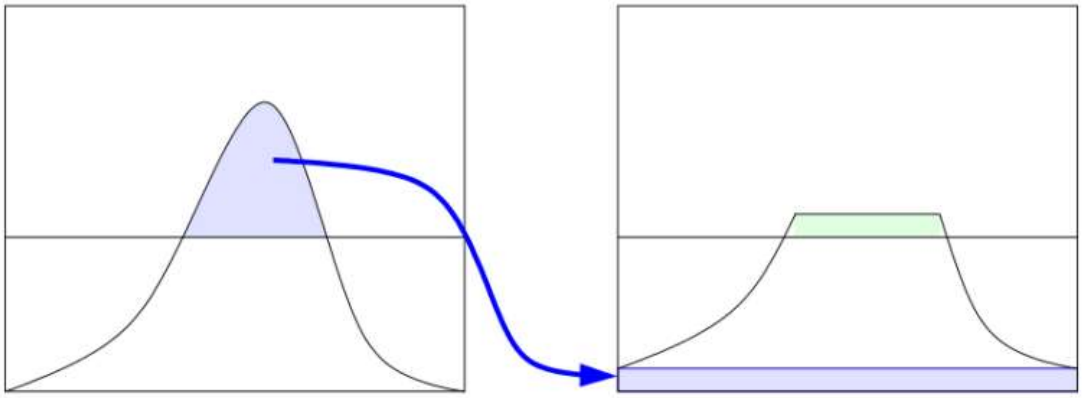
روش **ACE1** به این صورت که تعدادی grid برای تصویر در نظر میگیریم و هر grid را به طور جدا ارتقا میدهیم.در اینجا عملیات ارتقا را همان متعادل سازی هیستوگرام در نظر گرفتیم.
عیب این روش زمانی مشهود است که دو پیکسل نزدیک به هم مثلا در مجسمه داریم اما هر کدام در دو گرید متفاوت هستند(دو پیکسل مرزی مثلا).هنگامی که متعادل سازی هیستوگرام را اعمال میکنیم یکی از گرید ها مانند کلیت مجسمه روشن میماند و دیگری مانند اطراف مجسمه تاریک میشود. در نتیجه تصویر خروجی ما به صورت شطرنجی میشود.

روش **ACE2** ، به این صورت است که برای هر پیکسل یک پنجره همسایگی در نظر میگیریم و بر اساس همسایه هایش تابع تبدیل را محاسبه میکنیم.
در کد نیز ابتدا یک padding در نظر گرفتیم برای نقاط گوشه و مرزی(یعنی گریدی که به آن

میدهیم از سایز اصلی عکس خارج باشد برای برخی نقاط) و سپس با استفاده از متعادل سازی هیستوگرام ، ارتقا هر نقطه را بدست آوردیم. تصویر خروجی از حالت قبل بهتر است اما نویز در آن تقویت شده است و برخی جزئیات تصویر دارای نویز بیشتری شده اند و تشخیص آنها دشوار تر شده است.

از معایب این دو روش میتوان به تقویت نویز اشاره کرد.

روش **CLAHE** برای این است که میزان نویز را محدود کند.برای اینکار باید میزان کنتراست تصویر را محدود کنیم.
در کد این روش هم ابتدا padding در نظر میگیریم سپس هیستوگرام را محاسبه میکنیم سپس بر اساس محدودیتی که به عنوان پارامتر معین شده از هیستوگرام کم میکنیم و البته به انتهای آن اضافه میکنیم. و در نهایت با محاسبه cdf عملیات متعادل سازی را انجام میدهیم.
خروجی این روش به نسبت دو روش قبلی بهتر است و نویز کاهش پیدا کرده البته کمی تصویر تیره تر شده است و کنتراست آن کاهش پیدا کرده است که باعث میشود نوشته های پشت مجسمه دیدنشان سخت تر شود.



قسمت ج)

در قسمت کد با استفاده از کتابخانه clahe را پیاده سازی کردیم.
برای بررسی این قسمت با مقادیر مختلف گرید و محدودیت نتایج متفاوتی داریم.

هنگامی که ابعاد پنجره را بزرگ میگیریم مشخصا تابع تبدیل بر اساس همسایگی تعداد زیادی از پیکسل ها مشخص میشود و با توجه به اینکه این تصویر در برخی نقاط بسیار روشن و در برخی بسیار تاریک است نتیجه خوبی به همراه نخواهد داشت. حال اگر محدودیت را زیاد بگیریم مثلا ۱۲۸ نسبت به حالت محدودیت=۲ کنتراست بسیار کمتر میشود و تصویر در حالت بدتر میشود.

در حالت آخر کماکان نویز به صورت زیادی وجود دارد اما اندازه پنجره مناسب است و کلیت تصویر همسایگی خوبی را محاسبه کرده اما به دلیل threshold زیاد باز هم نویز زیادی داریم.

در حالت سوم که بهترین حالت است اندازه پنجره مناسب است و مقدار محدودیت نیز خوب است در نتیجه نویز کاهش پیدا کرده است و در برخی نقاط جزئیات بهتر قابل مشاهده هستند.

۴)

برای اضافه کردن نویز به صورت دستی ، مقدار رندومی را برای نویز نمک و مقدار رندومی را برای نویز فلفل انتخاب میکنیم. سپس پیکسل های رندومی از تصویر ورودی را انتخاب میکنیم و بر اساس نویز نمک و فلفل تصویر را سفید و سیاه میکنیم.

تابع reflect101 را با استفاده از کتابخانه نامپای و تابع pad و mode = reflect عملیات پدینگ انجام میشود. نکته در رابطه با این تابع اینست که ما از mode = reflect استفاده کردیم اما عملا عملیات reflect101 روی آن صورت میگیرد.برای بررسی صحت این مورد آزمون و خطا انجام داد و همچنین داخل داکیومنت خود نامپای نیز این مورد قابل مشاهده است.


```
>>> a = [1, 2, 3, 4, 5]
>>> np.pad(a, (2, 3), 'reflect')
array([3, 2, 1, 2, 3, 4, 5, 4, 3, 2])
```

که مشخص است منطبق هم هستند. اما در یک سِل که آنرا کامنت کردم پدینگ را کامل بررسی کردم و در تصویر ما پدینگ مانند reflect101 نیز انجام شده بود.

در تابع averaging ، ابتدا پدینگ را مطابق قسمت قبل انجام میدهیم سپس عملیات جمع را بر روی همسایگی به اندازه فیلتر انجام میدهیم و سپس در نهایت میانگین میگیریم.

در تابع median ، پس از انجام عملیات ها مقادیر را داخل یک لیست میریزیم سپس سورت میکنیم و در نهایت میانه این لیست را محاسبه میکنیم.

در تابع gaussian ، ابتدا کرنل را مشخص میکنیم و مقدار صفر میدهیم سپس عملیات مربوط به تابع گوسی را برای کرنل انجام میدهیم.

$$G(s,t) = Ke^{-\frac{s^2+t^2}{2\sigma^2}} = Ke^{-\frac{r^2}{2\sigma^2}}$$

این فرمول را پیاده سازی کرده ایم و سپس مقدار total را محاسبه میکنیم و در انتها نرمالایز میکنیم.

در قسمت آخر ، برای هر یک از فیلتر های خواسته شده از توابع موجود در کتابخانه opencv با همان اندازه ها استفاده کرده ایم.

بعد از بررسی نتایج متوجه میشویم که خروجی برای زمانی که خودمان پیاده سازی کردیم و در موردی که از کتابخانه استفاده کرده ایم یکسان است.

تاثیر اندازه کرنل در این است که با افزایش آن تصویر smoothing بیشتری را دارد در نتیجه جزئیات بیشتری از تصویر حذف میشود و همچنین نویز آن نیز کاهش میابد.