تمرین *color*

آریا ابراهیمی ۹۸۲۲۷۶۲۱۷۵

چکیده

هدف از انجام این تمرین،آشنایی با فضای های رنگی و آشنایی با پردازش تصاویر رنگی است. در بخش اول به معرفی چند فضای رنگی می‌پردازیم. در این میان روی فضای رنگی *HSI* بیشتر عمیق میشویم و آنرا به صورت جداگانه بررسی میکنیم.

در بخش دوم، به پردازش تصاویر رنگی *RGB* می‌پردازیم. این بخش مربوط به گسسته سازی این تصاویر می‌باشد. برای دو تمرین اول، تابعی *general* برای محاسبه و گسسته سازی مقادیر *graylevel* طراحی شده است که همانند تابعی است که در تمرین اول استفاده شده است با این تفاوت که برای هر سه کانال *RGB* این کار را انجام میدهد. برای تمرین سوم خواسته شده است تا رنگ های تصویر را کاهش دهیم و گسسته سازی را درواقع با کاهش رنگ انجام دهیم. برای حل این سوال با توجه به بررسی هایی که انجام دادم، یک روش استفاده از روش های یادگیری بدون ناظر و استفاده از الگوریتم های خوشه بندی است که برای حل این تمرین از خوشه بندی *K-Means* استفاده شده است.

۱- تحلیل تکنیکال

۱.۱.۱- در این تمرین از ما خوسته شده است تا عکس *Lena* که در فضای رنگی *RGB* است را به فضای رنگی *HSI* تبدیل کرده و مقادیر *Hue* و *Saturation* و *Intensity* را به صورت جداگانه نمایش دهیم. برای این کار، از فرمول های ارائه شده در کتاب برای تبدیل *RGB* به *HSI* استفاده میکنیم.

$$\theta = cos^{-1}\left\{\frac{\frac{1}{2}[(R-G)+(R-B)]}{[(R-G)^2+(R-B)(G-B)]^{\frac{1}{2}}}\right\}$$

$$H = \begin{cases}\theta & if B \le G \\ 360-\theta & if B > G\end{cases}$$

با استفاده از فرمول های بالا میتوان مقدار *Hue* را محاسبه کرد. برای پیاده سازی این بخش در کد ولی باید نکاتی رعایت شود. نکته اول این است که مخرج کسر صفر نشود. زمانی که مقادیر در همه چنل های برابر باشند مخرج کسر صفر میشود. برای همین در پیاده سازی مخرج با عددی کوچک جمع کرده میشود که از صفر شدن جلوگیری شود. تابع مربوط به محاسبه cos^{-1} در نامپای، مقادیری که حقیقی یا بینهایت نیستند را به صوت *NaN* برمیگرداند و با این کار از این اتفاق جلوگیری میکنیم.

برای محاسبه *Saturation* و *Intensity* هم از فرمول های زیر استفاده میشود و پیاده سازی آنها دشواری ندارد و به راحتی قابل پیاده سازی است.

$$S = 1 - \frac{3}{(R+G+B)}[min(R,G,B)]$$

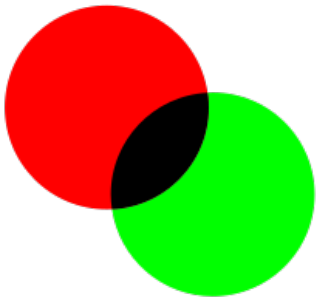
$$I = \frac{1}{3}(R+G+B)$$

برای تبدیل *RGB* به *HSI* تابعی به نام *rgb_to_hsi* پیاده سازی شده است که در ورودی عکس را دریافت میکند. در ابتدا برای راحتی کار مقادیر هرچنل را در یک متغیر ذخیره میکند و در ادامه با استفاده از فرمول ها و نکاتی که گفته شد، مقادیر *HSI* را محاسبه کرده و آنها را برمیگرداند.

۱.۱.۲- در این قسمت خواسته شده است تا سه فضای رنگی که در درس بیان نشده اند را توضیح دهیم.

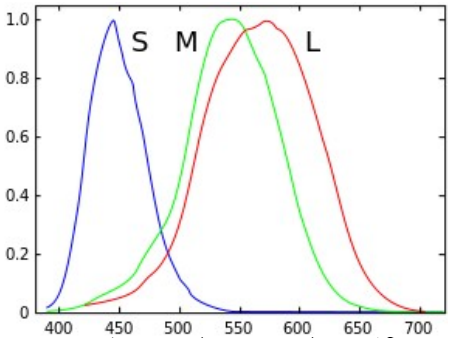
۱. فضای رنگی *RGK*: برای بررسی این فضا ابتدا فضای رنگی *RG* را بررسی میکنیم. فضای رنگی *RG* همانند *RGB* است با این تفاوت که کانال آبی در آن وجود ندارد. و فقط میتواند *shade* هایی از رنگ که با استفاده از قرمز و سبز ساخته میشوند را نمایش دهد.

مشکلی که این فضای رنگی دارد این است که رنگ های سیاه را نمیتواند نمایش دهد برای همین سراغ فضای رنگی *RGK* میرویم که شامل چنل سیاه نیز میشود. این فضای رنگی در ابتدای ظهور عکاسی رنگی کاربرد های زیادی داشته است ولی امروزه بجز در مواردی مثل صنایع بسته بندی و لیبل گذاری بسته ها، کاربردی ندارد و در صنعت چاپ از فضای رنگی *CMYK* استفاده میشود.



شکل ۱- رنگ های اصلی در فضای رنگی subtractive RGK

۲. فضای رنگی *LMS*: این فضای رنگی مخفف شده عبارت *Long-Medium-Short* است و فضای رنگی است که بیانگر سه نوع سلول مخروطی در رتینا چشم انسان است که بر اساس طول موج نام گذاری شده اند. این فضای رنگی برای بررسی *chromatic adaptation* (توانایی سیستم بینایی انسان برای تطبیق با تغییرات نور به منظور حفظ ظاهر رنگ اشیا) استفاده میشود. همچنین برای تحقیق درمورد کوررنگی زمانی که سلولهای مخروطی مشکل دارند و به خوبی کار نمیکنند بسیار کاربرد دارد.



شکل ۲- طیف پاسخ طبیعی سلول های مخروطی در چشم انسان

۳. فضای رنگی *TSL*: این فضای رنگی که مخفف عبارت *Tint-Saturation-Lightness* است، یک فضای رنگی ادراکی است که رنگ را بر اساس *tint* معرفی میکند. در نظریه رنگ ها، *tint* ترکیب یک رنگ با رنگ سفید است که باعث افزایش *lightness* میشود. پارامتر بعدی که *Saturation* است مفهومی همانند *Saturation* در فضای رنگی *HSI* که معرفی شده است دارد و بیانگر میزان *colorfulness* در یک محدوده بر اساس روشنایی میباشد. و *Lightness* نیز یک مفهوم ادراکی از میزان دریافت لومینانس از یک شی است.

برای تبدیل فضای رنگی *RGB* به *TSL* میتوان از فرمول های زیر استفاده کرد.

$$T = \begin{cases} \frac{1}{2\pi} tan^{-1}(\frac{r'}{g'}) + \frac{1}{4} & if g' > 0 \\ \frac{1}{2\pi} tan^{-1}(\frac{r'}{g'}) + \frac{3}{4} & if g' < 0 \\ 0 & if g' = 0 \end{cases}$$

$$S = \sqrt{\frac{9}{5}(r'^2 + g'^2)}$$

$$L = 0.299R + 0.587G + 0.114B$$

که در آن مقادیر به شرح زیر است:

$$\begin{cases} r = \frac{R}{R+G+B} \\ g = \frac{G}{R+G+B} \\ r' = r - \frac{1}{3} \\ g' = g - \frac{1}{3} \end{cases}$$

این فضای رنگی برای کاربرد در تشخیص چهره و صورت (*face recognition*) طراحی شده است. و از آن جایی که فضای رنگی نسبتا جدید تری است و کاربرد های محدود تری دارد، کمتر پیاده سازی شده است. کاربرد اصلی آن در همان *face recognition* و *skin detection* می‌باشد که خود آنها در موارد بسیاری کاربرد دارند.

۱.۲.۱ و ۱.۲.۲- در این تمرین خواسته شده است تا تصویر را به صورت *uniform* گسسته کنیم و در ادامه مقادیر *MSE* و *PSNR* در مقایسه تصویر اصلی و تصاویر *quantize* شده را بررسی کنیم.

برای گسسته سازی، میتوان مقادیر هر کانال را به صورت جداگانه گسسته سازی کرد و از آنجایی که هر کانال همانند یک تصویر *grayscale* است، میتوان از تابع گسسته کننده که در تمرین اول پیاده سازی شد استفاده کرد. همانند تمرین اول، این تابع تصویر و مقداری به عنوان تعداد بیت دریافت میکند و با شیفت دادن مقادیر تصویر به راست و سپس ضرب کردن آنها در ضریب آلفا که برابر با اندازه هر قسمت گسسته شده است، مقادیری گسسته شده از تصویر به دست آورد. این تابع را برای هر کانال تصویر *RGB* به صورت جداگانه اعمال میکنیم و در نهایت کانال های به دست آمده را با یکدیگر ترکیب کرده و به تصویری *RGB* میرسیم که در.اقع همان تصویر گسسته شده میباشد.

برای اینکه تابع ارائه شده را بتوان برای هر دو سوال استفاده کرد و حالتی *general* داشته باشد، تعداد بیت های مورد نظر برای هر کانال را به صورت جداگانه به عنوان ورودی به تابع میدهیم تا بتوان هم برای سوال اول و هم برای سوال دوم از آنها استفاده کرد.

بنابراین تابع گسسته سازی تصویر *RGB*، چهار مقدار به عنوان ورودی دریافت میکند که شامل تصویر و سه مقدار بیت به ازای هر کانال تصویر میباشد و در خروجی تصویر رنگی گسسته شده را برمیگرداند. تصاویر و نتیجه ها در قسمت تحلیل نتیجه ها به صورت کامل بررسی میشوند.

۱.۲.۳- در این تمرین خواسته شده است تا تعداد رنگ های موجود در تصویر را کاهش دهیم. راه های زیادی برای این کار وجود دارد. یک راه حل، همانند تمرین های گذشته است، در واقع با گسسته سازی مقادیر هر کانال، باعث میشویم که رنگ هایی که میتوانند نمایش دهند نیز گسسته شوند. برای مثال اگر برای هر کانال از دو بیت استفاده کنیم، ترکیب هر سه کانال میتواند $2 \times 2 \times 2 = 8$ رنگ ایجاد کند.

اما دو مشکل برای استفاده از این روش در این سوال وجود دارد.

۱- اگر همان مثال قبلی را فرض کنیم، نمیتوان تضمین کرد که هر ۸ رنگ در تصویر وجود داشته باشند و این مقدار حداکثر رنگ ها را به ما میدهد ولی کران پایینی برای آن تعیین نمیکند و درنتیجه نمیتوان به صورت دقیق تعداد رنگ های خواسته شده را نمایش داد.

۲- این روش حالتی *manual* دارد و به صورت دستی باید آنرا *adjust* کرد. برای مثال اگر بخواهیم حداکثر ۱۶ رنگ داشته باشیم، میتوان در دو چنل رنگی مقادیر را با ۲ بیت *quantize* کنیم و در یک چنل با ۴ بیت *quantize* کنیم. این کار باعث میشود حداکثر ۱۶ مقدار رنگی داشته باشیم ولی انتخاب اینکه کدام چنل ۴ بیت داشته باشد و کدام چنل ها ۲ بیت داشته باشند باید به صورت دستی انجام شود که ممکن است خیلی خوب نباشد. اگر هم برای تمامی چنل ها تعداد بیت یکسانی در نظر بگیریم، باعث میشود که فقط توان های ۳ از اعداد طبیعی را به عنوان حداکثر رنگ داشته باشیم و باز هم مقدار دقیقی از تعداد رنگها نخواهیم داشت.

راه بهتر آن است که از یادگیری بدون ناظر و الگوریتم های خوشه بندی استفاده کنیم. با این کار به صورت دستی *adjust* نمیکنیم و خود الگوریتم مراکز کلاسترکه به عنوان رنگ های نهایی در نظر گرفته میشوند را نتیجه میدهد. باید توجه داشته باشیم که برای حل این مسئله نمیتوان از هر الگوریتم خوشه بندی استفاده کرد. برای مثال اگر از الگوریتم های خوشه بندی که براساس تراکم خوشه بندی میکنند استفاده کنیم (همانند *DBSCAN* و *Mean Shift*) باز هم تعداد دقیق کلاستر ها را نخواهم داشت و در نتیجه تعداد دقیق رنگ ها نیز مشخص نخواهد شد. بنابراین بهتر است از الگوریتم های خوشه بندی استفاده شود که تعداد کلاستر ها را به عنوان ورودی دریافت میکند. برای این کار میتوان از الگوریتم خوشه بندی *K-Means* استفاده کرد. برای این کار از تابع آمده در کتابخانه *SciKit-Learn* استفاده شده است.

برای حل این سوال تابعی طراحی شده است با نام *reduce_color* که در ورودی تصویر و k که تعداد رنگ های مورد نظر است را دریافت میکند. سپس مدل *K-Means* را با استفاده از کتابخانه *SciKit* ساخته و آنرا روی تصویر اعمال میکنیم. قبل از این کار باید تغییراتی روی تصویر به وجود بیاوریم زیرا *K-Means* ورودی را به صورتی دریافت میکند که سطر های آن داده ها باشند و ستون ها بیانگر ویژگی های باشد ولی تصویر ما دارای ۳ کانال است. تصویر را به صورتی *reshape* میکنیم که مقادیر *RGB* مربوط به هر پیکسل که درواقع ویژگی های پیکسل هستند در ستون ها قرار بگیرند. اگر اندازه تصویر در هر چنل را N در نظر بگیریم، با این کار به داده ای دوبعدی با ابعاد $(N \times N, 3)$ میرسیم که هر پیکسل تصویر اصلی، ردیفی در آن است و مقادیر هر چنل رنگی آن در ستون ها قرار گرفته اند.

بعد از انجام الگوریتم *K-Means*، روی مقادیر *label* های خروجی پیمایش میکنیم و به ازای پیکسل هایی که داخل یک خوشه قرار گرفته اند، مرکز آن خوشه را قرار میدهیم و با این کار، تعداد k رنگ در تصویر خروجی خواهیم داشت.

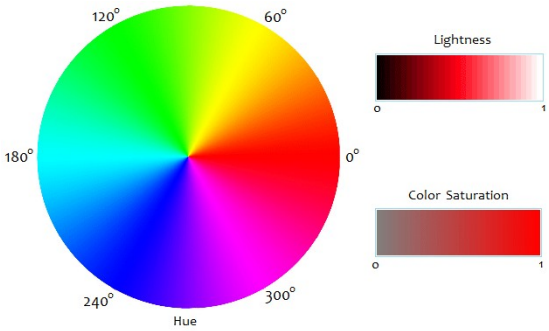
۲- تحلیل نتایج

۲.۱.۱- در شکل های ۳و ۴ و ۵ میتوان نتایج مربوط به هرکدام از چنل های *HSI* را مشاهده کرد. پارامتر اصلی که رنگ را بیان میکند *Hue* است که در ادامه به بررسی آن می‌پردازیم.

شکل ۳-مقادیر Hue برای تصویر Lena

شکل ۴-مقادیر Saturation برای تصویر Lena

شکل ۵-مقدار Intensity برای تصویر Lena



شکل ۶-نمایش رنگ ها در فضای رنگی HSI

شکل ۷-تصویر Lena در فضای رنگی RGB

در تصویر رنگی *RGB* مشاهده میشود که میزان قرمز در تصویر بیشتر است. از آنجایی که قرمز در *Hue* بیانگر زاویه ۰ درجه یا ۳۶۰ درجه است، در نتیجه *Hue* این مقادیر را باید بیشتر مشاهده

کنیم. در قسمت موهای *Lena*، میتوان مشاهده کرد که نسبت رنگ آبی به سبز بیشتر است و درنتیجه رنگهایی با زاویه ای بزرگ (بین ۳۰۰ تا ۳۶۰ درجه) تشکیل شده است و برای همین در نتیجه *Hue* میتوان برای قسمت مو ها مقادیر بزرگی قرار گرفته است. بر خلاف موها، در قسمت صورت میتوان مشاهده کرد که رنگ بیشتر به نارنجی نزدیک است و یعنی میزان سبز از آبی بیشتر است و درنتیجه زاویه کوچک خواهد بود و مقادیر کوچکتری در *Hue* قابل مشاهده است. برای بقیه نواحی هم میتوان به این ترتیب مقایسه را انجام داد.

برای *Saturation* هم میتوان مشاهده کرد که در نواحی که پررنگ تر است و رنگ بیشتری داریم مقدار بیشتر و در نواحی کم رنگ تر مقدار *Saturation* نیز کم تر میباشد.

۲.۱.۲- این تمرین در قسمت قبل توضیح داده شد و نتیجه ای ندارد.

۲.۲.۱- در این تمرین خواسته شده است تا چنل های تصویر را با استفاده از تعداد سطح های داده شده گسسته سازی کنیم. این مقادیر برابر با ۶۴ سطح، ۳۲ سطح، ۱۶ سطح و ۸ سطح میباشد که برای گسسته سازی آنها با استفاده از تابع معرفی شده در بخش اول میبایست به ترتیب از ۶، ۵، ۴ و سه بیت استفاده شود. نتایج را میتوان در تصاویر زیر مشاهده کرد.

شکل ۸- تصویر Lena که هر کانال رنگی آن با استفاده از ۶ بیت گسسته شده است که بیانگر ۶۴ سطح در هر کانال می‌باشد.

همانطور که مشاهده میشود، نمیتوان تفاوت زیادی بین تصویر *Lena* و تصویر گسسته شده با استفاده از ۶ بیت مشاهده کرد.

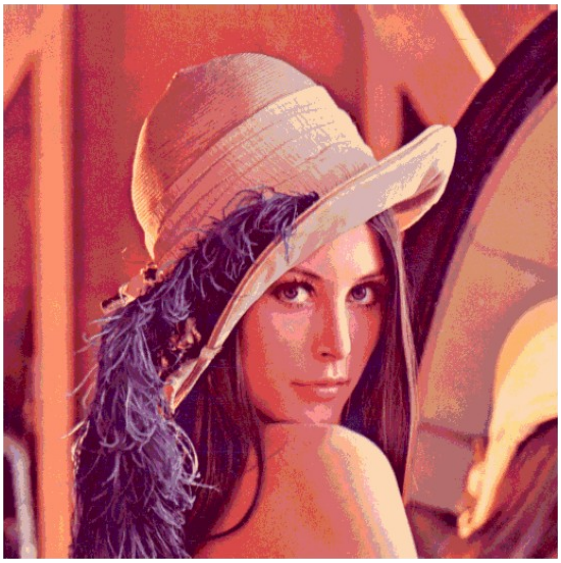
شکل ۹- تصویر Lena که هر کانال رنگی آن با استفاده از ۵ بیت گسسته شده است که بیانگر ۳۲ سطح در هر کانال می باشد.

شکل ۱۰- تصویر Lena که هر کانال رنگی آن با استفاده از ۴ بیت گسسته شده است که بیانگر ۱۶ سطح در هر کانال می‌باشد.

همانطور که مشاهده میشود، در تصویر گسسته شده با استفاده از ۵ بیت که شامل ۳۲ سطح میشود هم با چشم نمیتوان خیلی تفاوتی را نسبت به تصویر اصلی *Lena* تمایز داد.

اما در شکل ۱۰ که بیانگر تصویر گسسته شده با ۴ بیت یا همان ۱۶ سطح است، میتوان گسستگی در تصویر را مشاهده کرد. این گسستگی برای ناحیه های شلوغ (در تصویر *Lena* ناحیه مو ها شلوغ حساب میشود و جزئیات زیادی دارد) به خوبی مشخص نیست ولی برای ناحیه هایی که خلوت است و تمایز در روشنایی وجود دارد به خوبی قابل مشاهده است. برای مثال در ثسمت صورت و یا دست میتوان این گسستگی را به خوبی مشاهده کرد. در نواحی شلوغ همان طور که در فصل های گذشته مشاهده

کردیم تعداد بیت ها نقش کلیدی ایفا نمیکند بلکه نرخ نمونه برداری مهم تر است. (منحنی های *isopreference* نشان دادند که هرچقدر تصویر آرام تر باشد تعداد سطوح بیشتری قابل تمایز است و هر چقدر تصویر شلوغتر باشد، تعداد پیکسل ها مهم تر است).



شکل ۱۱- تصویر Lena که هر کانال رنگی آن با استفاده از ۳ بیت گسسته شده است که بیانگر ۸ سطح در هر کانال می‌باشد.

در شکل ۱۱ میتوان به وضوح گسستگی در تصویر را مشاهده کرد. ولی باز هم در ناحیه شلوغ توسط چشم خیلی قابل تمایز نیست ولی در بقیه قسمت ها به خوبی قابل مشاهده میباشد.

در قسمت بعدی خواسته شده است تا مقادیر *MSE* و *PSNR* گزارش شوند. مقدار *PSNR* برای تصاویر مورد بررسی ما اگر بین ۳۰ تا ۵۰ باشد مقدار خوبی میباشد که هر چقدر بزرگتر باشد بهتر است. و بیانگر این است که نویز خیلی تصویر را نسبت به تصویر اصلی خراب نکرده است و برای *compress* کردن تصویر نتیجه شده مناسب است.

جدول ۱- مقادیر PSNR و MSE به ازای هر سطح در تصویر Lena

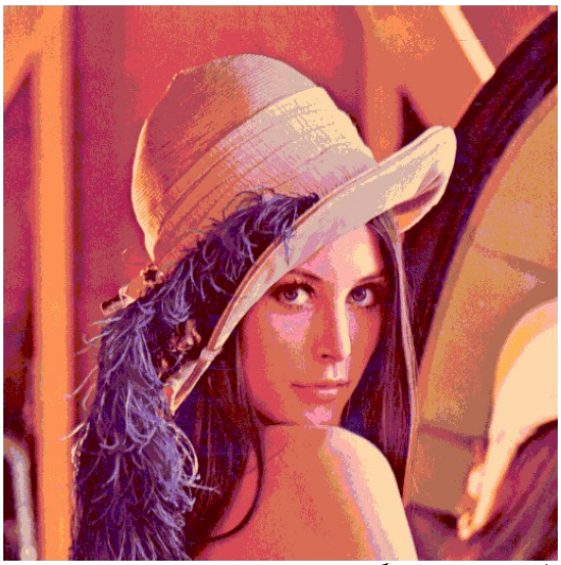
تعداد سطح ها	مقدار MSE	مقدار PSNR
۶۴	۴.۶۷	۴۶.۲۰
۳۲	۳۹.۹۶	۳۶.۸۸
۱۶	۹۳.۹۳	۳۳.۱۷
۸	۴۷۷.۲۰	۲۶.۱۱

همانطور که مشاهده میشود، در ۶۴ سطح خیلی تفاوتی با تصویر اصلی ایجاد نمیشود به صورتی که مقدار *mse* هم خیلی کوچک است و *psnr* هم مقدار خوبی است.

هر چقدر تعداد سطح ها کاهش پیدا میکند، *mse* بیشتر شده و *psnr* کمتر میشود. همانطور که در تصاویر نیز مشاهده کردیم، در سه تصویر اول تفاوت کمتری با چشم قابل مشاهده بود و برای این سه تصویر نیز مقادیر *psnr* بالای ۳۰ داریم که قابل قبول است ولی در تصویر آخر، مقدار *psnr* کوچکتر از ۳۰ است که نشان میدهد نویز ها و تغییرات با چشم به خوبی قابل تمایز است.

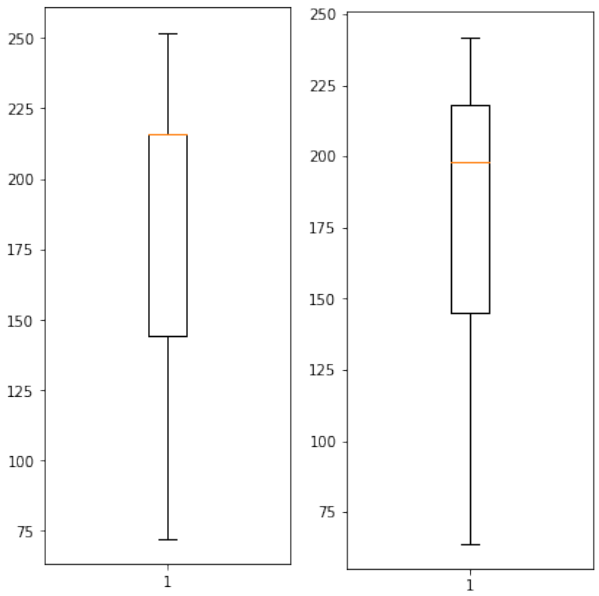
۲.۲.۲- در این تمرین خواسته شد تا تعداد سطح های گسسته شده در کانال ها متفاوت باشد. خواسته شده است تا کانال قرمز با ۳ بیت، کانال سبز هم با ۳ بیت و کانال آبی با ۲ بیت در تصویر *Lena* گسسته شوند.

شکل ۱۲- تصویر اصلی Lena



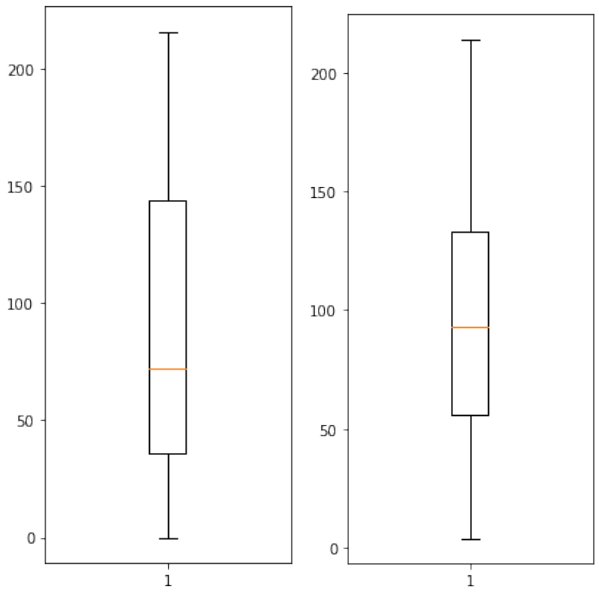
شکل ۱۳- تصویر گسسته شده با ۳ بیت برای کانال قرمز، ۳ بیت برای کانال سبز و ۲ بیت برای کانال آبی.

همانطور که مشاهده میشود، میتوان دید که تصویر میزان نارنجی بودنش بیشتر شده است.



شکل ۱۴- نمودار box-plot مربوط به کانال قرمز تصویر اصلی Lena

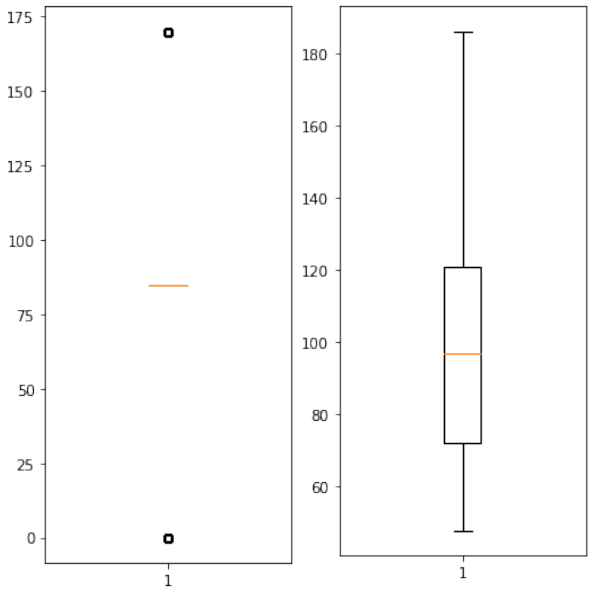
شکل ۱۵- نمودار box-plot مربوط به کانال قرمز تصویر گسسته شده Lena



شکل ۱۷- نمودار box-plot مربوط به کانال سبز تصویر اصلی Lena

شکل ۱۶- نمودار box-plot مربوط به کانال سبز تصویر گسسته شده Lena

همانطور که مشاهده میشود، با گسسته سازی در کانال قرمز، میانه نزدیک به چارک سوم شده است ومقدار آن نیز در مقایسه با نمودار اصلی بیشتر شده است. کانال سبز هم مقار میانه با چارک اول نزدیکتر شده است و میتوان نتیجه گرفت مقادیر سبز کمتر شده اند. این باعث میشود تا قسمت هایی که نارنجی کمرنگ تری بوده اند، نارنجی پررنگ تری شوند زیرا قرمز آنها تقویت و سبز کمتر شده است.



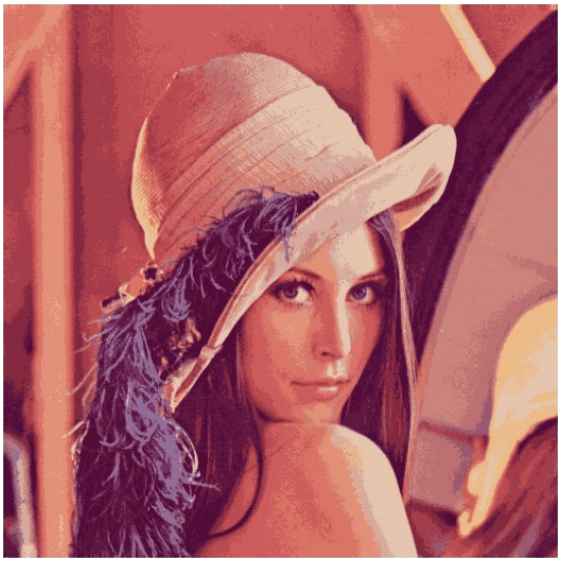
شکل ۱۸- نمودار box-plot مربوط به کانال آبی تصویر اصلی Lena

شکل ۱۹- نمودار box-plot مربوط به کانال آبی تصویر گسسته شده Lena

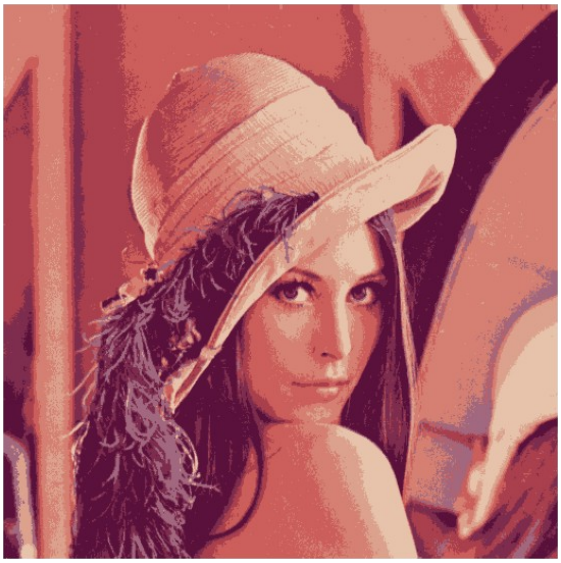
از آنجایی که برای کانال آبی ۲ بیت برای گسسته سازی در نظر گرفتیم، کلا ۴ مقدار آبی قابل دسترسی است و همانطور که مشاهده میشود، اکثر مقادیر کانال آبی به یک مقدار map شده اند که نسبت به میانه در تصویر اصلی نیز مقدار کمتری دارد. در نتیجه مقادیر آبی نیز در بیشتر نقاط کاهش پیدا کرده اند.

۲.۲.۳- در این تمرین خواسته شده است تا رنگهای تصویر را به ۳۲ رنگ، ۱۶ رنگ و ۸ رنگ کاهش دهیم. برای این کار همانطور که گفته شد از الگوریتم خوشه بندی K-Means استفاده میکنیم. درادامه نتایج را میتوان مشاهده کرد.

شکل ۲۰- تصویر Lena با ۳۲ رنگ.



شکل ۲۱- تصویر Lena با ۱۶ رنگ.



شکل ۲۲- تصویر Lena با ۸ رنگ.

همانطور که مشاهده میشود، در شکل ۲۰ که تصویر *Lena* با استفاده از ۲۳ رنگ نمایش داده شده است، با چشم نمیتوان خیلی تفاوت هارا نسبت به تصویر اصلی مشاهده کرد. ولی در تصاویر با ۱۶ رنگ و ۸ رنگ این تفاوت با چشم به خوبی قابل تمایز است.

برای بررسی بهتر، مقادیر *MSE* و *PSNR* هم مورد بررسی قرار گرفته اند.

جدول ۲- مقادیر PSNR و MSE برای تصویر کاهش رنگ یافته Lena در مقایسه با تصویر اصلی.

تعداد رنگ ها	مقدار MSE	مقدار PSNR
۳۲	۸۸.۹۹	۳۳.۴۰
۱۶	۱۷۸.۹۷	۳۰.۳۷
۸	۳۶۴.۶۸	۲۷.۲۸

مشاهده میشود که در شکل ۲۱، مقدار *PSNR* خیلی لب مرز است و در تصویر هم میتوان گسستگی ها در تصویر را مشاهده کرد. در شکل ۲۲ هم که *PSNR* کمتر از ۳۰ میباشد و گسستگی ها به خوبی قابل مشاهده هستند.

این کار را برای تصویر *Baboon* نیز انجام میدهیم. در تصاویر زیر، ورژن اصلی تصویر *Baboon* به همراه تصاویر کاهش رنگ یافته با استفاده از ۳۲ رنگ، ۱۶ رنگ و ۸ رنگ قابل مشاهده است.

شکل ۲۳- تصویر Baboon.

شکل ۲۴- نمایش تصویر Baboon با استفاده از ۳۲ رنگ.

همانطور که مشاهده میشود، در تصویر *Baboon* برخلاف تصویر لنا، *shade* های متفاوتی از رنگها موجود است. با کاهش رنگ تصویر به ۳۲ رنگ، *shade* های مختلف رنگ کمتر میشوند برای

مثال در قسمت موهای زرد *Baboon* میتوان مشاهده کرد که برخی از لبه های زرد از بین رفته اند و به رنگی یکسان تبدیل شده اند ولی این تفاوت در لحظه اول با چشم قابل مشاهده نخواهد بود زیرا برخلاف تصویر لنا، این تصویر شلوغتر است.

شکل ۲۵- نمایش تصویر Baboon با استفاده از ۱۶ رنگ.

شکل ۲۶- نمایش تصویر Baboon با استفاده از ۸ رنگ.

با کاهش بیشتر رنگها، میتوان مشاهده کرد که تعداد *shade* های تصویر نیز کمتر میشود به صورتی که در شکل ۲۶ میتوان مشاهده کرد که تنها یک *shade* رنگ زرد باقی مانده است و رنگ نارنجی هم به کلی از بین رفته است و چشم های *Baboon* به نزدیکترین مرکز کلاستر که قرمز بوده است تغییر پیدا کرده اند.و لبه های زیادی نیز از بین رفته است.

در جدول۳، مقادیر *MSE* و *PSNR* برای تصویر *Baboon* مشاهده میشود.

جدول ۳- مقادیر PSNR و MSE برای تصویر کاهش رنگ یافته Baboon در مقایسه با تصویر اصلی.

تعداد رنگ ها	مقدار MSE	مقدار PSNR
۳۲	۳۷۴.۰۸	۲۷.۱۷
۱۶	۶۲۸.۳۶	۲۴.۹۱
۸	۱۱۱۷.۰۳	۲۲.۴۲

میتوان مشاهده کرد از آنجایی که تصویر *Baboon* نسبت به *Lena* دارای *shade* های بیشتری از رنگها است، با کاهش رنگ خطای خیلی بیشتری نسبت به تصویر اصلی میگیرد.

```
1
2  import cv2
3  import numpy as np
4  import matplotlib.pyplot as plt
5  import math
6  from sklearn.cluster import KMeans
7  # %matplotlib inline
8
9
10 lena = cv2.imread('Lena.bmp')
11 baboon = cv2.imread('Baboon.bmp')
12
13 def show_img(*args, figsize=10, is_gray=True, title=None, fontsize=12):
14     if isinstance(figsize, int):
15         figsize = (figsize, figsize)
16     images = args[0] if type(args[0]) is list else list(args)
17     cmap=None
18     if not is_gray:
19         images = list(map(lambda x: cv2.cvtColor(x, cv2.COLOR_BGR2RGB), images))
20     else:
21         cmap = 'gray'
22     plt.figure(figsize=figsize)
23     for i in range(1, len(images)+1):
24         plt.subplot(1, len(images), i)
25         if title is not None:
26             plt.title(title[i-1], fontsize=fontsize)
27
28         plt.imshow(images[i-1], cmap=cmap)
29         plt.axis('off')
30
31 show_img(lena, is_gray=False)
32
33 #----------------------6.1.1----------------------#
34 def rgb_to_hsi(img):
35     b, g, r = img[:, :, 0], img[:, :, 1], img[:, :, 2]
36
37     b = b / 255.0
38     g = g / 255.0
39     r = r / 255.0
40
41     num = 0.5*(r-g) + 0.5*(r-b)
42     denum = np.sqrt((r-g)**2 + (r-b)*(g-b))
43
44     theta = np.arccos(num/(denum+0.00001))
45
46     H = np.zeros((img.shape[0], img.shape[1]))
47     S = np.zeros((img.shape[0], img.shape[1]))
48
49     for i in range(img.shape[0]):
50         for j in range(img.shape[1]):
51
52             if denum[i, j] != 0:
53                 if b[i, j] <= g[i, j]:
54                     H[i, j] = theta[i, j]
55                 else:
56                     H[i, j] = 2*math.pi - theta[i, j]
57
58             _min = min(min(r[i, j], g[i, j]), b[i, j])
59             sum = r[i, j] + g[i, j] + b[i, j]
```

```
            if (sum != 0):
                S[i, j] = 1 - 3*_min/sum

    H = H/(2*math.pi) * 255

    I = (r+g+b)/3

    return H.astype('uint8'), S, I

h, s, i = rgb_to_hsi(lena)

show_img(h)
show_img(s)
show_img(i)

#----------------------6.2.1----------------------#
def quantize(img, k):
    n = 8 - k
    img = img >> n
    alpha = math.floor(255/(2**k - 1))
    return img * alpha

def mse(img1, img2):
    diff = (img1 - img2) ** 2
    return np.sum(diff.ravel()) / (img1.shape[0] * img1.shape[1])

def quantize_rgb(img, k1, k2, k3):
    B, G, R = img[:, :, 0], img[:, :, 1], img[:, :, 2]

    R = quantize(R, k1)
    G = quantize(G, k2)
    B = quantize(B, k3)

    r = np.zeros((img.shape[0], img.shape[1], img.shape[2]))
    r[:, :, 0] = B
    r[:, :, 1] = G
    r[:, :, 2] = R

    return r.astype('uint8'), mse(img, r), cv2.PSNR(img, r.astype('uint8'))

for i in [3, 4, 5, 6]:
    r, mse_v, psnr_v = quantize_rgb(lena, i, i, i)

    show_img(r, is_gray=False)
    print('number of bits: ', i, end=' -> ')
    print ('psnr is: ', psnr_v, end='   ')
    print ('mse is: ', mse_v)

#----------------------6.2.2----------------------#

r, m, p = quantize_rgb(lena, 3, 3, 2)
show_img(r, is_gray=False)

fig = plt.figure(figsize =(3, 7))
plt.boxplot(lena[:,:,0].flatten())
plt.show()

fig = plt.figure(figsize =(3, 7))
plt.boxplot(r[:,:,0].flatten())
plt.show()
```

```
120
121 #----------------------6.2.3----------------------#
122 def reduce_color(img, k):
123     shape = img.shape
124     kmeans_model = KMeans(n_clusters=k, random_state=0)
125     kmeans_model.fit_predict(img.reshape(shape[0]*shape[1], shape[2]))
126     centroids = kmeans_model.cluster_centers_.astype('uint8')
127     l = kmeans_model.labels_.reshape(shape[0], shape[1])
128     r = np.empty(shape=shape)
129 
130     for i in range(l.shape[0]):
131         for j in range(l.shape[1]):
132             r[i,j,:] = centroids[l[i, j]]
133 
134     return r.astype('uint8'), mse(img, r), cv2.PSNR(img, r.astype('uint8'))
135 
136 for i in [32, 16, 8]:
137     quan, m, p = reduce_color(baboon, i)
138     show_img(quan, is_gray=False)
139     print('number of colors: ', i, end=' -> ')
140     print ('psnr is: ', p, end='   ')
141     print ('mse is: ', m)
142 
```