رنگ

رضا عباس زاده

اطلاعات گزارش

تاریخ:

واژگان کلیدی:
Color space
RGB
HIS
Quantization
K-means clustering

چکیده

رنگ یکی از مولفه‌های اصلی در توصیف تصاویر است که به شناسایی و استخراج اشیا در تصویر کمک می‌کند. در این بخش با فضاهای رنگی RGB و HSI آشنا شده و کاربردهای هریک بررسی می‌شوند. سپس به گسسته سازی رنگ‌های تصویر پرداخته می‌شود و روش بهینه‌ای برای گسسته‌سازی معرفی می‌شود.

۱-مقدمه

استفاده از رنگ در پردازش تصویر، ناشی از دو عامل است. اولاً، رنگ توصیفگر قدرتمندی است که غالباً شناسایی و استخراج اشیا را از صحنه آسان می‌سازد. ثانیاً، انسان می تواند در مقایسه با فقط ۲۴ سایه خاکستری، هزاران سایه رنگ و شدت را تشخیص دهد. این عامل دوم، مخصوصاً در تحلیل تصویر دستی (یعنی وقتی که توسط انسان انجام می‌گیرد) مهم است.

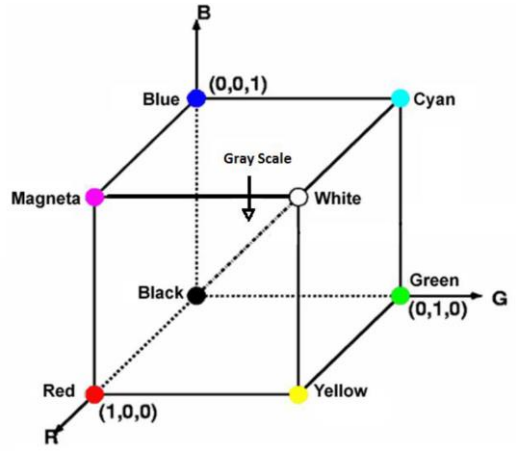
پردازش تصویر رنگی به دو ناحیه مهم تقسیم می شود: پردازش تمام رنگی و شبه رنگی. در دسته اول، رنگی یا اسکنر رنگی. در دسته دوم، TVتصاویر معمولاً توسط حسگر تما مرنگی دریافت می شوند، مثل دوربین مسئله، تخصیص رنگ به شدت تک رنگ خاص یا بازه ای از شدت ها است. تقریباً تاکنون، اغلب پردازش های تصویر رنگی در سطح شبه رنگی انجام شدند. اما، در دهه گذشته، حسگرهای رنگی و سخ تافزار مربوط به پردازش تصاویر رنگی، با قیمت‌های مناسبی فراهم شدند.

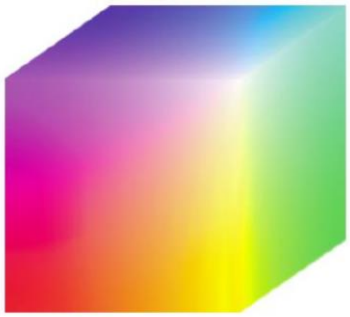
نتیجه این است که اکنون تکنیک های پردازش تصویر تمام رنگی در گستره ای از کاربردها استفاده می شوند، از جمله چاپ و نشر، تجسم و اینترنت.

۲-شرح تکنیکال

فضای رنگ RGB

در این فضای رنگی، هر رنگ از ترکیب سه رنگ اصلی قرمز، سبز و آبی به وجود می‌آید. با در نظر گرفتن ۸ بیت برای ذخیره‌سازی هر رنگ اصلی، به هریک از آن‌ها عددی در بازه ۰ تا ۲۵۵ اختصاص می‌دهیم. بنابراین برای ذخیره هر رنگ به ۲۴ بیت نیاز خواهیم داشت. این فضای رنگی مکعبی به شکل زیر تشکیل می‌دهد:





چشم انسان رنگ‌ها را در این فضای رنگی دریافت می‌کند.

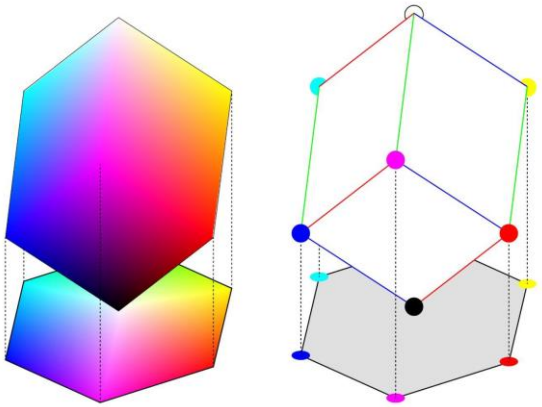
فضای رنگ HSI

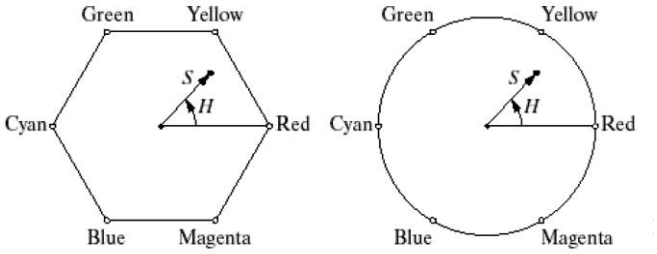
در این فضای رنگی، برای توصیف یک رنگ از سه مولفه hue، saturation و intensity استفاده می‌کنیم.

مولفه hue یا فام مشخص کننده طول موج رنگ غالب است.

مولفه saturation یا اشباع میزان خلوص رنگ را مشخص می‌کندو به این معنی که هرچه رنگ به رنگ اصلی نزدیک تر باشد، اشباع بالاتری دارد و هرچه رنگ سفید به آن اضافه شود از اشباع آن کاسته می‌شود. مولفه intensity یا شدت میزان روشنایی آن رنگ را مشخص می‌کند. هرچه روشنایی یک رنگ کم شود به رنگ سیاه نزدیک‌تر می‌شود.

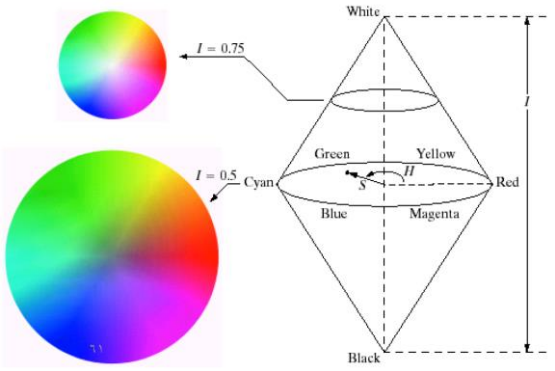
اگر مکعب rgb را دوران دهیم، می‌توان این فضای رنگی را توصیف کرد:





در این تصاویر می‌بینید که رنگ قرمز به عنوان مبدا فام درنظر گرفته می‌شود و زاویه‌ای که با رنگ مورد نظر با رنگ قرمز می‌سازد، فام تصویر می‌باشد.

اشباع تصویر با فاصله نسبت به مرکز شش‌ضلعی رابطه مستقیم دارد و هرچه به لبه‌ها نزدیک‌تر باشد اشباع بالاتری دارد.



نحوه به دست آوردن شدت رنگ نیز در تصویر بالا مشخص شده‌است. می‌بینیم که هرچه شدت بیشتر باشد به رنگ سفید میل می‌کند و هرچه کوچکتر باشد به رنگ سیاه.

مغز انسان از این فضای رنگی برای پردازش رنگ‌ها استفاده می‌کند.

تبدیل فضای رنگ rgb به hsi

فرمول محاسبه فام رنگ:

$$H=\begin{cases}\theta & \text{if } B\le G\\ 360-\theta & \text{if } B>G\end{cases} \qquad \theta=\cos^{-1}\left\{\frac{\frac{1}{2}[(R-G)+(R-B)]}{[(R-G)^2+(R-B)(G-B)]^{1/2}}\right\}$$

از این فرمول می‌توان ثابت کرد که اگر مقادیر r، g و b تحت یک تابع ثابت اسکیل شوند، فام آن رنگ تغییر نخواهد کرد.

فرمول محاسبه اشباع رنگ:

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

فرمول محاسبه شدت رنگ:

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

اگر مقادیر قرمز، سبز و آبی به یکدیگر عوض شوند، اشباع و شدت آن رنگ تغییری نخواهد کرد.

در تمامی این فرمول‌ها مقادیر rgb به بازه ۰ تا ۱ اسکیل شده‌اند.

۵-۱-۲

فضای رنگ XYZ

در سال CIE ۱۹۳۱ اقدام به معرفی یک مجموعه رنگ پایه ی جدید برای برطرف کردن مشکلات فضای رنگ RGB نمود و فضای رنگ XYZ را معرفی کرد.

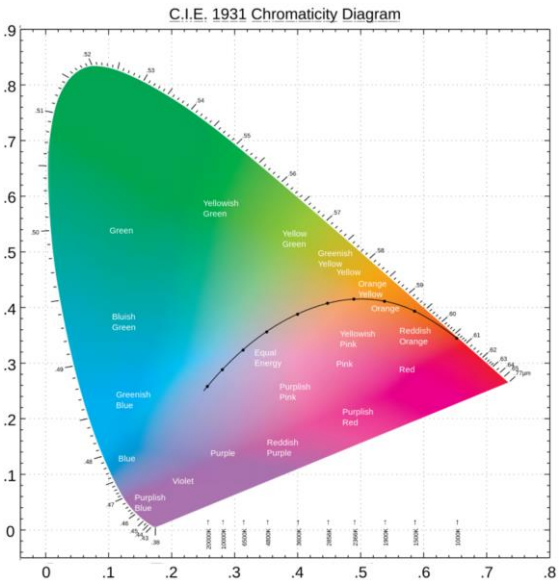
مشکل فضای رنگ قبلی در این بود که برای مشاهده‌گرهای مختلف نظیر انسان ها، سنسورها و ... تعریف واحدی برای دریافت اطلاعات رنگی موجود نبود.

یعنی در فضای RGB مشخصات رنگی نیمه شهودی و وابسته به وسایل اندازه‌گیری است. در این فضای رنگ پایه-ی جدید، اجزای رنگ های پایه رنگ های واقعی نیستند. میزان درخشندگی رنگ در این فضا برخلاف فضای قبل فقط به یک پارامتر Y بستگی دارد و بقیه پارامترها، ویژگی-های رنگی را پوشش می‌دهند.

فرمول تبدیل فشای رنگ RGB به XYZ و برعکس:

$$\begin{bmatrix}X\\Y\\Z\end{bmatrix}=\begin{bmatrix}0.412453 & 0.357580 & 0.180423\\0.212671 & 0.715160 & 0.072169\\0.019334 & 0.119193 & 0.950227\end{bmatrix}*\begin{bmatrix}R\\G\\B\end{bmatrix}$$

$$\begin{bmatrix}R\\G\\B\end{bmatrix}=\begin{bmatrix}3.240479 & -1.537150 & -0.498535\\-0.969256 & 1.875992 & 0.041556\\0.055648 & -0.204043 & 1.057311\end{bmatrix}*\begin{bmatrix}X\\Y\\Z\end{bmatrix}$$



فضاهای رنگی YUV و YIQ

این فضاها با تغییرات کوچکی از روی مدل R'G'B' بدست آمده است. این تغییرات برای آن است که مدل جدید برای انتقال، نسبت به مدل RGB کارآمدتر شود و همین طور با سیستم تلویزیون های سیاه و سفید، سازگار گردد. در حقیقت جزء Y از این فضاها اطلاعات ویدیویی مورد نیاز یک سیستم تلویزیون سیاه و سفید را بطور کامل تامین می کند. برای تبدیل فضای RGB به فضاهای YIQ و YUV و بالعکس از روابط زیر استفاده می شود:

$$\begin{bmatrix}Y\\I\\Q\end{bmatrix}=\begin{bmatrix}0.299 & 0.587 & 0.114\\0.596 & -0.275 & -0.321\\0.212 & -0.523 & 0.311\end{bmatrix}\begin{bmatrix}R'\\G'\\B'\end{bmatrix} \qquad \begin{bmatrix}R'\\G'\\B'\end{bmatrix}=\begin{bmatrix}1 & 0.9557 & 0.6199\\1 & -0.2716 & -0.6469\\1 & -1.1082 & 1.7051\end{bmatrix}\begin{bmatrix}Y\\I\\Q\end{bmatrix}$$

تبدیل از فضای رنگ RGB به فضای YIQ (سیستم PAL) و بالعکس

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.147 & -0.289 & 0.436 \\ 0.615 & -0.515 & -0.100 \end{bmatrix} \begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} \qquad \begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.1398 \\ 1 & -0.3946 & -0.5805 \\ 1 & 2.0320 & -0.0005 \end{bmatrix} \begin{bmatrix} Y \\ U \\ V \end{bmatrix}$$

تبدیل از فضای رنگ RGB به فضای YUV (سیستم NTSC) و بالعکس

توجه داشته باشید که:

$$U = 0.492\,(B'-Y)\,,\quad V = 0.877(R'-Y)$$

$$I = V\cos 33 - U\sin 33\,,\quad Q = V\sin 33 + U\cos 33$$

حساسیت سیستم بینایی انسان به تغییرات روشنایی بیشتر از حساسیت آن به ته رنگ و اشباع است و فضاهای YIQ و YIQ به گونه ای طراحی شده اند که از این خاصیت استفاده کنند. به همین دلیل در استانداردهای منطبق بر این فضاها ، بیشتر پهنای باند (تعداد بیت در رنگ دیجیتال) در دسترس را، به پارامتر Y اختصاص می دهند و پهنای باند کمتری را برای اجزای کرومینانس نگه می‌دارند.

مزیت عمده دیگری که این فضاها نسبت به مدل RGB دارند آن است که پارامتر مشخص کننده روشنایی Y از دو پارامتر مشخص کننده رنگ جدا است. با توجه به آن که متغیر روشنایی متناسب با مقدار نوری است که چشم بیننده دریافت می کند، با استفاده از این خاصیت می توان روشنایی یک تصویر را تغییر داد بدون آنکه در ترکیب رنگ آن تغییری ایجاد شود. برای مثال می توان تکنیک یکسان سازی هیستوگرام را تنها در مورد پارامتر Y از تصویری با فرمت YIQ یا YUV اعمال کرد. در این حالت رنگ تصویر تغییری نخواهد کرد.

فضای رنگ YCbCr

این فضای رنگی نیز عمدتا در تصاویر ویدیوی دیجیتال و در کاربردهایی که روی این تصاویر انجام می شود، مورد استفاده قرار م یگیرد. در این فضای رنگی Y لومینانس بوده و Cb و Cr نشان دهنده ی کرومینانس می باشند. حرکت از فضای R'G'B' به YCbCr و بالعکس توسط ماتریس-های زیر قابل انجام است (فرض شده است که از فضای R'G'B' دیجیتال که مولفه های آن در بازه ی ۰ و ۲۵۵ واقع هستند استفاده شده است):

$$\begin{bmatrix} Y_{601} \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.257 & 0.504 & 0.098 \\ -0.148 & -0.291 & 0.439 \\ 0.439 & -0.368 & -0.071 \end{bmatrix} \begin{bmatrix} R' \\ G' \\ B' \end{bmatrix}$$

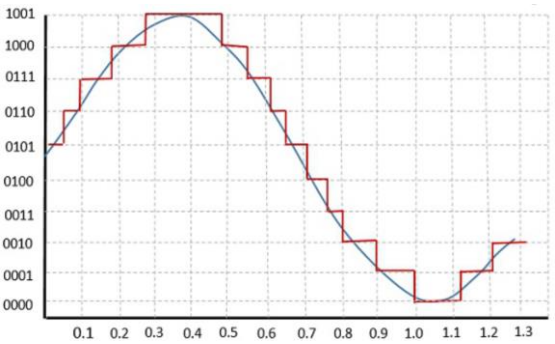
$$\begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} = \begin{bmatrix} 1.164 & 0 & 1.596 \\ 1.164 & -0.391 & -0.813 \\ 1.164 & 2.018 & 0 \end{bmatrix} \begin{bmatrix} Y_{601} \\ Cb \\ Cr \end{bmatrix}$$

دقت کنید که از مقادیر Cb و Crی که نهایتا استفاده می شوند، مقدار لومینانس کم شده است (اگر قبول ندارید، روابط بالا را دوباره نویسی و ساده کنید)، در نتیجه این اجزای کرومینانس مستقل از تغییرات نوری (انرژی) بوده و در کاربردهایی که مایل باشیم تغییرات نوری کمترین تاثیر را داشته باشند، استفاده از این جزء مناسب می‌نمایاند.

گسسته‌سازی یا Quantization

گسسته‌سازی یک نوع روش فشرده سازی است که سطوحی که به یکدیگر نزدیک هستند را یکسان درنظر میگیرد. بنابراین تعداد سطوح کاهش می‌یابد و نیاز به فضای ذخیره-سازی کمتری خواهند داشت.

در این تمرین هر سطح خاکستری دارای مقداری بین ۰ تا ۲۵۵ می‌باشد. اگر هر دوسطح کجاور را یکسان درنظر بگیریم آنگاه بازه به عنوان مثال اعداد ۰ و ۱ هردو در فضای جدید مقدار صفر خواهند داشت. بنابراین برای ذخیره هر سطح به یک بیت کمتر یعنی ۷ بیت نیاز داریم.



اگر بخواهیم L سطح داشته باشیم:

$$d = \frac{255}{L}$$

$$F = \left\lfloor \frac{f}{d} \right\rfloor \times d$$

Quantization with minimum loss

برای اینکه در فرایند گسسته‌سازی اطلاعات کمتری از بین برود، باید سطوحی که پیکسل‌های زیادی حول آن مقدار تجمع دارند را پیدا کنیم و سپس تمام پیکسل‌ها را به نزدیک‌ترین مرکز پیدا شده نسبت می‌دهیم. این عمل در واقع مشابه با الگوریتم K-means از روش‌های خوشه-بندی unsupervised می‌باشد.

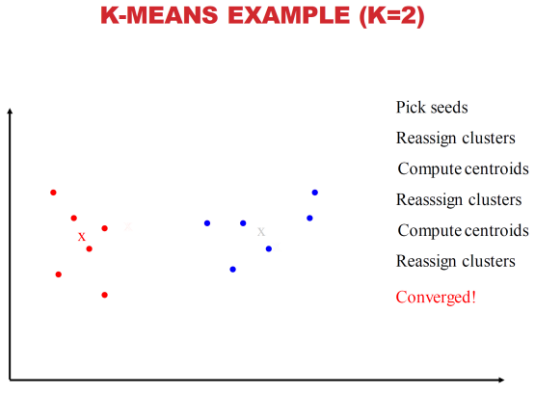
در این الگوریتم، ابتدا به تعداد سطوحی که نیاز داریم، در هر کانال قرمز، سبز و آبی نقاط رندوم در نظر می‌گیریم و فرض می‌کنیم که مراکز هر خوشه همان نقاط هستند.

سپس هر پیکسل را با توجه به فاصله‌ای که با این مراکز خوشه ها دارد، به یکی از این خوشه‌ها نسبت می‌دهیم.

بعد از آن، در هر خوشه، میانگین اعضای جدید را محاسبه می‌کنیم و نتیجه را به عنوان مرکز جدید آن خوشه در نظر می‌گیریم. و مجدد پیکسل‌ها را با توجه به مراکز جدید خوشه‌بندی می‌کنیم.

این حلقه را تکرار می‌کنیم تا به مرحله‌ای برسیم که تغییری در خوشه‌ها ایجاد نشود.

درنهایت مقدار هر پیسکل را برابر با مقدار مرکز خوشه مرتبط با آن قرار می‌دهیم.



۳- شرح نتایج

۵-۱-۱

تصویر اصلی:

مولفه hue:

همانطور که در بخش قبل توضیح داده شد، این مولفه طول موج رنگ را مشخص می‌کند. از آنجایی که این مولفه خاصیت زاویه‌ای دارد و مبدا آن رنگ قرمز می‌باشد، در این رنگ گسستگی‌هایی مشاهده می‌شود. به عنوان مثال دو رنگ با h بسیار کم و نزدیک به صفر، مشابه رنگ قرمز با h نزدیک به یک می‌باشد. بنابراین در این تصویر با

اینکه دیوار پشت سر لنا تقریبا یک رنگ می‌باشد اما می-بینیم که بخشی از آن سفید است و بخشی از آن سیاه.

مولفه saturation:

از آنجایی که بیشتر قسمت‌های تصویر متمایل به رنگ قرمزی است که خلوص پایینی دارد، می‌بینیم که این قسمت‌ها به رنگ تیره درآمده‌اند. از ظرفی در قسمت موها، رنگ با خلوص بالاتری داریم.

مولفه intensity:

این تصویر در واقع میزان روشنایی تصویر را مشخص می‌کند. در قسمت‌های روشن‌تر تصویر، نتیجه به رنگ سفید متمایل است و در قسمت‌های تاریک‌تر به رنگ سیاه. از این تصویر می‌توان به عنوان تصویر gray scale از تصویر اصلی نیز استفاده کرد.

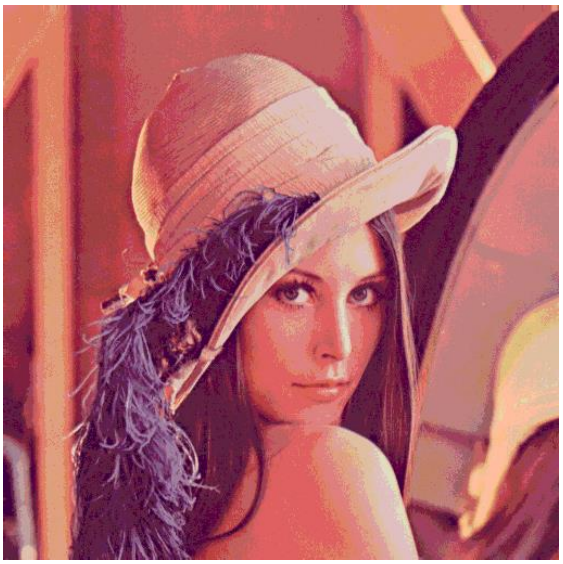
۵-۲-۱

نتیجه گسسته سازی به ۶۴ سطح:

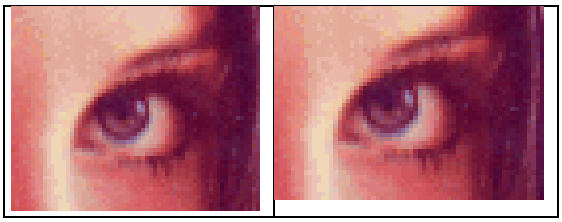
نتیجه گسسته سازی به ۳۲ سطح:

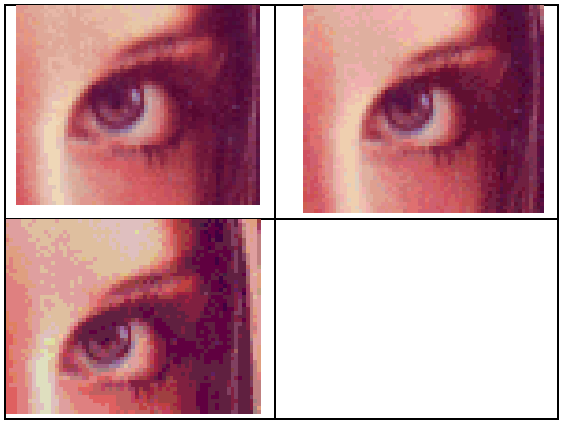
نتیجه گسسته سازی به ۱۶ سطح:

نتیجه گسسته سازی به ۸ سطح:



برای مشاهده بهتر کیفیت تصاویر، آن‌ها را بزرگنمایی کرده‌ام:





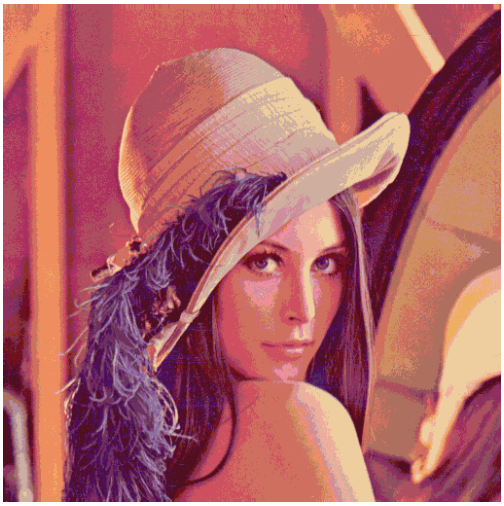
همانطور که مشاهده می‌کنید در این تصویر کاهش تعداد سطوح تا ۱۶ سطح خیلی به کیفیت تصویر آسیب نمی‌زند اما در ۸ سطح شاهد افت کیفیت زیادی هستیم.

	64	32	16	8
mse	1.3744805523504813	5.093944059063991	21.741873254378635	81.5345654686292
psnr	46.749417615168134	41.060261895715314	34.75783401200807	29.0173859985971

در این جدول می‌بینیم که مقدار خطای mse از ۱۶ سطح به پایین بسیار افزایش می‌یابد. و psnr تصاویر هرچه تعداد سطوح کمتر می‌شود، کاهش می‌یابد با این معنی که مقاومت تصویر نسبت به نویز کاهش یافته است.

۵-۲-۲

نتیجه گسسته‌سازی کانال‌های r و g به ۳ بیت و کانال b به ۲ بیت:



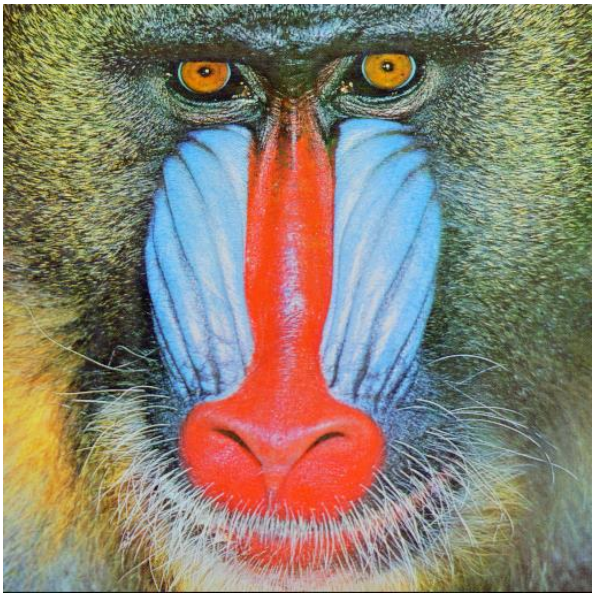
مشابه سوال قبل، در این تصویر با گسسته‌سازی کیفیت تصویر کاهش چشمگیری داشته است. به دلیل اینکه این گسسته‌سازی برای هر سه کانال rgb یکسان نبوده است، شاهد به وجود آمدن رنگ‌های جدید هستیم. به عنوان مثال می‌بینیم که قسمتی از بالای کلاه به رنگ زرد در آمده است. به این دلیل که رنگ زرد در فضای rgb تقریبا مکمل رنگ آبی می‌باشد و بعضی نقاط تصویر رنگ آبی کاهش یافته است که موجب افزایش رنگ زرد می‌شود.

در مقایسه با تصویر اصلی:

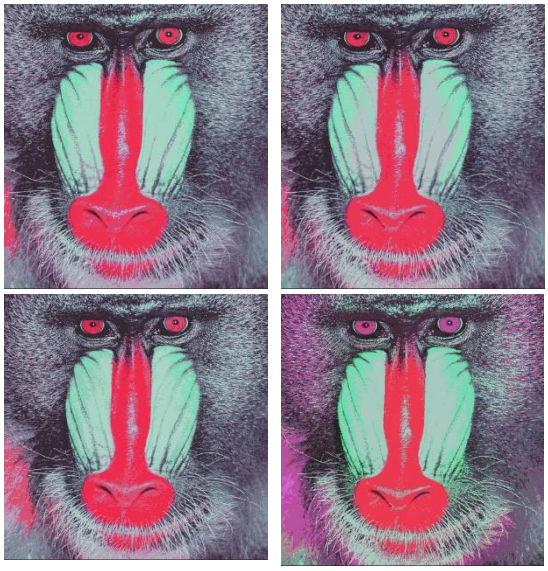
MSE = 26.19

PSNR = 156.16

مشخص است که خطای زیادی در این تصویر وجود دارد.



گسسته‌سازی به ۳۲ رنگ:



می‌توان مشاهده کرد که اکثر جزییات تصویر حفظ شده است. در تصویر پایین سمت چپ احتمالا در seed اولیه چندین مرکز خوشه متمایل به قرمز وجود داشته و باعث شده است که جزییات شامل این رنگ و رنگ‌های مشابه مانند رنگ زرد به سمت قرمز میل کنند. در حالی که در سه تصویر دیگر تمرکز بیشتری بر روی رنگ‌های سبز و آبی بوده است.

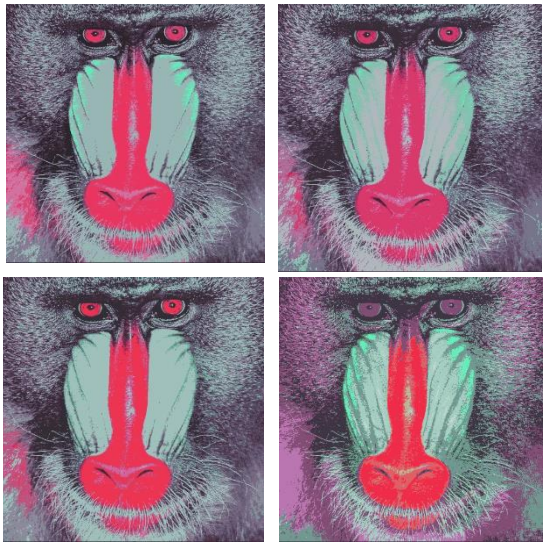
۵-۲-۳

به دلیل اینکه در الگوریتم k-meas نقاط شروع (seed) به صورت رندوم انتخاب می‌شوند، با هربار اجرای الگوریتم تعداد مراحل مورد نیاز تا converge شدن خوشه ها، و همچنین مراکز خوشه‌ها متفاوت خواهند بود. بنابراین در این تمرین، به ازای هر تعداد سطح، تعدادی از خروجی-هایی که طی اجراهای مختلف به وجود آمده‌اند را در ادامه آورده‌ام.

به طور متوسط، الگوریتم ۱۰ مرتبه اجرا می‌شود تا خوشه‌ها تعین شوند.

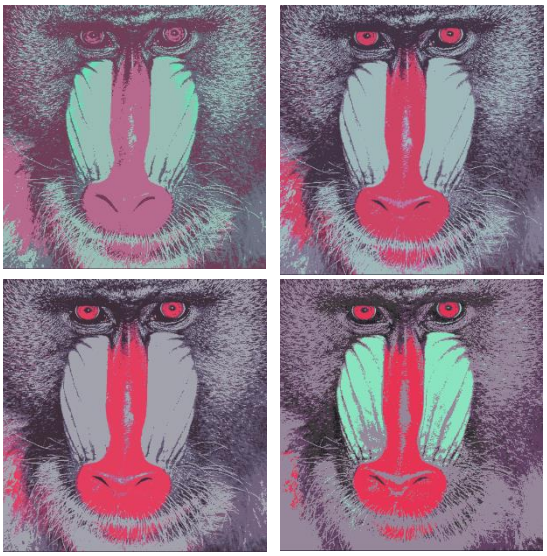
تصویر اصلی:

گسسته‌سازی به ۱۶ رنگ:



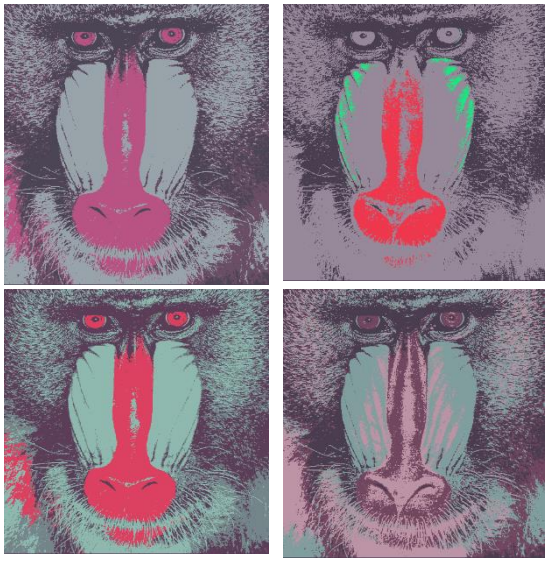
مشابه حالت قبل است، با افت کیفیت بیشتر

گسسته‌سازی به ۸ رنگ:



مشاهده می‌شود که به دلیل استفاده از seed رندوم، نتایج می‌تواند بسیار متفاوت باشد:

گسسته‌سازی به ۴ رنگ:



در این قسمت فقط با استفاده از ۴ رنگ تصویر را بازسازی کرده‌ایم.

با عنوان نمونه، برای یک بار اجرای برنامه mse و psnr به صورت زیر است:

	4	8	16	32
mmse	99.13205464680989	98.74805577596028	95.45009485880534	96.26845677693684
psnr	28.16866253251833	28.185518071930503	28.333039965338745	28.295963509165908

با وجود اینکه میزان خطا بسیار بالا است، اما با کم کردن تعداد سطوح، این مقدار تغییر زیادی نمی‌کند.

از دید انسان واضح است که این روش نسبت به روش گسسته‌سازی بخش ۵-۱-۱ عملکرد به مراتب بهتری دارد.

پیوست

تابع مربوط به نرمال‌سازی تصویر:

```
def normalize(img):
    min = np.min(img)
    max = np.max(img)
    img = (img - min) / (max - min) * 255
    return img
```

تابع مربوط به محاسبه

```

def compute_psnr(im1, im2):
    mse = np.mean((im1 - im2) ** 2)
    if mse == 0:
        return np.inf, 0
    max_pixel = 255.0
    psnr = 20 * np.log10(max_pixel / np.sqrt(mse))
    return psnr, mse
```

۵-۱-۱

```
import copy

import cv2
import numpy as np

from common import normalize


def rgb_to_hsi(img):
    src = copy.deepcopy(img)
    src = src.astype(float)
    src += 1
    r = np.float32(src[:, :, 2])
    g = np.float32(src[:, :, 1])
    b = np.float32(src[:, :, 0])
    theta = np.degrees(
        np.arccos(
            np.divide(
                ((r - g) + (r - b)) / 2,
                np.sqrt(np.power(r - g, 2) + ((r - b) * (g - b))) + 1
            )
        )
    )
    h = theta
    s = np.zeros(h.shape, dtype=float)
    i = (r + g + b) / 3
    for y in range(src.shape[0]):
        for x in range(src.shape[1]):
            s[y, x] = 1 - (3 / (r[y, x] + g[y, x] + b[y, x]) * min([r[y,
x], g[y, x], b[y, x]]))
            if b[y, x] > g[y, x]:
```

```
                h[y, x] = 360 - theta[y, x]
    return h, s, i


img = cv2.imread('Lena.bmp')
h, s, i = rgb_to_hsi(img)

h = normalize(h)
s = normalize(s)
i = normalize(i)

cv2.imwrite('5-1-1/h.png', h)
cv2.imwrite('5-1-1/s.png', s)
cv2.imwrite('5-1-1/i.png', i)
```

۵-۲-۱

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

from common import compute_psnr


def quantize(img, level):
    d = 255 / level
    return np.rint(img / d) * d


img = cv2.imread('Lena.bmp')
levels = [64, 32, 16, 8]
PSNRs = ["psnr"]
MSEs = ["mse"]
for level in levels:
    quantized = quantize(img, level)
    cv2.imwrite('5-2-1/{}-level.png'.format(level), quantized)
    psnr, immse = compute_psnr(img, quantized)
    PSNRs.append(psnr)
    MSEs.append(immse)
data = [MSEs, PSNRs]
labels = (" ",) + tuple(levels)
fig, ax = plt.subplots(dpi=200, figsize=(6, 1))
ax.axis('off')
ax.table(colLabels=labels, cellText=data, loc='center')

fig.savefig('5-2-1/result.png')
```

۵-۲-۲

```
import cv2
import numpy as np

from common import compute_psnr
```

```
def quantize(img, levelB, levelG, levelR):
    dr = 255 / levelR
    dg = 255 / levelG
    db = 255 / levelB
    quantized_r = (np.rint(img[:, :, 2] / dr) * dr)
    quantized_g = (np.rint(img[:, :, 1] / dg) * dg)
    quantized_b = (np.rint(img[:, :, 0] / db) * db)
    res = np.zeros(img.shape)
    res[:, :, 0] = quantized_b
    res[:, :, 1] = quantized_g
    res[:, :, 2] = quantized_r
    return res


img = cv2.imread('Lena.bmp')
quantized = quantize(img, 4, 8, 8)
cv2.imwrite('5-2-2/result.jpg', quantized)
psnr, mse = compute_psnr(img, quantized)
print("psnr:", str(psnr))
print("mse:", str(mse))
```

۵-۲-۳

```
import copy

import cv2
import numpy as np
import matplotlib.pyplot as plt
import random

from common import compute_psnr


def cluster(img, levels):
    cluster_centers = []
    cluster_items = []
    for i in range(levels):
        cluster_items.append([])
    img_clusters = np.zeros((img.shape[0], img.shape[1]))
    for i in range(levels):
        #             blue                  green                    red
        t = (random.randint(0, 255), random.randint(0, 255),
random.randint(0, 255))
        cluster_centers.append(t)

    any_cluster_updated = True
    counter = 0
    while counter<0:
        counter += 1
        print(str(counter))
        any_cluster_updated = False

        for i in range(img.shape[0]):
            for j in range(img.shape[1]):
                min_distance = 999999
                min_distance_cluster = 0
                for cluster_index in range(len(cluster_centers)):
                    distance = np.sqrt(
```

```
                        ((img[i, j, 0] - cluster_centers[cluster_index][0])
** 2) +
                        ((img[i, j, 1] - cluster_centers[cluster_index][1])
** 2) +
                        ((img[i, j, 2] - cluster_centers[cluster_index][2])
** 2))
                    if distance < min_distance:
                        min_distance = distance
                        min_distance_cluster = cluster_index
                if img_clusters[i, j] != min_distance_cluster:
                    img_clusters[i, j] = min_distance_cluster
                    any_cluster_updated = True
                    cluster_items[min_distance_cluster].append(img[i, j,
:])
        if not any_cluster_updated:
            break
        for cluster_index in range(len(cluster_items)):
            sum_rgb = [0, 0, 0]
            for item in cluster_items[cluster_index]:
                sum_rgb += item
            if len(cluster_items[cluster_index]) > 0:
                new_rgb = np.rint(sum_rgb /
len(cluster_items[cluster_index]))
                cluster_centers[cluster_index] = (new_rgb[1], new_rgb[0],
new_rgb[2])
    return cluster_centers, img_clusters

levels = [4, 8, 16, 32]
img = cv2.imread('Baboon.bmp')

PSNRs = ["psnr"]
MSEs = ["mmse"]
for level in levels:
    print("********** {} **********".format(level))
    new_img = copy.deepcopy(img)
    kernel_mat, clusters = cluster(new_img, level)
    for i in range(new_img.shape[0]):
        for j in range(new_img.shape[1]):
            new_img[i, j, :] = kernel_mat[int(clusters[i, j])]
    cv2.imwrite('5-2-3/res{}.png'.format(level), new_img)

    res = cv2.imread('5-2-3/res{}.png'.format(level))
    psnr, mmse = compute_psnr(img, res)
    MSEs.append(mmse)
    PSNRs.append(psnr)

data = [MSEs, PSNRs]
labels = (" ",) + tuple(levels)
fig, ax = plt.subplots(dpi=200, figsize=(7, 1))
ax.axis('off')
ax.table(colLabels=labels, cellText=data, loc='center')

fig.savefig('5-2-3/result.jpg')
```