

# تصاویر رنگی

## سحر شیخ الاسلامی

چکیده

اطلاعات گزارش

با افزایش استفاده از وسایل دیجیتال، امروزه با حجم باور نکردنی از عکس روی رو هستیم. عموم عکس های مورد استفاده انسان نیز عکس های رنگی است. استفاده و انجام عملیات بر روی عکس های رنگی (به دلیل حجم زیادتر آنها) بار محاسباتی زیادی دارد. در این گزارش به بررسی چند عملیات مقدماتی بر روی تصاویر رنگی می پردازیم.

تاریخ: ۵/۱۰/۱۳۹۹

واژگان کلیدی:

Color space

HSI

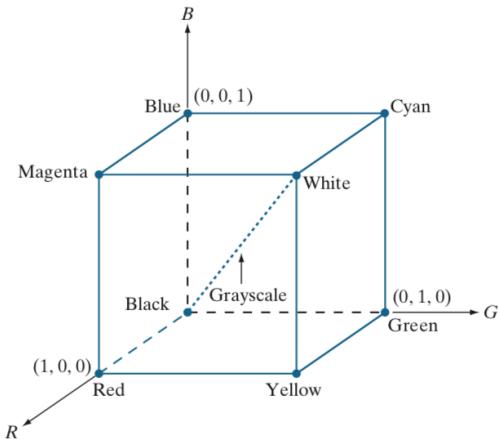
RGB

YUV

YIQ

Quantization

Kmeans



شکل ۱.۱: تصویر از مکعب RGB

شکل ۱.۱ این مختصات را نشان می دهد، که در آن مقادیر اولیه RGB در سه گوشه قرار دارند. رنگهای ثانویه فیروزه ای، سرخابی و زرد در سه گوشه دیگر قرار دارند. رنگ سیاه در مبدأ است. و سفید در دورترین گوشه از مبدأ قرار دارد. در این مدل، سطوح خاکستری ( نقاط با مقادیر برابر (RGB) از سیاه به سفید در امتداد خطی که از این دو نقطه می گذرد ، گسترش می یابد. رنگ های مختلف در این مدل نقاط روی مکعب یا داخل آن است و توسط بردارهای از مبدأ تعریف می شود.

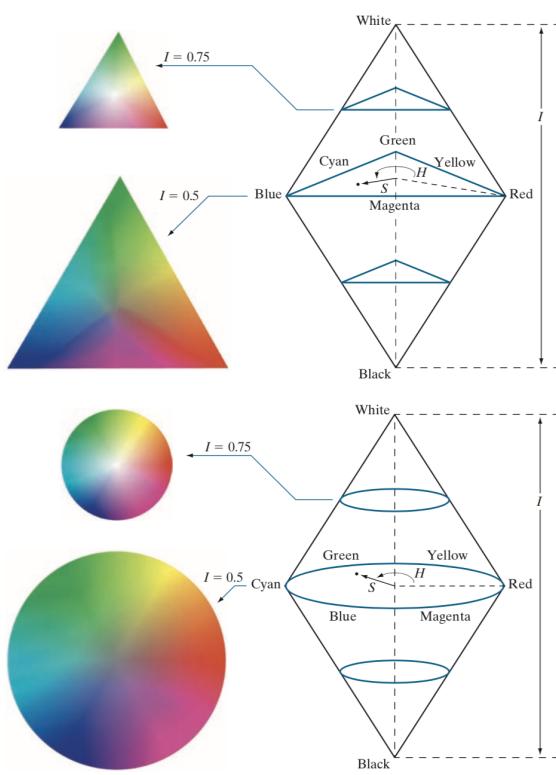
تصاویر نشان داده شده در مدل رنگی RGB از سه مولفه تشکیل شده است، هر رنگ اصلی یک مولفه است. در وسیله های الکترونیکی، هر پیکسل از سه مولفه RGB کنار هم

امروز بیشتر عکس های مورد استفاده، عکس های رنگی هستند. استفاده از عکس های رنگی دو دلیل دارد. دلیل اول این است که این است که رنگ ابزار قدرتمندی برای توصیف است که خیلی از موارد تشخیص و استخراج اجسام را به شدت ساده می کند. به همین دلیل انجام عملیات detection در فضای رنگی ساده تر است. دلیل دیگر به توانایی مغز و چشم انسان در تشخیص رنگ ها است. انسان می تواند هزاران رنگ متفاوت را تشخیص دهد در حالیکه فقط حدود صد سطح خاکستری را می تواند تشخیص دهد. برای به دست آوردن عکس رنگی، دو روش متقابل کاربرد دارد، استفاده از سنسور های رنگی و رنگی کردن عکس های سیاه-سفید. همانطور که عملیات مختلفی را بر روی عکس های سیاه و سفید انجام دادیم، میتوانیم عملیات متفاوتی را بر روی عکس های رنگی نیز بررسی کنیم.

## ۲- شرح تکنیکال

### ۲.۱.۱: قسمت ۵

ابتدا فضای رنگی RGB و فضای رنگی HSI را بررسی می کنیم. سپس در ادامه روش تبدیل این دو را بررسی می کنیم. در مدل RGB ، هر رنگ در اجزای اصلی طیفی قرمز ، سبز و آبی ظاهر می شود. این مدل مبتنی بر سیستم مختصات دکارتی است.



شکل ۱.۲: مدل رنگی HSI بر اساس مثلثی و دایره ای

برای تبدیل از فضای RGB به شکل زیر عمل می کنیم.

#### بدست آوردن : Hue

Hue نشان دهنده شباهت یا تفاوت یک رنگ از یکی از عوامل اصلی، آبی، قرمز، زرد و سبز است. برای بدست آوردن این شباهت ابتدا با زاویه را از فرمول زیر بدست آوریم:

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

پس از بدست آوردن زاویه، حال hue مقداری برابر مقدار پایین دارد.

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

#### بدست آوردن saturation

تشکیل شده است که هر کدام سطح روشنایی متناسب با مقدار این مولفه ها دارند. در نهایت چشم ما اما این پیکسل را به شکل پیوسته و یک رنگ می بیند. به تعداد بیت های استفاده شده برای نمایش هر پیکسل در فضای RGB، عمق پیکسل گفته می شود. یک تصویر RGB را در نظر بگیرید که در آن هر یک از تصاویر قرمز، سبز و آبی یک تصویر 8 بیتی است. در این شرایط، هر پیکسل رنگی RGB [یعنی یک سه گانه مقادیر (R, G, B)] دارای عمق 24 بیت است.

اصطلاح تصویر تمام رنگی اغلب برای نشان دادن یک تصویر رنگی 24 بیتی RGB استفاده می شود. تعداد کل رنگهای ممکن در یک تصویر RGB تمام رنگی حدودا ۱۶ میلیون است.

سیستم رنگی RGB به طور ایده آل برای پیاده سازی سخت افزار مناسب است. علاوه بر این، سیستم RGB با این واقعیت که چشم انسان به طور ادراکی حساسیت زیادی به قرمز و آبی و سبز دارد تطابق کامل دارد. متأسفانه CMY.RGB و سایر مدلها رنگی مشابه برای توصیف رنگها برای تفسیر انسان از نظر کاربردی مناسب نیستند.

به طور مثال اگر سه عدد که هر کدام مقدار RGB را نشان می دهد را به شما بگوییم، احتمالاً ایده ای از رنگی نهایی نخواهید داشت.

هنگامی که ما یک شی رنگی را مشاهده می کنیم، ما آن را با رنگ، اشباع و روشنایی توصیف می کنیم. می دانیم که hue یک ویژگی رنگی است که یک رنگ خالص (زرد، نارنجی یا قرمز خالص) را توصیف می کند، در حالی که اشباع میزان رقیق شدن یک رنگ خالص توسط نور سفید را نشان می دهد. روشنایی یک توصیف گر ذهنی است که اندازه گیری آن عملاً غیرممکن است. این مفهوم آکرمواتیک شدت است و یکی از عوامل کلیدی در توصیف احساس رنگ است. ما می دانیم که شدت (سطح خاکستری) مفیدترین توصیف کننده تصاویر آکرمواتیک است. این مقدار قطعاً قابل اندازه گیری است و به راحتی قابل تفسیر است. مدلی که ما در حال ارائه آن هستیم، مدل رنگی HSI (رنگ، اشباع، شدت) نامیده می شود.

در یک تصویر رنگی، intensity شدت را از اطلاعات حامل رنگ (رنگ و اشباع) جدا می کند. در نتیجه، مدل HSI ابزاری مفید برای توسعه الگوریتم های پردازش تصویر است.

این مولفه بافت تصویر را عوض نمی کند و فقط عکس را سیاه سفید می کند.

برای بدست آوردن  $Y$  باید از فرمول زیر استفاده کنیم:

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

Saturation یعنی هر رنگ با نور سفید کمتری ترکیب شده باشد.

saturation بیشتری می باشد.

برای بدست آوردن saturation از رابطه زیر استفاده می کنیم.

**U:** مولفه  $U$

این مولفه و مولفه  $Y$  دیگر برای نشان دادن Blue projection بیان شده اند.  $U$  که نشان دهنده  $I$  می باشد از رابطه های زیر بدست می آید:

$$U = -0.147R - 0.289G + 0.436B$$

همچنین می توان  $U$  را از رابطه زیر بدست آورد.

$$U = B - Y$$

**V:** مولفه  $V$

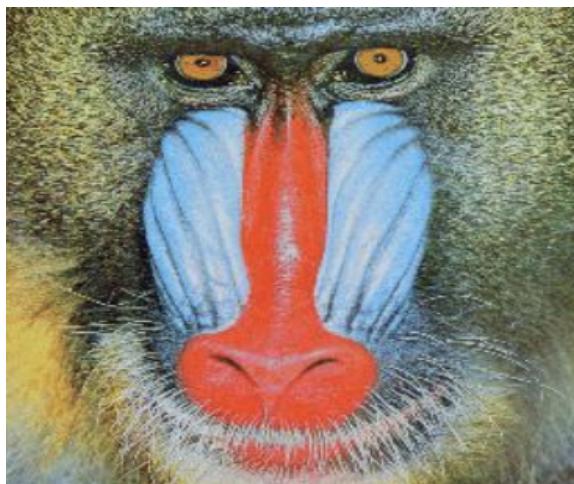
این مولفه همانند  $U$  برای نشان دادن chrominance استفاده می شود و نشان دهنده  $I$  Red projection می باشد. برای بدست آوردن  $V$  از روی تصویر RGB دو فرمول زیر استفاده می شود:

$$V = 0.615R - 0.515G - 0.100B$$

همچنین می توان  $V$  را از رابطه زیر بدست آورد.

$$V = R - Y$$

در ادامه یک تصویر را در این سه حوزه نشان می دهیم:



شکل ۱.۳: تصویر اصلی

بدست آوردن **intensity**

Intensity در واقع شدت رنگی یک پیکسل را نشان می دهد.

برای بدست آوردن intensity از رابطه زیر استفاده می کنیم.

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

در قسمت نتایج بر روی هر کدام از این مولفه ها بیشتر بحث می کنیم.

#### ۵.۱.۲: قسمت

در قسمت قبل با دو تا از محبوب ترین مدل های رنگی در دنیای دیجیتال آشنا شدیم. در این قسمت به بررسی سه مدل رنگی دیگر می پردازیم:

Mdl YUV, YUV (به علت شباهت این دو مدل را با هم بیان میکنیم)

این مدلها از مدل های رنگی هستند که در ویدیو استفاده می شود. در روش های آنالوگ کدگذاری تلویزیون Chrominance (درخشندگی) از اطلاعات Luminance (رنگ) جدا می شود.

از مدل های مبتنی بر این روش می توان به YUV و YIQ و YCbCr و

و اشاره کرد. سیگنال های تلویزیون آمریکای شمالی و ژاپن در فضای YIQ و سیگنال های آنالوگ در اروپا در فضای YUV است. کد گذاری های نوارهای ویدیوی VHS نیز از فن آوری YIQ استفاده می کردند. این دو مدل

رنگی (YUV و YIQ) بسیار شبیه به هم هستند.

همچنین YCbCr در فشرده سازی تصویر JPEG و فشرده سازی ویدیوی MPEG استفاده شده است.

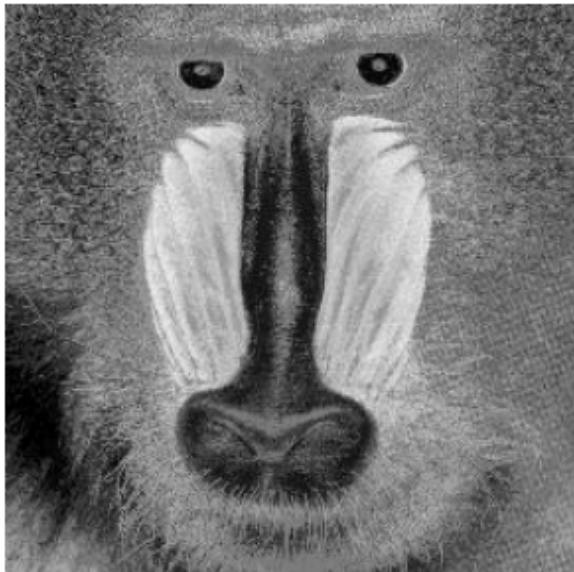
حال به بررسی سه مولفه  $Y$  این فضای رنگی می پردازیم:

**Y:** مولفه  $Y$

این مولفه همانند مولفه  $I$  در مدل رنگی HSI است. این

مولفه نشان دهنده درخشندگی (Luminance) است.

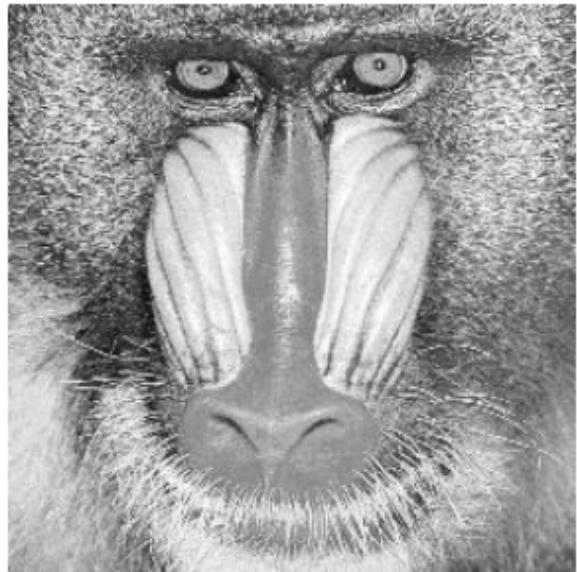
در واقع این مولفه از میانگین وزن دار سه مقدار RGB بدست می آید و تصویر سیاه سفید را نشون می دهد.



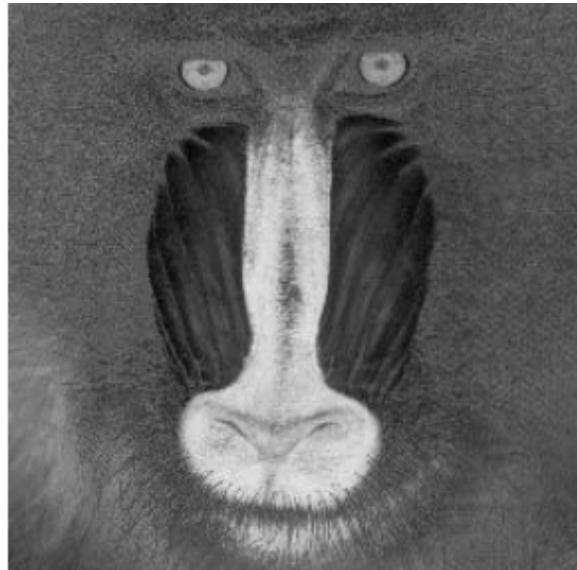
شکل ۱.۶: مولفه  $V$  شکل ۱.۳ در مختصات YUV

همانطور که توضیح دادیم،  $Y$  میانگینی از عکس است و بافت را حفظ میکند،  $V$  تقریبا نشان دهنده رنگ آبی است و در مناطق با حجم بالای آبی، مقدار بزرگتری دارد و سفید تر است، در حالیکه  $U$  نشان دهنده رنگ قرمز است.

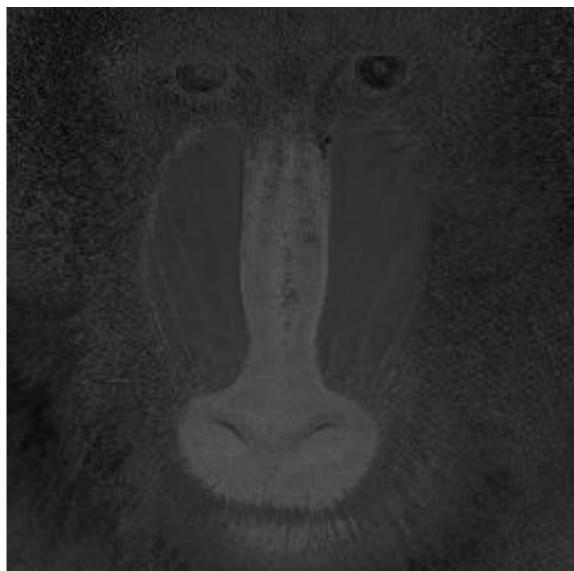
مدل  $YIQ$  نیز بسیار شباهت به این مدل دارد. تنها تفاوت آن این است که اگر  $V$  و  $U$  را محور های مختصات در نظر بگیریم،  $I$  و  $Q$  دوران داده شده همان  $U$  و  $V$  هستند.(به اندازی ۳۳ درجه چرخیده اند) مولفه  $Y$  نیز مشابه است.



شکل ۱.۴: مولفه  $Y$  شکل ۱.۳ در مختصات YUV



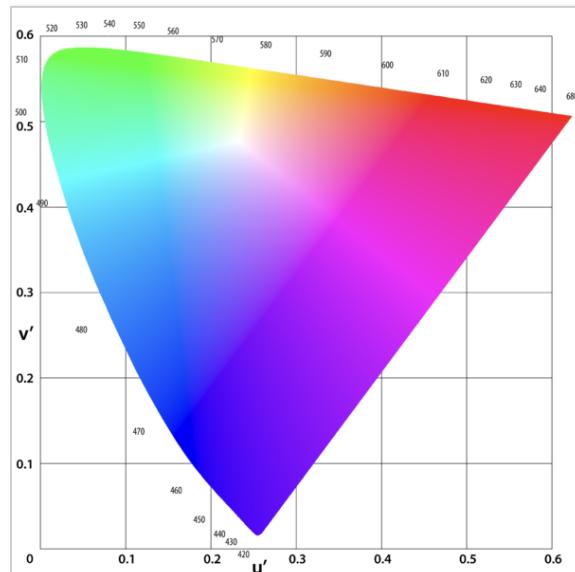
شکل ۱.۵: مولفه  $U$  شکل ۱.۳ در مختصات YUV



شکل ۱.۷: مولفه  $I$  شکل ۱.۳ در مختصات YIQ

که در آن ظاهر یک رنگ با اندازه گیری فیزیکی مربوط به منبع محرك مطابقت ندارد.

رنگ از ذهن ناظر سرچشمه می‌گيرد و فقط توزيع نيروى طيفي نورى است که با چشم روبرو می شود. از اين نظر، هر درك رنگي، ذهنی است. با اين حال ، تلاش های موفقیت آمیزی برای ترسیم توزیع قدرت طیفی نور به واکنش حسی انسان به روشی قابل اندازه گیری صورت گرفته است. در سال 1931 ، با استفاده از اندازه گیری های روانی ، کمیسیون بین المللی روشنایی (CIE) فضای رنگی XYZ را ایجاد کرد که بینایی رنگی انسان را با موفقیت در این سطح حسی اساسی مدل می کند.



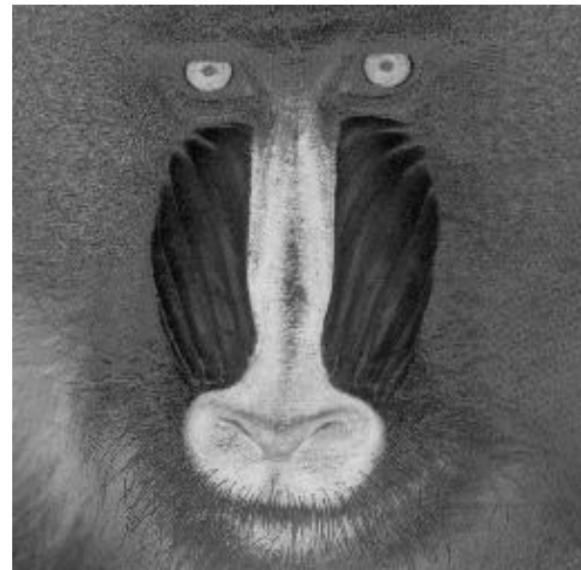
شکل ۱.۹: فضای رنگی CAM

فضای رنگی، مبتنی بر XYZ است. برای تبدیل از RGB به XYZ از رابطه زیر استفاده می کنیم؛

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.4124564 & 0.3575761 & 0.1804375 \\ 0.2126729 & 0.7151522 & 0.0721750 \\ 0.0193339 & 0.1191920 & 0.9503041 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

همچنین برای تبدیل بر عکس از رابطه زیر استفاده می کنیم:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 3.2404542 & -1.5371385 & -0.4985314 \\ -0.9692660 & 1.8760108 & 0.0415560 \\ 0.0556434 & -0.2040259 & 1.0572252 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$



شکل ۱.۸: مولفه‌ی Q شکل ۱.۳ در مختصات YIQ.

در مدل YIQ سه مولفه به شرح زیر داریم:

- I که معرف in-phase هست
- Q که معرف Quadrate هست.
- Y که نمایش دهنده Luma یا روشنایی تصویر است.

برای تبدیل از فضای RGB به این فضا کافی است که از رابطه زیر استفاده کنیم:

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} \approx \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.5959 & -0.2746 & -0.3213 \\ 0.2115 & -0.5227 & 0.3112 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

علاوه بر رابطه خطی بالا، از رابطه های زیر نیز می توان استفاده کرد:

$$I = 0.492111(R - Y) \cos 33^\circ - 0.877283(B - Y) \sin 33^\circ$$

$$Q = 0.492111(R - Y) \sin 33^\circ + 0.877283(B - Y) \cos 33^\circ$$

همچنین برای عکس این قضیه، و تبدیل از فضای رنگی YQI به RGB می توان از ماتریس زیر استفاده کرد:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0.956 & 0.619 \\ 1 & -0.272 & -0.647 \\ 1 & -1.106 & 1.703 \end{bmatrix} \begin{bmatrix} Y \\ I \\ Q \end{bmatrix}$$

**:CAM فضای رنگی**  
برخلاف RGB که نشان دهنده عملیات وسائل دیجیتال است، این فضای رنگی بسیار شبیه به درک انسان ساخته شده است.

((CAM)) یک مدل ریاضی است که سعی در توصیف جنبه های ادراکی بینایی رنگ انسان دارد ، یعنی شرایط مشاهده

## قسمت ۵.۲.۱:

در قسمت ۵.۱.۱ توضیح دادیم که ذخیره سازی و انجام عملیات مختلف بر روی تصویر رنگی به علت حجم بیشتر آن، بسیار زمانگیر است. هر پیکسل در تصویر سیاه، سفید به سه برابر شده است و در واقع حجم اندازه‌ی تصویر سه برابر می‌شود.

برای حل این مشکل روش‌های متفاوتی را پیشنهاد داده اند، که ادامه یکی از این روش‌ها را به سه رنگ قرمز، آبی و سبز نشان می‌دهیم و برای نشان دادن هر کدام از این ها از ۸ بیت استفاده می‌کنیم، ۲۴ بیت به ازای پیکسل. می‌دانیم که در RGB معمول، هر خانه را به سه رنگ قرمز، آبی و سبز نشان می‌دهیم و برای نشان دادن هر کدام از این توان این مقدار بیت را کاهش داد.

مثالاً می‌توان به جای استفاده از ۲۴ بیت به ازای هر پیکسل از ۱۲ بیت (۴ بیت برای هر رنگ) یا از ۶ رنگ (۲ بیت به ازای هر رنگ) استفاده کرد. همانطور که قابل مشاهده است این عملیات میتواند سایز و حجم عکس را تا حد قابل توجهی کاهش دهد. طبیعی است که با انجام این عملیات، رنگ اعداد قابل نمایش نیز کاهش یافته، بنابراین رنگ‌های موجود نیز کاهش می‌یابد، که این تاثیر واضحی بر روی کیفیت عکس دارد. بنابراین با کاهش سایز عکس، کیفیت تصویر نیز کاهش می‌یابد.

به طور مثال، با ۸ بیت برای هر رنگ توانایی نشان دادن ۲۵۶ مقدار برای آن رنگ و در کل حدود ۱۶ میلیون رنگ خواهیم داشت، اگر این مقدار را به ۶ کاهش دهیم، به جای ۲۵۶ سطح رنگی مختلف برای RGB حال ۶۴ سطح خاکستری مختلف خواهیم داشت.

در این قسمت از ما خواسته شده که ابتدا عکس را با سطح خاکستری مختلف نشان دهیم (۶۴ سطح ۶ بیت) و سطح (۵ بیت) و ... و سپس بررسی کنیم که تصویر تا چه حد با تصویر اصلی تفاوت خواهد داشت. ابتدا الگوریتم برای کاهش سطح خاکستری را بررسی می‌کنیم:

می‌دانیم که در تصویر اصلی به ازای هر رنگ (RGB) ۸ بیت و بنابراین ۶۴ سطح خاکستری وجود دارد. اگر بخواهیم این سطح را به طور مثال به ۶۴ تبدیل کنیم، ابتدا باید level را بدست آوریم، از تقسیم ۲۵۶ بر تعداد سطوح خاکستری دلخواه‌مون بدست می‌آید (برای ۶۴ برابر ۴ می‌شود در واقع یعنی هر ۴ مقدار را حال باید یک مقدار در نظر بگیریم). حال کافی است که از فرمول پایین برای هر رنگ استفاده کنیم.

$$\lfloor \frac{\text{value}}{\text{level}} \rfloor * \text{level}$$

یعنی ابتدا کف مقدار تقسیم را بدست می‌آوریم، و سپس در همان مقدار ضرب می‌کنیم. واضح است که همه سطوح خاکستری الان بر level بخش پذیر هستند که یعنی بیت‌های کم ارزش مقدار صفر دارند، پس به همین دلیل می‌توان به اندازه level (log level) از بیت‌ها کم کرد.

حال کافی است که عملیات بالا را بر روی هر سه کanal رنگی انجام دهیم.

در ادامه خواسته شده است که با دو معیار MSE و PSNR میزان شباهت با عکس اصلی را توضیح دهیم. این دو معیار را توضیح می‌دهیم:

### MSE:(Mean Square Error)

این تابع دو ماتریس (در اینجا عکس) را مقایسه می‌کند. به ازای هر خانه از ماتریس، مربع تفاوت دو ماتریس را بدست می‌ورد. و میانگین این تفاصل را به عنوان خروجی بر میگرداند. بنابراین هر چه عدد بزرگتر باشد، نشانه‌ی تفاوت بیشتر دو ماتریس است.

### معیار PSNR(Peak Signal Noise Ratio):

این معیار مقدار سیگنال به نویز را با داشتن سیگنال اصلی محاسبه می‌کند. بر خلاف معیار قبلی، هر چی مقدار PSNR بیشتر باشد، تصویر به تصویر اصلی شبیه‌تر بوده و مقدار نویز آن کمتر است.

برای محاسبه PSNR از رابطه‌ی زیر استفاده می‌کنیم.

$$\begin{aligned} PSNR &= 10 \cdot \log_{10} \left( \frac{MAX_I^2}{MSE} \right) \\ &= 20 \cdot \log_{10} \left( \frac{MAX_I}{\sqrt{MSE}} \right) \\ &= 20 \cdot \log_{10}(MAX_I) - 10 \cdot \log_{10}(MSE) \end{aligned}$$

نتیجه را در قسمت نتایج به تفصیل بررسی خواهیم کرد.

## قسمت ۵.۲.۲:

این قسمت بسیار مشابه قسمت قبل می‌باشد، با این تفاوت که در قسمت قبل همه‌ی کانال‌های رنگی را به تعداد بیت‌های یکسان کاهش میدادیم، اما در این قسمت از ما خواسته شده است که کانال فرمز و سبز را به سه بیت (۸ مقدار متفاوت) و کانال آبی را به ۲ بیت (۴ مقدار متفاوت) کاهش دهیم. مشابه قسمت قبل عمل می‌کنیم.

نتیجه را در قسمت نتایج به تفصیل بررسی خواهیم کرد.

## قسمت ۵.۲.۳:

توضیح دادیم که در عکس‌های دیجیتال بعضاً تا ۱۶ میلیون رنگ به کار رفته است. استفاده از این تعداد رنگ در برخی کاربردها عملاً غیر ممکن است، مثلاً برای بافت فرش ۱۶ میلیون نخ با رنگ متفاوت را نمی‌توان تهیه کرد.

به همین دلیل باید تعداد رنگ‌ها را کاهش دهیم (همانطور که در قسمت‌های قبل اشاره شد). در این کاهش رنگ باید حواسمن باشد که ترکیب و بافت عکس کمترین تخریب را داشته باشد و خروجی نزدیکترین به عکس اصلی دیجیتال باشد.

برای این کار دو راهکار پیشنهاد می‌دهیم و نتیجه را در قسمت نتایج بررسی می‌کنیم.

راه اول: مشابه قسمت قبل:

فرض کنیم که در نهایت می‌خواهیم  $k^8$  رنگ داشته باشیم.

حال‌های مختلف بیت برای RGB را در نظر می‌گیریم که تعداد بیت‌ها در نهایت  $k$  شود. مثلاً  $n$  بیت برای آبی و  $m$  بیت برای قرمز و  $k-m-n$  بیت برای سبز. به ازای تمام  $n$  و

$m$  های ممکن این کار را انجام می دهیم، و بهترین نتیجه را بر می گردانیم.

راه دوم: با استفاده از الگوریتم های یادگیری ماشین:

الگوریتم kmeans یک الگوریتم یادگیری ماشین است که در دسته الگوریتم های بدون ناظر قرار میگیرد. این الگوریتم در مرحله اول نقاطی رندوم را به عنوان مرکز دسته انتخاب می کند. سپس در هر مرحله هر نقطه را به دسته ای که مرکز آن نزدیکترین فاصله را دارد تخصیص می دهد. این عملیات را آنچا انجام می دهد که پاسخ به یک جواب همگرا شود و دیگر تغییری نکند.

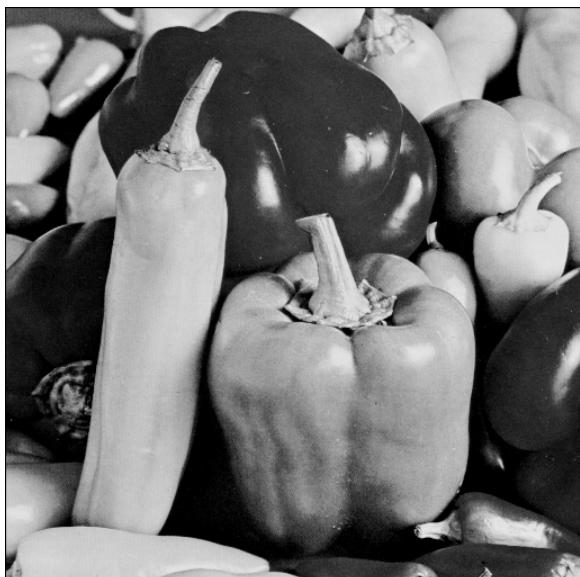
برای پیدا کردن فاصله معمولاً این الگوریتم از  $2L$  استفاده می کند که فاصله اقلیدسی است.

در این مسیله می خواهیم از این الگوریتم استفاده کنیم. همه رنگ های به کار رفته در تصویر را در نظر میگیرم. حال روی این بردار ها (هر بردار از سه عضو RGB تشکیل شده است) دسته بندی را انجام می دهد. به علت اینکه فاصله اندازه گیری این الگوریتم  $2L$  است بهتر است از مدل رنگی  $L^*a^*b$  استفاده کنیم که یک مدل رنگی است که فضای اقلیدسی دارد.

حال به تعداد رنگ های مورد دسته تولید می کنیم و مرکز دسته را برابر میانگین تمام عناصر آن دسته قرار می دهیم. پس از انجام دسته بندی با استفاده از الگوریتم k means حال هر پیکسل تصویر را با مقدار مرکز دسته آن پیکسل جا به جا می کنیم. این روش از روش قبلی ذکر شده بسیار بهتر است. نتیجه را در نتایج بررسی خواهیم کرد.

**نتایج:**  
**قسمت ۵.۱.۱:**

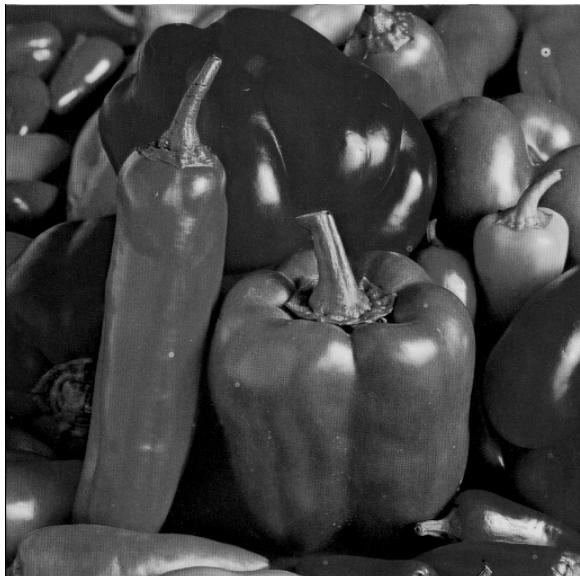
همانطور که در شرح توضیح دادیم RGB و HSI دو مدل رنگی متفاوت است.



شکل ۲.۳: مولفه‌ی سبز تصویر اصلی



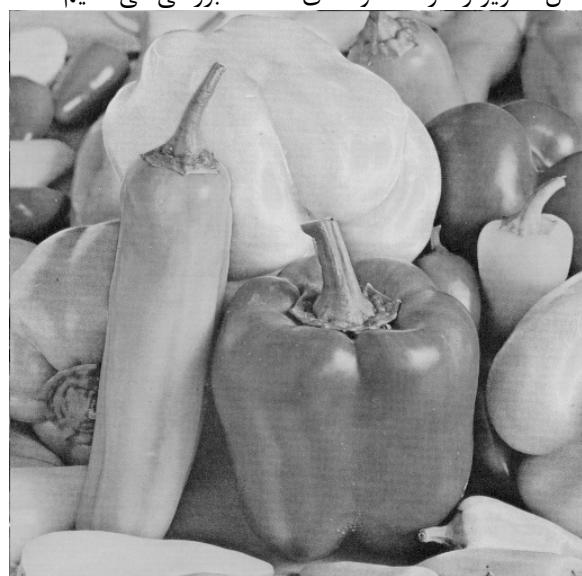
شکل ۲.۱: تصویر اصلی



شکل ۲.۴: مولفه‌ی آبی تصویر اصلی

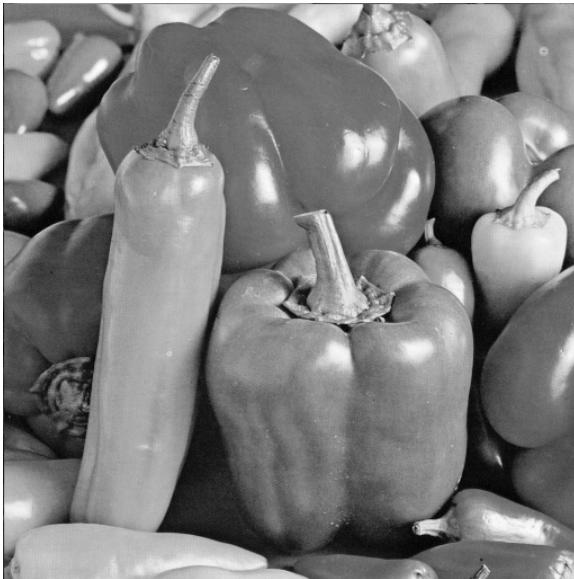
همانطور که در تصویر اصلی مشاهده می‌شود، بیشتر عکس سبز و قرمز هستند، به همین دلیل این دو مولفه مقادیر بیشتر و نزدیک به ۲۵۵ دارند و بنابراین روشتر هستند. در حالیکه آبی خیلی کم به کار رفته و مقادیری نزدیک به صفر دارد، به همین دلیل عکس مولفه‌ی آبی تیره‌تر است.

حال به بررسی مولفه‌های HSI می‌پردازیم.



شکل ۲.۲: مولفه‌ی قرمز تصویر اصلی

در شرح ذکر کردیم که میزان خلوص رنگ را نشان می دهد. در واقع هر چه رنگ با نور سفید بیشتر تر کیب شده باشد، saturation آن مقدار کمتری دارد. مشاهده این امر بسیار ساده است، در قسمت هایی که بر روی فلفل دلمه ای نور افتاده است، در saturation سیاه شده است. همچنین قسمت های خالصتر در شکل ۲.۱ نیز در مقادیر بالایی دارند و سفید هستند.

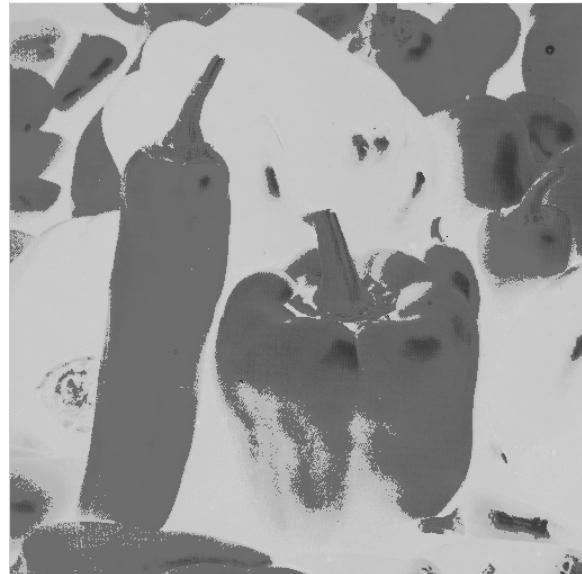


شکل ۲.۷: مولفه‌ی intensity شکل ۲.۱

در شرح گفتیم که معیاری برای اندازه گیری شدت رنگ تصویر است، از میانگین RGB بدست می آید. در واقع intensity همان تصویر سیاه سفید است. شکل ۲.۷ نیز بافت تصویر را به خوبی حفظ کرده و تقریباً تمامی ویژگی های تصویر اصلی را حفظ کرده است.

#### قسمت ۵.۲.۲:

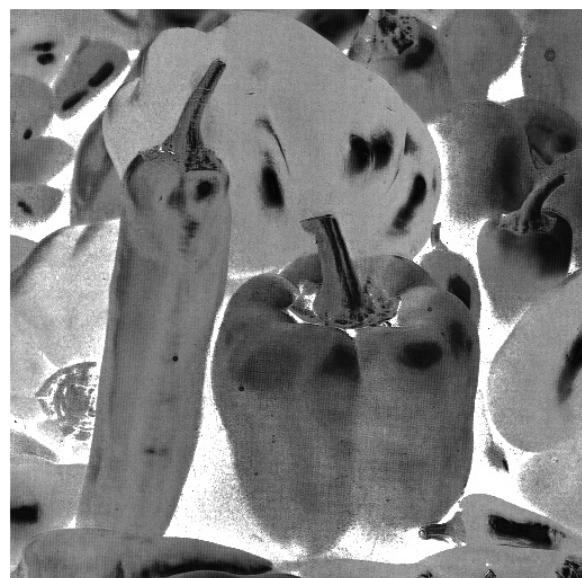
با مقایسه عکس ها می بینیم که در عکس با ۸ سطح کاهش کیفیتی داشته ایم، اما عکس هنوز قابل تشخیص است و فقط کمی از جزئیات آن کم شده است. عکس با ۱۶ سطح نسبت به ۸ کیفیت بهتری دارد(فلفل پاپریکای ایستاده را مقایسه کنید) و به عنوان عکس با کیفیت قابل قبول میتوان پذیرفت. عکس با ۳۲ و ۶۴ اما کیفیت بسیار قابل قبولی دارند(در عین کاهش عده حجم عکس).



شکل ۲.۵: مولفه‌ی hue شکل ۲.۱

همانطور که در شرح توضیح دادیم، hue مقدار نزدیکی رنگ به یکی از رنگ های اصلی را نشان می دهد. هر چی hue مقدار بیشتری داشته باشد، یعنی رنگ به رنگ اصلی نزدیکتر است. همانطور که مشاهده می شود، فلفل های دلمه ای قرمز، رنگی بسیار نزدیک به قرمز خام دارد، به همین دلیل مقدار آن بسیار بالاست. در حالیکه فلفل های سبز، روشنتر هستند، به همین دلیل از رنگ اصلی فاصله بیشتری دارند و hue آنها کمتر است که یعنی ان قسمت عکس تیره تر است.

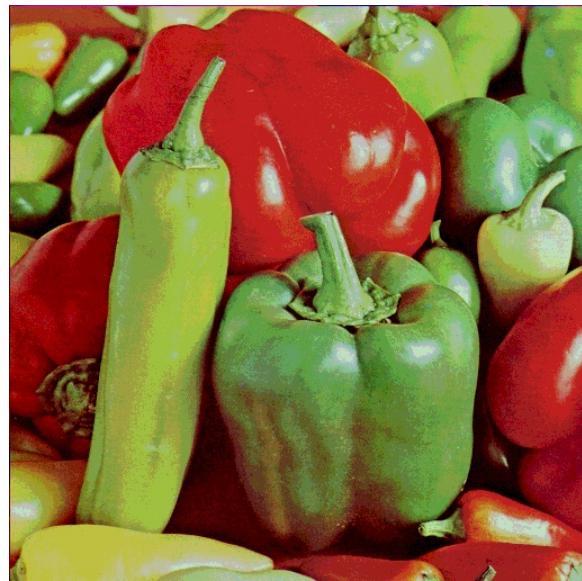
بنابراین مشاهده کردیم که hue با بررسی میزان نزدیکی به رنگ های اصلی، حدوداً رنگ را مشخص می کند.



شکل ۲.۶: مولفه saturation شکل ۲.۱



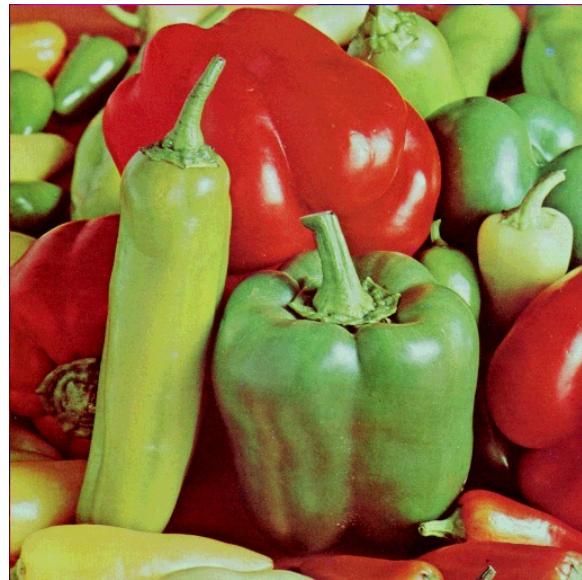
شکل ۲.۱۰: تصویر با ۳۲ سطح رنگی برای هر رنگ



شکل ۲.۸: تصویر با ۸ سطح رنگی برای هر رنگ



شکل ۲.۱۱: تصویر با ۶۴ سطح رنگی برای هر رنگ



شکل ۲.۹: تصویر با ۱۶ سطح رنگی برای هر رنگ

نتیجه PSNR و MSE نیز به شکل زیر است. تصویر اصلی با تصویر با ۶۴ سطح با معیار MSE فقط ۳ واحد تفاوت دارند در حالی که عکس ۴/۱ شده است. با کاهش سطح MSE افزایش یافته و PSNR نیز کاهش می باید، که نشان دهنده تفاوت بیشتر با عکس اصلی است.

	8	16	32	64
MSE	88.9658	73.3051	16.6718	3.35106
PSNR	28.6386	29.4795	35.911	42.879



شکل ۲.۱۳: تصویر اصلی



شکل ۲.۱۴: کانالی رنگی ابی شکل ۲.۱۳

**قسمت ۵.۲.۲:** همانطور که در شرح گفتیم، در این قسمت برخلاف قسمت قبل میزان بیت به کار رفته برای هر رنگ متفاوت است. همانطور که در شکل ۲.۱ که تصویر اصلی است مشاهده می شود، این تصویر به صورت عمده از رنگ های سبز و قرمز تشکیل شده است. در سوال نیز سه بیت به قرمز و سبز و دو بیت به آبی اختصاص داده است.

نتیجه در شکل زیر قابل مشاهده است.



شکل ۲.۱۲: شکل ۲.۱ بعد از کاهش از ۲۴ بیت به ۸ بیت

همانطور که مشاهده می کنیم، این عکس با اینکه یک بیت کمتر از شکل ۲.۸ دارد، اما از دیدگاه ناظر انسانی تفاوت چندانی با آن ندارد.

زیرا عکس مقادیر آبی کمتری دارد.

مقدار mse برای این تصویر ۹۴.۷۵۳۷۹۹۴۳۸۴۷۶۵۶ است که بسیار نزدیک به مقدار mse گزارش شده برای تصویر ۲.۸ است.

همچین PSNR نیز برابر ۲۸.۳۶۴۸۳۷۲۷۵۳۵۶۴۳۲ است که این معیار هم بسیار نزدیک به عکس ۲.۸ است.

### قسمت ۵.۲.۳

#### روش اول:

ابتدا عکس و هر کanal رنگی آن را بررسی می کنیم.

	54
Green	58.5157963479589 75

بنابراین به هر رنگ یک بیت را اختصاص خواهیم داد. نتیجه به شکل زیر خواهد بود:



شکل ۲.۱۷: تصویر شکل ۲.۱۳ با هشت رنگ همانطور که مشاهده می شود کیفیت به طور قابل مشاهده ای کاهش یافته است.  $Mse$  بدست آمده PSNR نیز ۱۱۲.۰۷۴۲۳۲۷۳۷۲۲۳۱ و ۲۷.۶۳۵۷۴۵۸۶۵۰۹۹۲۲ می باشد.

۱۶ رنگ:  
برای ۱۶ رنگ سه حالت را می توان در نظر گرفت:  
**B:2bits G:1bit R:1bit**  
الف: خروجی این حالت به شکل زیر است:



شکل ۲.۱۵: کanalی رنگی سبز شکل ۲.۱۳



شکل ۲.۱۶: کanalی رنگی قرمز شکل ۲.۱۳  
رنگ:

برای ۸ رنگ یعنی کلا ۳ بیت داریم، با توجه به شکل های ۲.۱۶ تا ۲.۱۴ هیچ کدام از رنگ ها مقدار کمی ندارد که قابل حذف باشد. از طرف دیگر انحراف معیار عکس ها مقدار بالای دارد و نمی توان یک رنگ را با مقدار خاصی جایگزین کرد. انحراف معیار به شکل زیر گزارش شده است:

Red	55.8806197190715 54
Blue	55.8806197190715

### **ج: B:1bit G:1bit R:2bit**

خروجی این قسمت به شکل زیر است:



شکل ۲.۲۰: خروجی قسمت ج

همانطور که در شکلها مشاهده می شود، اختصاص دو بیت برای رنگ قرمز عکسی نزدیکتر به عکس واقعی را به عنوان خروجی می دهد.  
جدول زیر نیز گواهی بر این حرف است.

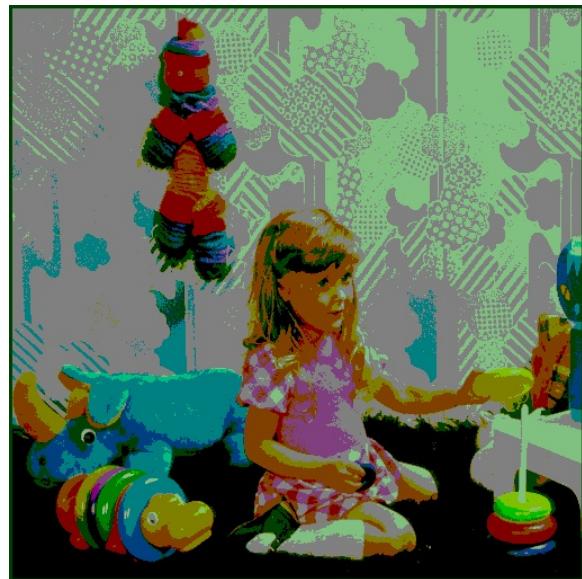
	الف	ب	ج
PSNR	27.6049 25	27.7104 42	27.7350 22
MSE	112.872 40	110.163 09	109.541 35



شکل ۲.۱۸: خروجی قسمت الف

### **ب: B:1bit G:2bits R:1bit**

خروجی این قسمت به شکل زیر است:



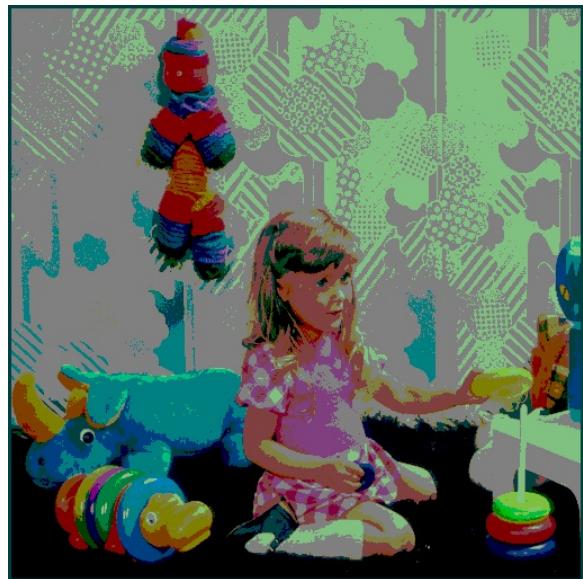
شکل ۲.۱۹: خروجی قسمت ب

**ج: B:1bit G:2bits R:2bit**



شکل ۲.۲۳: خروجی قسمت ج

**الف: B:2bit G:2bits R:1bit**



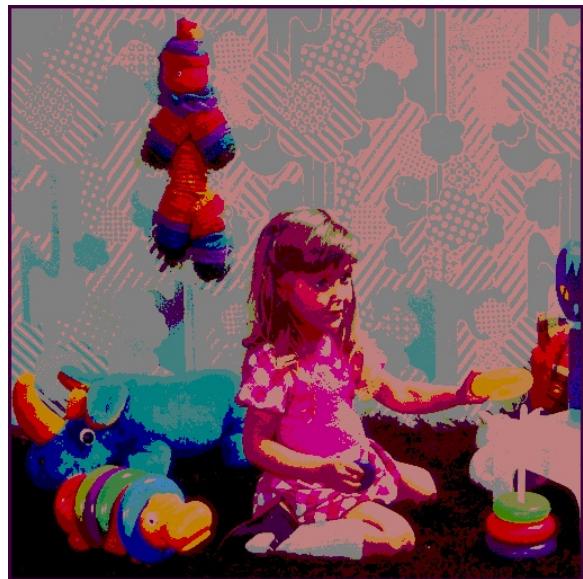
شکل ۲.۲۱: خروجی قسمت الف

**د: B:3bit G:1bits R:1bit**



شکل ۲.۲۴: خروجی قسمت د

**ب: B:2bit G:2bits R:2bit**



شکل ۲.۲۲: خروجی قسمت ب

**۳۲ رنگ:**

برای ۳۲ رنگ می توان ۶ حالت را در نظر گرفت:

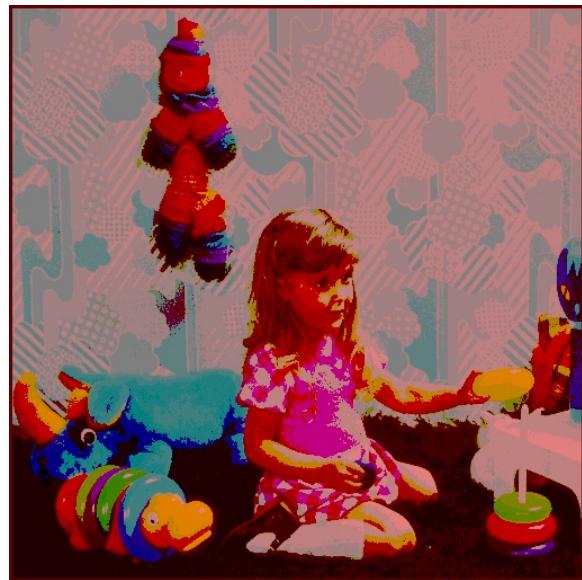
	MSE	PSNR
الف	110.96127	27.6790891
ب	110.33953	27.70349
ج	107.63022	27.8114612
د	103.394870	27.9858136
ه	105.70086	27.890018
و	104.89967	27.9230622

**روش دوم:**  
همانطور که مشاهده می شود خروجی این روش بسیار بهتر است و بسیار به عکس اصلی نزدیکتر است.



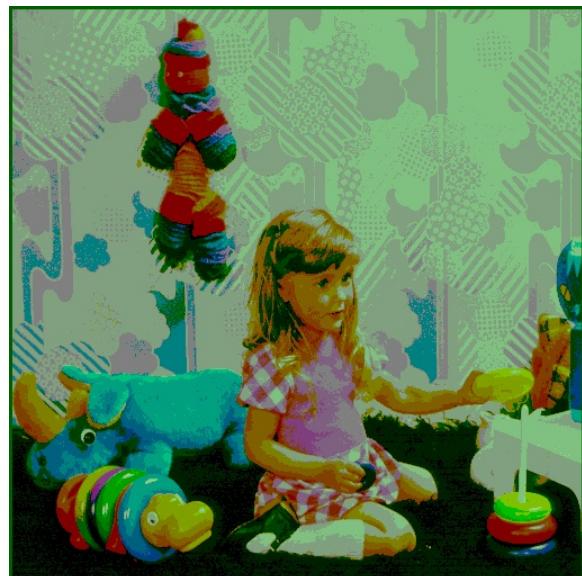
شکل ۲.۲۶: خروجی با استفاده از الگوریتم kmeans و هشت رنگ

**B:1bit G:1bits R:3bit :**



شکل ۲.۲۵: خروجی قسمت د

**B:1bit G:3bits R:1bit :**



شکل ۲.۲۵: خروجی قسمت و

در این قسمت تشخیص اینکه کدام عکس به عکس اولیه نزدیکتر است با چشم کار آسانی نیست. به همین دلیل از معیار های شباهت استفاده می کنیم. مشاهده می کنیم قسمت د بهترین نتیجه را داشته است. یعنی اختصاص سه بیت به رنگ قرمز و یک رنگ به رنگهای آبی و سبز. این نشانگر اهمیت رنگ قرمز در این تصویر است.

در حالیکه کمترین مقدار به دست امده mse برای هشت رنگ در روش اول ۱۱۲ است در این روش به ۷۲ کاهش پیدا کرده است. در بقیه حالات هم کاهش بسیاری در MSE اتفاق افتاده است.



شکل ۲.۲۷: خروجی با استفاده از الگوریتم kmeans و ۱۶ رنگ



شکل ۲.۲۸: خروجی با استفاده از الگوریتم kmeans و ۳۲ رنگ

:

	۳۲ رنگ	۱۶ رنگ	۸ رنگ
PSNR	31.3372	30.2948	29.5478
MSE	47.7923	60.7572	72.1603

```

def compute_S(img):
    width , height = img.shape[0:2]
    saturation    = np.ones((width , height))
    for i in range (0,width):
        for j in range(0,height):
            pixel = img[i][j]
            saturation[i][j] = 1.00 - ((3 * np.amin(pixel)) / np.sum(pixel))
    # print(saturation)
    return saturation

img = cv.imread("image/Pepper.bmp")

img = img / 255

img_h = normalize(compute_H(img))
img_s = compute_S(img) * 255
img_i = normalize(compute_I(img))

cv.imwrite("result/5.1.1/hue3.jpg" , img_h)
cv.imwrite("result/5.1.1/saturation2.jpg" , img_s)
cv.imwrite("result/5.1.1/intensity2.jpg" , img_i)

```

#### قسمت ۵.۲.۱

```

import cv2 as cv
import numpy as np
import pandas as pd
import math

def quantizer(img , level):
    t_level = int(256 / level)
    result = np.uint8((np.floor(img / t_level)) * t_level)
    return result

def MSE(img , new_img):
    mse = np.mean(np.square(new_img - img))
    return mse

```

#### APPENDIX:

##### قسمت ۵.۱.۱

```

import cv2 as cv
import numpy as np

def normalize(img):
    minimum = np.amin(img)
    return (img - minimum) * (255 / np.amax(img))

#compute Hue
def compute_H(pixel):
    width , height = img.shape[0:2]
    hue = np.zeros((width , height))
    for i in range (0,width):
        for j in range(0,height):
            pixel = img[i][j]
            den = ((pixel[0] - pixel[1]) ** 2 + (pixel[2] - pixel[1])*(pixel[0] - pixel[2])) ** (1/2)
            # den = ((pixel[2] - pixel[1]) ** 2 + ((pixel[0] - pixel[1]) * (pixel[2] - pixel[0]))) ** (1/2)
            exp = ((pixel[2] - pixel[1]) + (pixel[2] - pixel[0])) / 2
            if(den != 0):
                hue[i][j] = np.arccos(exp / den)
                if(pixel[0] > pixel[1]):
                    hue[i][j] = 2 * np.pi - hue[i][j]
            print(hue)
    return hue

#compute Intensity
def compute_I(img):
    width , height = img.shape[0:2]
    intensity    = np.zeros((width , height))
    for i in range (0,width):
        for j in range(0,height):
            pixel = img[i][j]
            intensity[i][j] = np.sum(pixel) / 3
    return intensity

#compute Saturation

```

```

        mse = np.mean(np.square(new_img - img))
        return mse

def PSNR(img , new_img):
    mse = MSE(img , new_img)
    if mse == 0:
        return 100
    PIXEL_MAX = 255.0
    return 20 * math.log10(PIXEL_MAX / math.sqrt(mse))

img = cv.imread("image/Pepper.bmp")

cv.imwrite("result/5.2.2/org_blue.jpg" ,
img[:, :, 0])
cv.imwrite("result/5.2.2/org_green.jpg" ,
img[:, :, 1])
cv.imwrite("result/5.2.2/org_red.jpg" ,
img[:, :, 2])

result = np.zeros(img.shape , np.uint8)

result[:, :, 0] = quantizer(img[:, :, 0] ,
4)
result[:, :, 1] = quantizer(img[:, :, 1] ,
8)
result[:, :, 2] = quantizer(img[:, :, 2] ,
8)

print("MSE" , MSE(result , img))
print("PSNR" , PSNR(result , img) )
cv.imwrite("result/5.2.2/result.jpg" ,
result)
cv.imwrite("result/5.2.2/blue.jpg" ,
result[:, :, 0])
cv.imwrite("result/5.2.2/green.jpg" ,
result[:, :, 1])
cv.imwrite("result/5.2.2/red.jpg" ,
result[:, :, 2])

```

الف: ٥.٢.٣

```

import cv2 as cv
import numpy as np
import math

```

```

def PSNR(img , new_img):
    mse = MSE(img , new_img)
    if mse == 0:
        return 100
    PIXEL_MAX = 255.0
    return 20 * math.log10(PIXEL_MAX / math.sqrt(mse))

img = cv.imread('image/Pepper.bmp')

report = pd.DataFrame(index=['MSE' ,
'PSNR'], columns = [ '8' , '16' , '32' ,
'64'])

gray_level = np.array([8 , 16 , 32 , 64])

for level in gray_level:
    quantized_img = quantizer(img,
level)
    report.set_value('MSE' , str(level),
MSE(quantized_img , img))
    report.set_value('PSNR' , str(level),
PSNR(quantized_img , img))
    cv.imwrite("result/5.2.1/" +
str(level) + ".jpg" , quantized_img)

print(report)

```

:٥.٢.٤

```

import cv2 as cv
import numpy as np
import math

def quantizer(img , level):
    t_level = int(256 / level)
    result = np.uint8((np.floor(img /
t_level)) * t_level)
    return result

def MSE(img , new_img):

```

```

cv.imwrite("result/5.2.3/blue.jpg" ,
img[:, :, 0])
cv.imwrite("result/5.2.3/green.jpg" ,
img[:, :, 1])
cv.imwrite("result/5.2.3/red.jpg" ,
img[:, :, 2])

print("blue variance" ,
np.sqrt(img[:, :, 0].var()))
print("green variance" ,
np.sqrt(img[:, :, 1].var()))
print("red variance" ,
np.sqrt(img[:, :, 2].var()))

#for 8 level
eight_color =
quantizer_with_different(img , [2 , 2 ,
2])
print("For 8 levels. MES" ,
MSE(eight_color , img))
print("For 8 levels. PSNR" ,
PSNR(eight_color , img))
cv.imwrite("result/5.2.3/8level.jpg" ,
eight_color)

# for 16 levels
levels_16 = np.array([[4 , 2 , 2], [2 ,
4 , 2] , [2 , 2 , 4]])
index = 0
for level in levels_16:
    sixteen_color =
quantizer_with_different(img , level)
    cv.imwrite("result/5.2.3/16level" +
str(index) + ".jpg" , sixteen_color)
    print("For 8 levels. MES " +
str(index), MSE(sixteen_color , img))
    print("For 8 levels. PSNR " +
str(index) , PSNR(sixteen_color , img))
    index += 1

# for 32 levels

levels_32 = np.array([[4 , 4 , 2], [4 ,
2 , 4] , [2 , 4 , 4] , [8 , 2 , 2] ,
[2 , 2 , 8] , [2 , 8 , 2]])
index = 0

```

```

def MSE(img , new_img):
    mse = np.mean(np.square(new_img -
img))
    return mse

def PSNR(img , new_img):
    mse = MSE(img , new_img)
    if mse == 0:
        return 100
    PIXEL_MAX = 255.0
    return 20 * math.log10(PIXEL_MAX /
math.sqrt(mse))

def quantizer(img , level):
    t_level = int(256 / level)
    result = np.uint8((np.floor(img /
t_level)) * t_level)
    return result

def quantizer_with_different(img ,
levels):
    result = np.zeros(img.shape , np.uint8)
    result[:, :, 0] =
quantizer(img[:, :, 0] , levels[0])
    result[:, :, 1] =
quantizer(img[:, :, 1] , levels[1])
    result[:, :, 2] =
quantizer(img[:, :, 2] , levels[2])
    return result

def MSE(img , new_img):
    mse = np.mean(np.square(new_img -
img))
    return mse

def PSNR(img , new_img):
    mse = MSE(img , new_img)
    if mse == 0:
        return 100
    PIXEL_MAX = 255.0
    return 20 * math.log10(PIXEL_MAX /
math.sqrt(mse))

img = cv.imread('image/Girl.bmp')

```

```

quant = quant.reshape((h, w, 3))
image = image.reshape((h, w, 3))
quant      = cv2.cvtColor(quant,
cv2.COLOR_LAB2BGR)
image      = cv2.cvtColor(image,
cv2.COLOR_LAB2BGR)
print("mse" , MSE(image , quant))
print("psnr" , PSNR(image , quant))

```

```

for level in levels_32:
    sixteen_color           =
quantizer_with_different(img , level)
    # cv.imwrite("result/5.2.3/32level"
+ str(index) + ".jpg" , sixteen_color)
    print("For 16 levels. MES " +
str(index), MSE(sixteen_color , img))
    print("For 16 levels. PSNR " +
str(index) , PSNR(sixteen_color , img))
    index += 1

```

```

from      sklearn.cluster      import
MiniBatchKMeans
import numpy as np
import cv2
import math

def MSE(img , new_img):
    mse   = np.mean(np.square(new_img -
img))
    return mse

def PSNR(img , new_img):
    mse = MSE(img , new_img)
    if mse == 0:
        return 100
    PIXEL_MAX = 255.0
    return 20 * math.log10(PIXEL_MAX /
math.sqrt(mse))

image = cv2.imread("image/Girl.bmp")
(h, w) = image.shape[:2]
image      = cv2.cvtColor(image,
cv2.COLOR_BGR2LAB)
image = image.reshape((image.shape[0] *
image.shape[1], 3))
clt = MiniBatchKMeans(8)
labels = clt.fit_predict(image)
quant           =
clt.cluster_centers_.astype("uint8")[la
bels]

```