به نام خدا

گزارش پروژه پایانی پردازش تصویر دیجیتال

موضوع:

# جابجایی پیکسل ها (Pixel shuffling)


اعضای تیم:

آرش رسولی
ابوالفضل اسد

استاد:
دکتر کسایی


تابستان 1403

# فهرست

**1**. مقدمه.....3
2. بهبود کارایی.....3
2.1. بهبود سرعت.....3
2.2. بهبود کیفیت.....4
2.2.1. بهبود کیفیت به صورت **Global** ..... **4**
2.2.2. بهبود کیفیت به صورت **Local** ..... **6**
3. کاربرد ها.....12
3.1. تغییر دامنه تصویر.....12
3.2. رنگی کردن تصاویر سیاه و سفید.....17
3.3. فشرده سازی ویدیو.....19
4. جمع بندی.....20
5. مراجع.....20

# 1. مقدمه

پروژه pixel shuffling یا همان جابجایی پیکسل ها، یک روش شکل گیری تصویر[1] است. در این روش با جابجا کردن پیکسل های تصویر اول و مرتب کردن آن ها ، سعی می شود که تصویر دوم تولید شود. نتیجه تصویری است که فرم و ساختار تصویر دوم و رنگ های تصویر اول را دارد. در تصویر زیر یک نمونه از کارکرد این روش قابل مشاهده است که با حفظ ساختار تصویر سمت چپ و استفاده از رنگ های تصویر وسط، تصویر سمت راست ایجاد شده است.

نحوه عملکرد الگوریتم به این صورت است، بر اساس دقت مورد نظر تصویر اول و تصویر دوم را به بلوک های کوچکتر می شکند. سپس میانگین هر بلاک را در سه کانال رنگی متفاوت محاسبه می کند. حال به ازای هر بلاک در تصویر هدف نزدیک ترین بلاک(با توجه به فاصله (نرم) میانگین آنها) را انتخاب و آن را جایگزین می کند. این فرآیند با استفاده از الگوریتم Balanced Assignment Problem صورت می گیرد. کد های این پروژه در این لینک قابل دسترسی است.

ما در این پروژه قصد داریم به بهبود کارایی کد های موجود چه از لحاظ کیفیت و چه از لحاظ سرعت بپردازیم و همچنین با استفاده از مفاهیم درس کاربرد های جدید و جالبی برای این پروژه معرفی کنیم .

# 2. بهبود کارایی

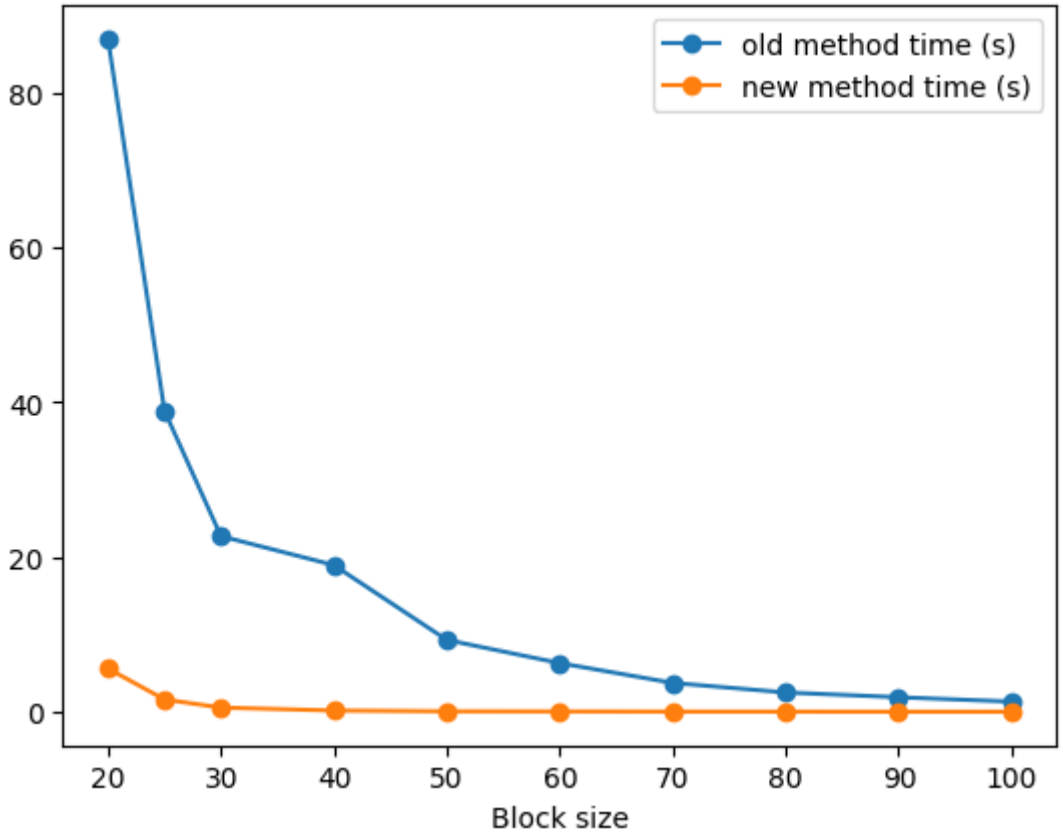
## 2.1. بهبود سرعت

در کد های اولیه محاسبات به صورت حلقه های تو در تو صورت می گرفت و اگر دو تصویر n*n (n تعداد بلوک ها است) داشتیم اردر انجام محاسبات برای بدست آوردن ماتریس هزینه ها (cost matrix) که در الگوریتم Assignment استفاده می شد $O(n^4)$ بود. ما حلقه های محاسبه را از بین برده و به صورت ماتریسی آنها را انجام دادم که منجر شد به مرتبه زمانی $O(n^2)$ برسیم. البته مرتبه زمانی کل الگوریتم وابسته به مرتبه زمانی الگوریتم Balanced Assignment است که برابر با $O(n^3)$ می‌باشد.

در ادامه مقایسه‌ی زمان اجرای کد اولیه و بهبود یافته را مشاهده می‌کنید. در اینجا ما الگوریتم را روی ورودی اصلی مسئله چندین بار برای سایزهای مختلف بلوک اجرا کرده و میانگین زمان آن‌ها را محاسبه کردیم.

---

[1] Image Morphing

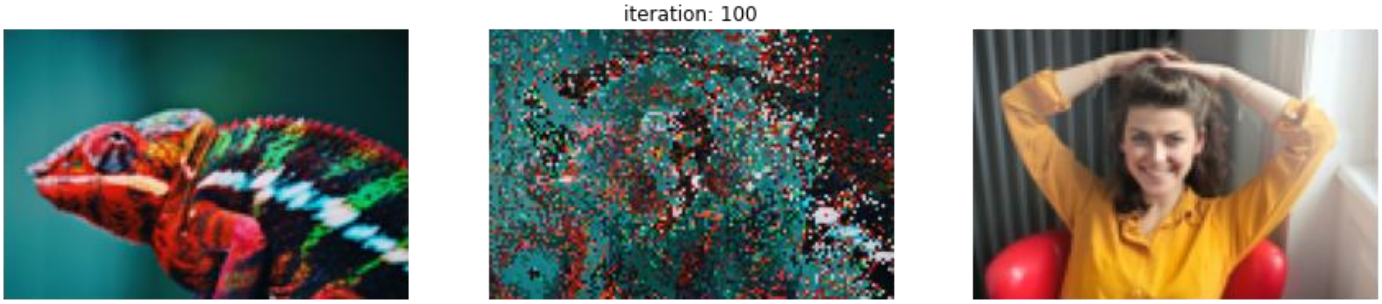



## 2.2. بهبود کیفیت

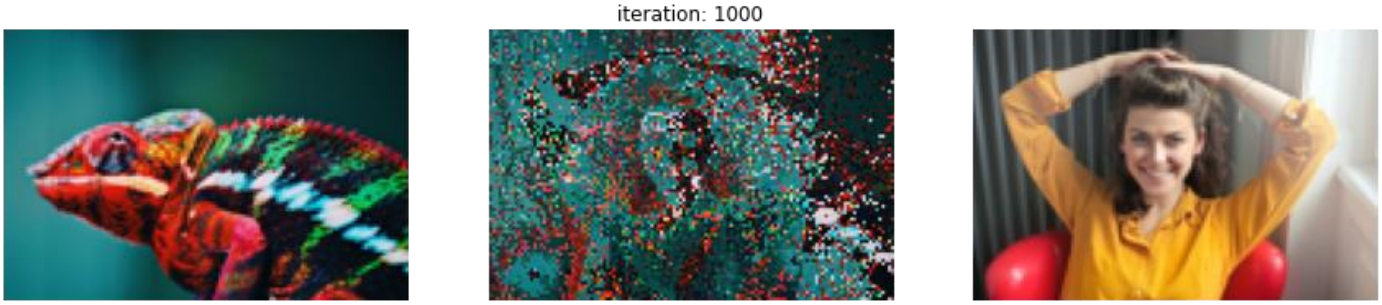
### 2.2.1. بهبود کیفیت به صورت **Global**

در کد اولیه نحوه‌ی کار به صورت بلوکی است به این معنا که با جابه‌جا کردن بلوک‌های مختلف، تلاش می‌کند تصویر مقصد را از بلوک‌های تصویر مبدا بسازد، این کار با وجود سرعت بالاتر، کیفیت پایین‌تری دارد و جزئیات تصویر به خوبی ساخته نمی‌شود. در اینجا ما به دنبال این رفتیم که این جابه‌جا کردن واحدهای تصویر را در سطح پیکسل انجام دهیم. می‌دانیم که با توجه به ابعاد تصاویر و مرتبه‌ی زمانی $O(n^3)$ انجام این کار به صورت دقیق امکان پذیر نبوده و باید تلاش کنیم آن را به صورت تقریبی انجام دهیم.

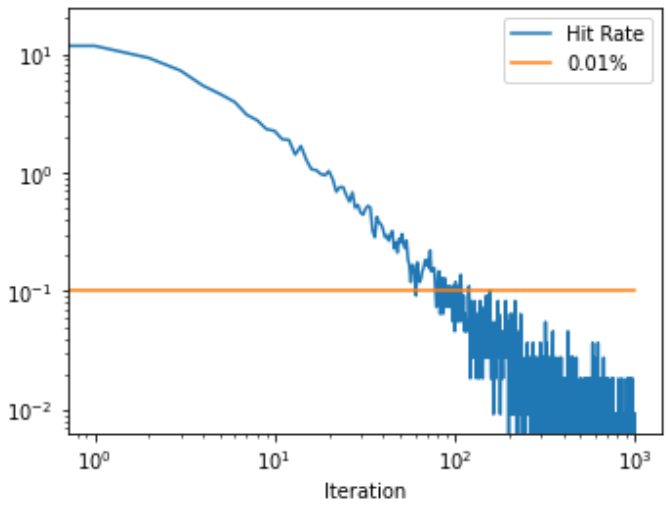
برای اینکار ابتدا به این صورت عمل کردیم که در تعداد دفعات بالا دو پیکسل از تصویر را به صورت تصادفی انتخاب کرده و بررسی می‌کنیم که آیا با جابه‌جا کردن آن‌ها فاصله‌ی دو تصویر از هم کم می‌شود یا نه و اگر این اتفاق می‌افتاد این پیکسل‌ها را جابه‌جا می‌کردیم. با این روش انتظار داریم که در هر مرحله فاصله کمتر شده و ما به تصویر نهایی نزدیک‌تر شویم. برای اینکه این کار سریع‌تر انجام شود ابعاد تصاویر را در هر بعد ۱۰ برابر کوچک کردیم. نتیجه‌ی آن پس از ۱۰۰ iteration به صورت زیر شد که در آن هر iteration به اندازه‌ی تعداد پیکسل‌های تصویر زوج‌های کاندید برای جابه‌جایی انتخاب می‌کند و جابه‌جایی آن‌ها را بررسی می‌کند.

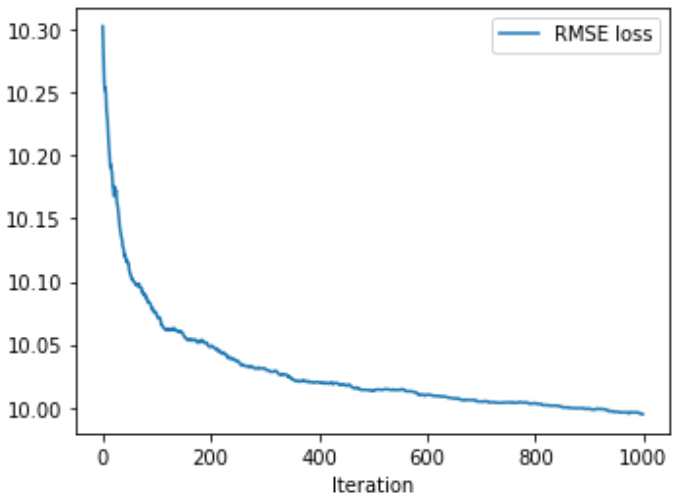



پس از ۱۰۰۰ iteration نیز خروجی تغییر زیادی نمی‌کند.



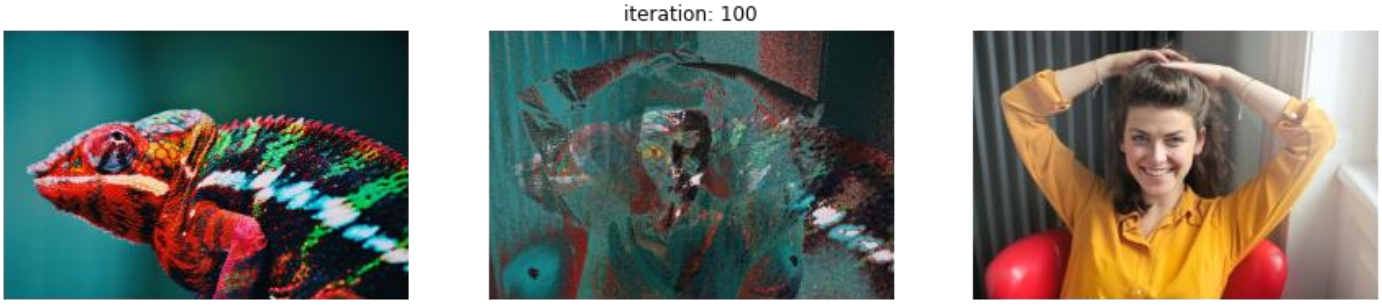

دلیل آن نیز این است که بعد از تعدادی مرحله، نرخ hit rate بسیار کاهش یافته و تعداد خیلی کمی پیکسل جابه‌جا می‌شوند.



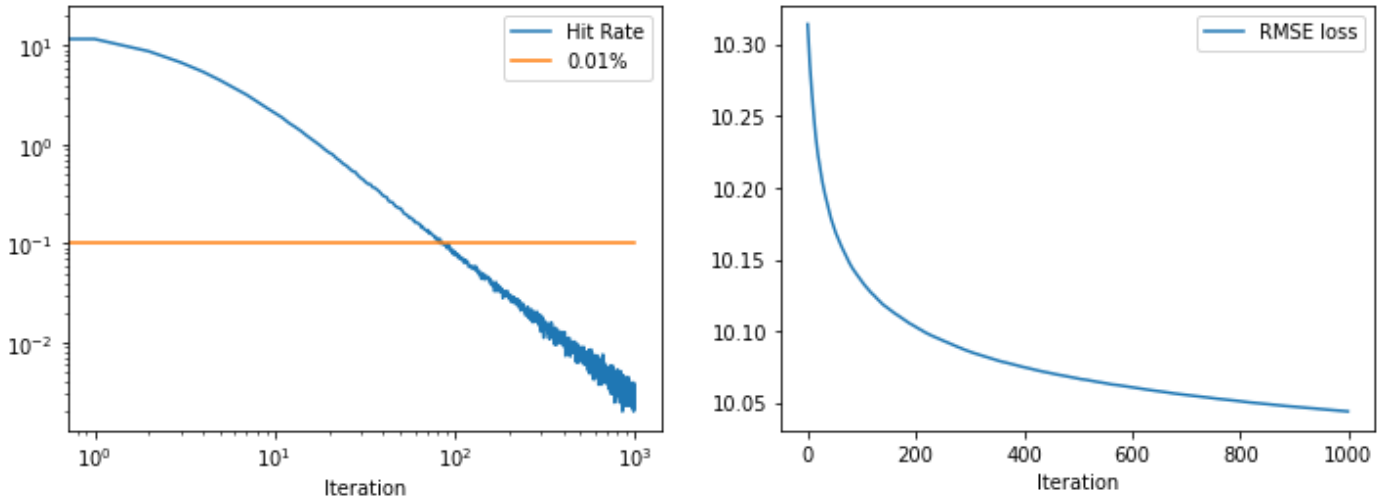




همانطور که مشاهده می‌کنید در نمودار بالا که لگاریتمی است، پس از حدود ۱۰۰ iteration نرخ hit rate کمتر از 0.1 درصد خواهد شد و در نتیجه تعداد بسیار کمی جابه‌جایی اتفاق می‌افتد. همچنین همانطور که مشاهده می‌کنید میزان فاصله‌ی تصویر تولید شده با تصویر مقصد نیز در حال کاهش است.

ما همین کار را برای تصاویر در ابعاد اصلی آن‌ها با صرف زمان بیشتر تکرار کردیم. نتایج پس از ۱۰۰ و ۱۰۰۰ iteration به صورت زیر است.




که همانطور که مشاهده می‌کنید مشکل برخورد پایین در اینجا هنوز وجود دارد.

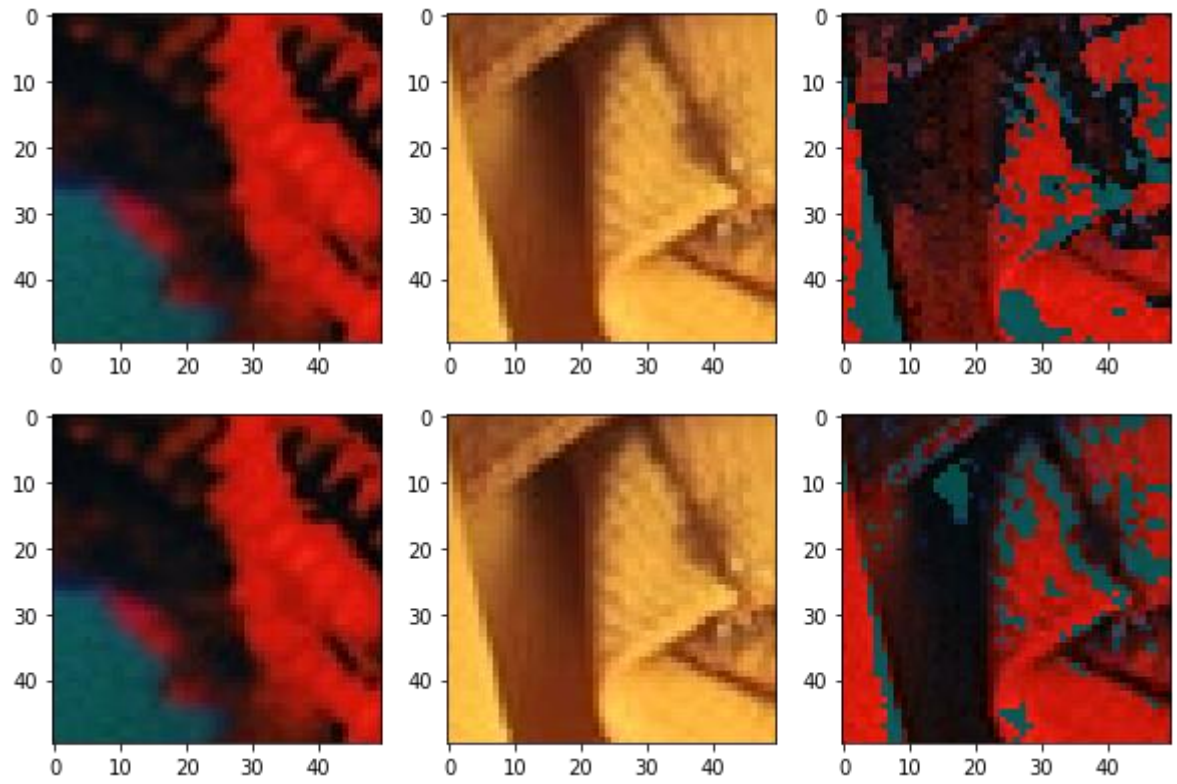



همچنین همانطور که مشاهده می کنید حالهای از تصویر اولیه در تصویر خروجی وجود دارد که قابل قبول نیست، برای رفع این مشکل، تصویر مبدا را ابتدا کاملا بههم ریخته و سپس الگوریتم را روی آن اجرا کردیم که خروجی آن به صورت زیر در آمد.


در اینجا همانطور که مشاهده می‌کنید جزئیات مناسب است ولی ترکیب رنگی مناسبی ندارد.

### 2.2.2. بهبود کیفیت به صورت **Local**

برای رفع مشکل ترکیب رنگی تصمیم گرفتیم که به این صورت عمل کنیم که ابتدا با جابهجایی بلوک‌ها ترکیب رنگی مناسب را ایجاد کرده و سپس در بلوک‌های کوچک به صورت دقیق تلاش کنیم که تصویر را بازسازی کنیم.
برای ساخت تصویر در بلوک‌های کوچک ابتدا به این صورت عمل شد که تلاش شود پیکسل‌ها به صورتی جابه‌جا شوند که فاصله‌ی تصویر ساخته شده از تصویر مقصد کمینه شود ولی ما در اینجا دوست داریم که الگوی تصویر ساخته شده و لزوما فاصله‌ی دو بلوک مهم نیست پس برای این کار به این صورت عمل کردیم که در هر بلوک، میانگین کانال‌های مختلف کم شده و تلاش می‌شود تصویر با جابه‌جایی پیکسل‌ها ساخته شود سپس میانگین کانال‌های تصویر مبدا به تصویر ساخته شده اضافه می‌شود. تفاوت این کار را در زیر مشاهده می‌کنید.

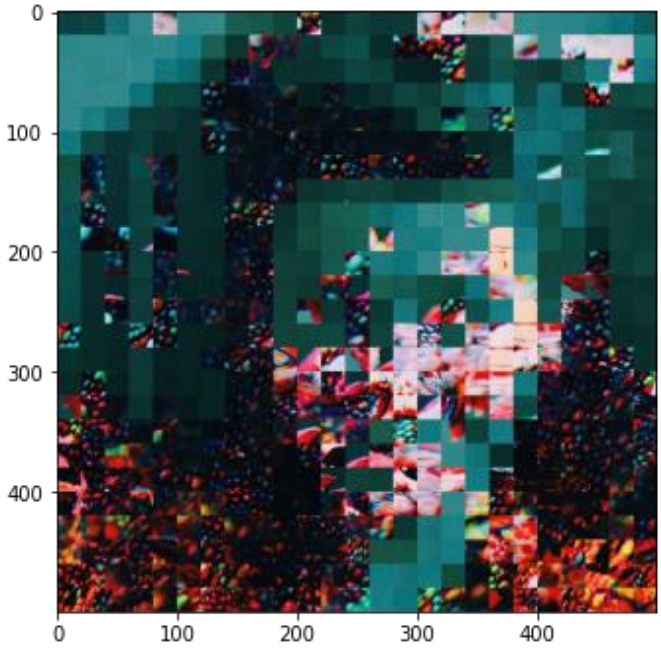


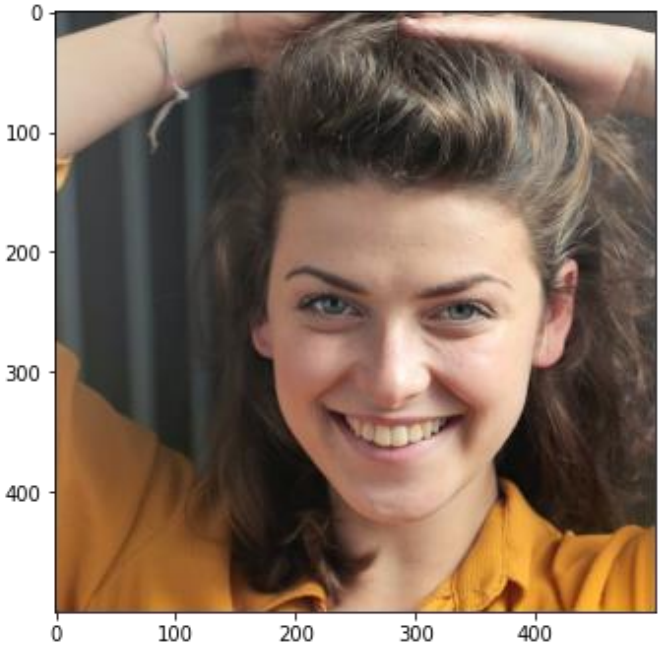
همانطور که مشاهده می‌کنید در این جا ما تصویر سمت چپ را به عنوان مبدا در نظر گرفته و می‌خواهیم با آن تصویر وسط را بسازیم. با حذف میانگین و ساختن تصویر بدون میانگین خروجی پایین بدست می‌آید که کیفیت بهتری دارد.
حال برای ساختن تصویر نهایی به این صورت عمل کردیم که ابتدا به صورت بلوکی جابه‌جایی را انجام داده و سپس با استفاده از خروجی‌آن به صورت محلی تصویر را بازسازی کردیم. برای اینکار ابتدا با بلوک‌هایی با ابعاد ۲۰*۲۰ تصویر را بازسازی کردیم.

سپس در دو ابعاد ۲۵*۲۵ و ۳۰*۳۰ به صورت محلی پیکسل‌ها را جابه‌جا کردیم. خروجی آن‌ها به صورت زیر در آمد.

همانطور که مشاهده می‌کنید با این کار هم ترکیب رنگی تا حدی بازسازی شده و هم اینکه جزییات از بین نرفته است. با بزرگ‌تر شدن ابعاد نیز مربع‌ها بزرگتر شده و پیکسل‌های بیشتری در هر ناحیه برای بازسازی وجود دارد و در نتیجه کیفیت بهتر است ولی زمان بیشتری برای اجرا نیاز دارد.

مشکلی که در خروجی بالا وجود دارد این است که حاشیه‌های مربع‌ها قابل مشاهده است برای رفع این مشکل در اینجا به این صورت عمل کردیم که گام پرش‌ها را کمتر از ابعاد بلوک‌های بازسازی گذاشتیم، در این حالت با فدا کردن زمان اجرا، کیفیت خروجی را به صورت قابل ملاحظه‌ای افزایش دادیم و دیگر مرزهای بلوک‌ها قابل مشاهده نیست. در دو تصویر زیر، ابعاد بلوک‌های محلی همان ۲۵*۲۵ و ۳۰ *۳۰ است با این تفاوت که گام پرش ۵ پیکسل می‌باشد.
ورودی مبدا و مقصد ما به صورت زیر است.




و خروجی‌های آن نیر برای ۲۵*۲۵ و ۳۰*۳۰ به صورت زیر است.

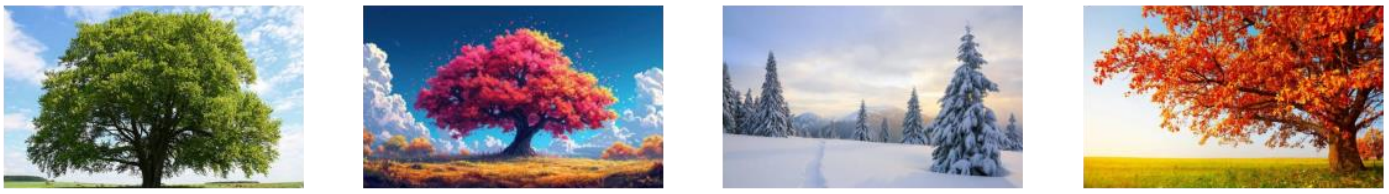
که همانطور که مشاهده می‌کنید با این کار جزئیات به صورت کامل وجود دارد. ترکیب رنگی نیز تا حدی شباهت دارد. و مشکل مشبک بودن خروجی نیز رفع شده است. خروجی با بلوک‌های ۳۰*۳۰ نیز برای کل تصویر به صورت زیر است.

# 3. کاربرد ها

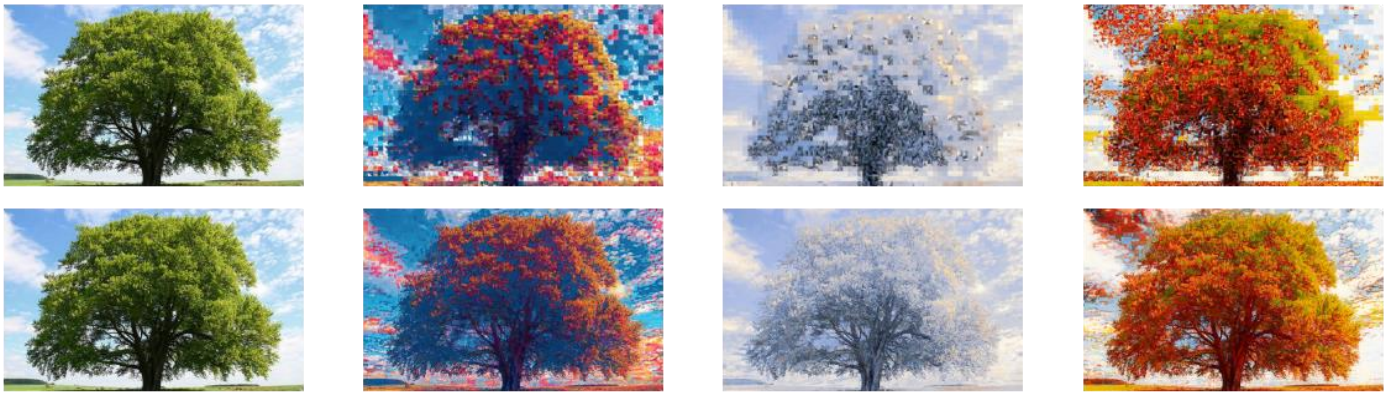
## 3.1. تغییر دامنه تصویر

با توجه به امکان بهبود کیفیت خروجی‌ها، ما از این روش برای تغییر دامنه‌ی تصاویر استفاده کردیم. به عنوان مثال تصویر زیر را در نظر بگیرید.

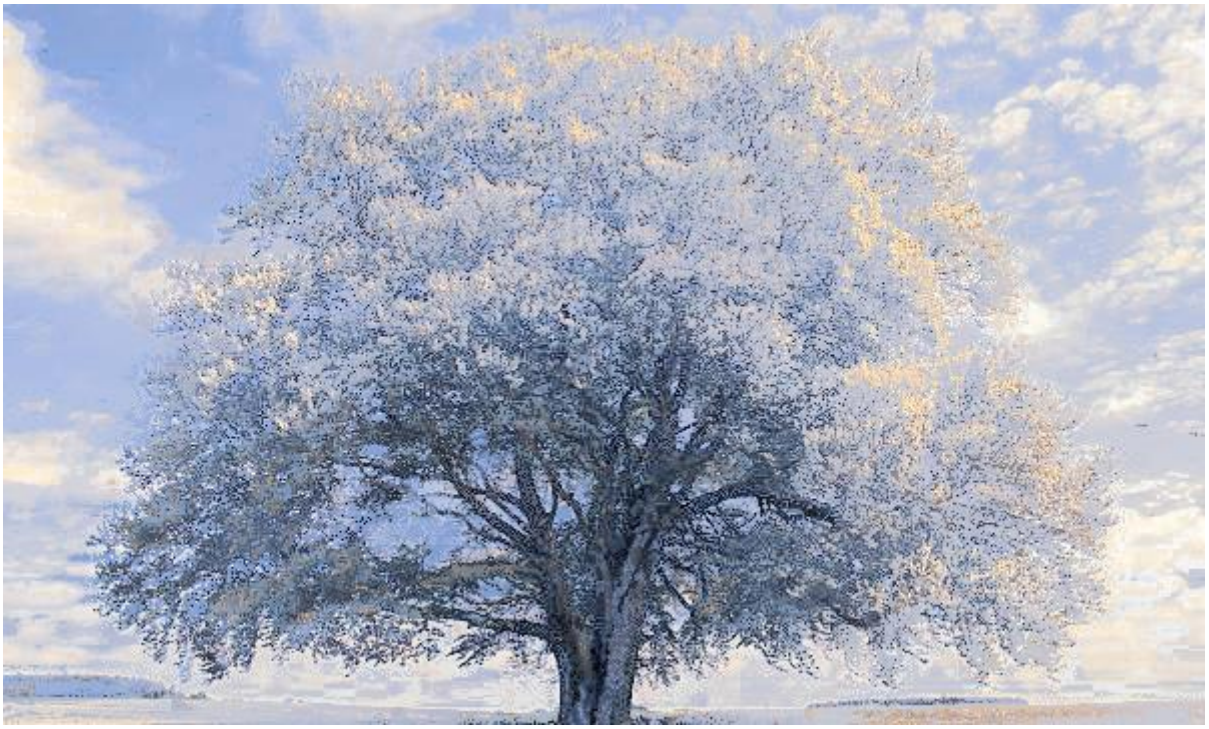
حال می‌خواهیم این تصویر را با پیکسل‌های تصاویر دیگری بازسازی کنیم. تصاویری که قرار است با آن‌ها تصویر بالا ساخته شود به صورت زیر هستند.



ابتدا با استفاده از روش بلوکی، پیکسل‌ها را بلوکی جابه‌جا کرده و سپس تصویر با کیفیت را بازسازی می‌کنیم.

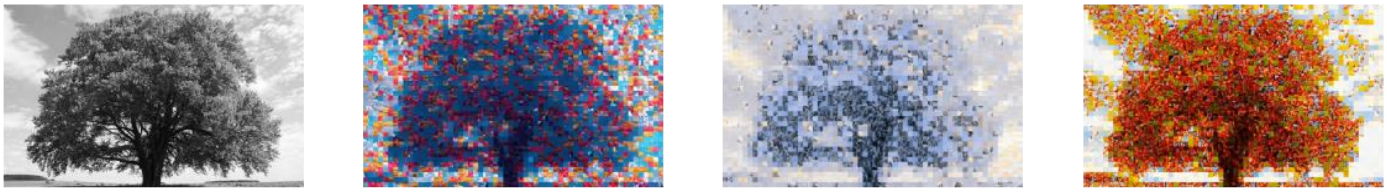


در ادامه نیز هر کدام از آن‌ها را به صورت کامل مشاهده می‌کنید.

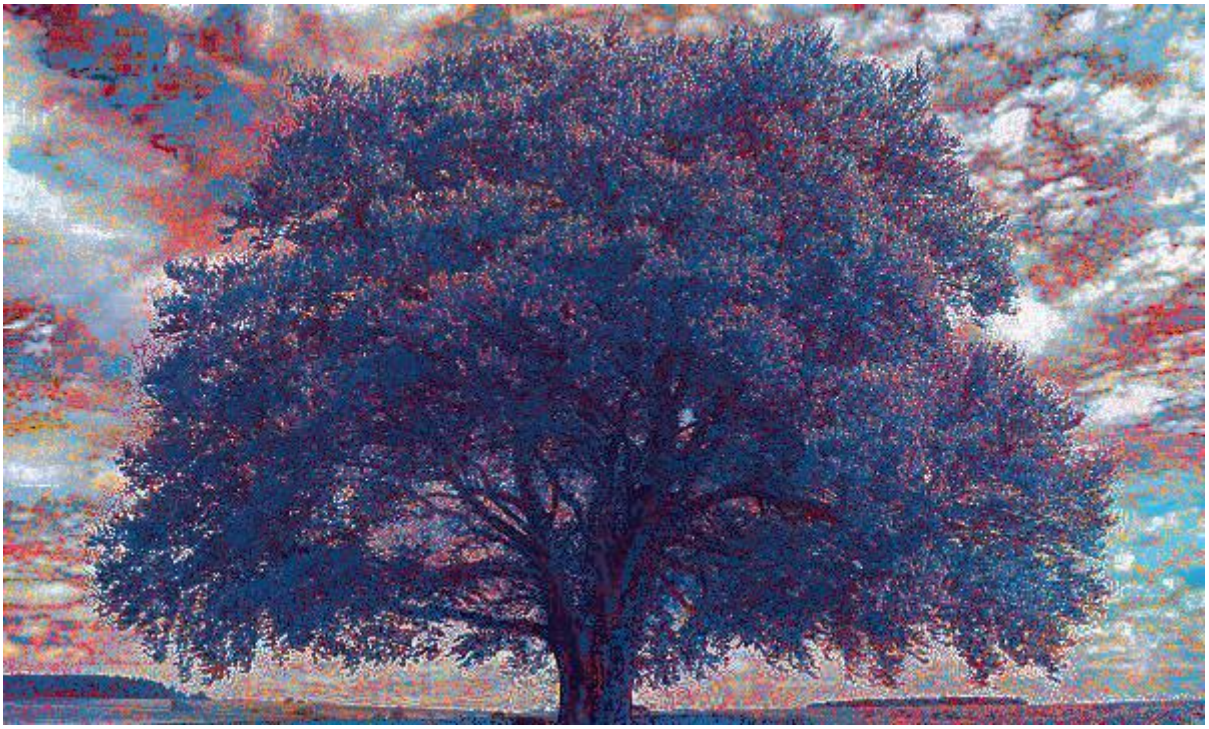


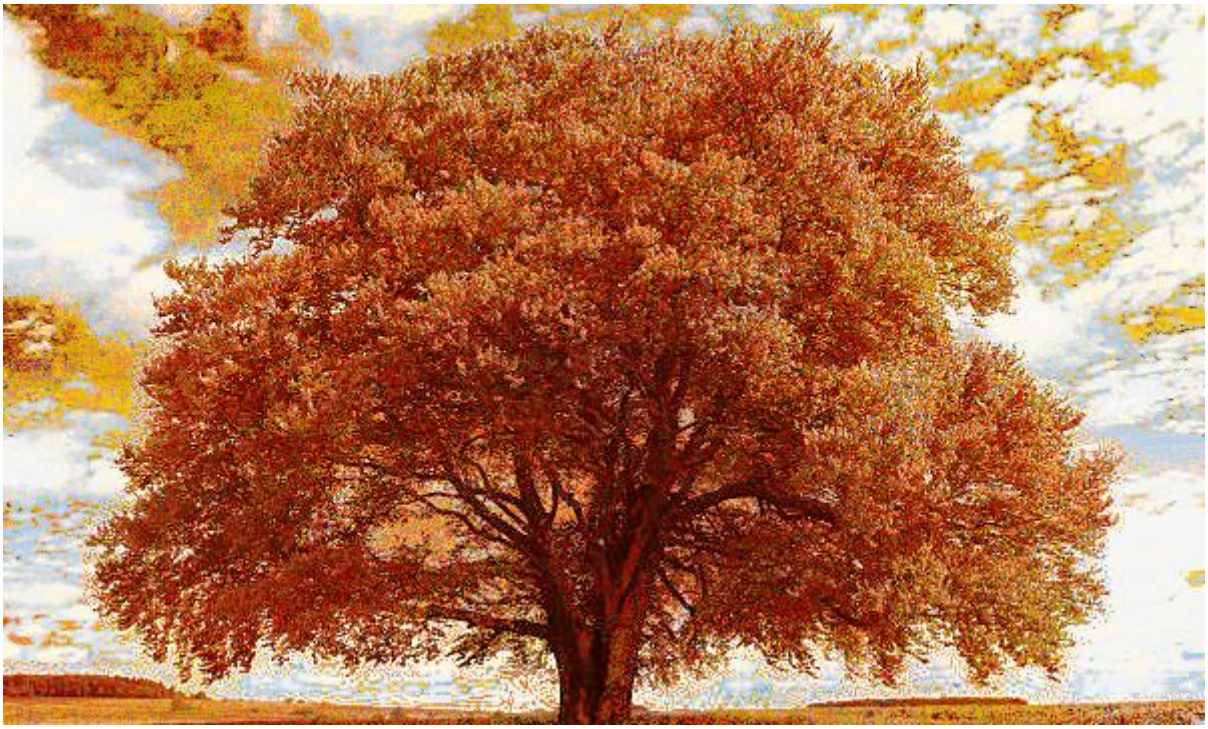
که مشاهده می‌شود که تصویر ورودی ما به صورت تقریبا مناسبی به دامنه‌ی دیگر رفته است.

همچنین از این ایده می‌توان برای رنگی کردن تصاویر استفاده کرد به این صورت که تصویر اولیه را سیاه‌سفید در نظر گرفته و تلاش کنیم آن را با استفاده از تصویرهای رنگی دیگر بازسازی کنیم. خروجی الگوریتم بالا با ورودی حالت سیاه‌سفید تصویر قبلی به صورت زیر خواهد بود.



و بعد از بهبود کیفیت به صورت زیر در آمده است.



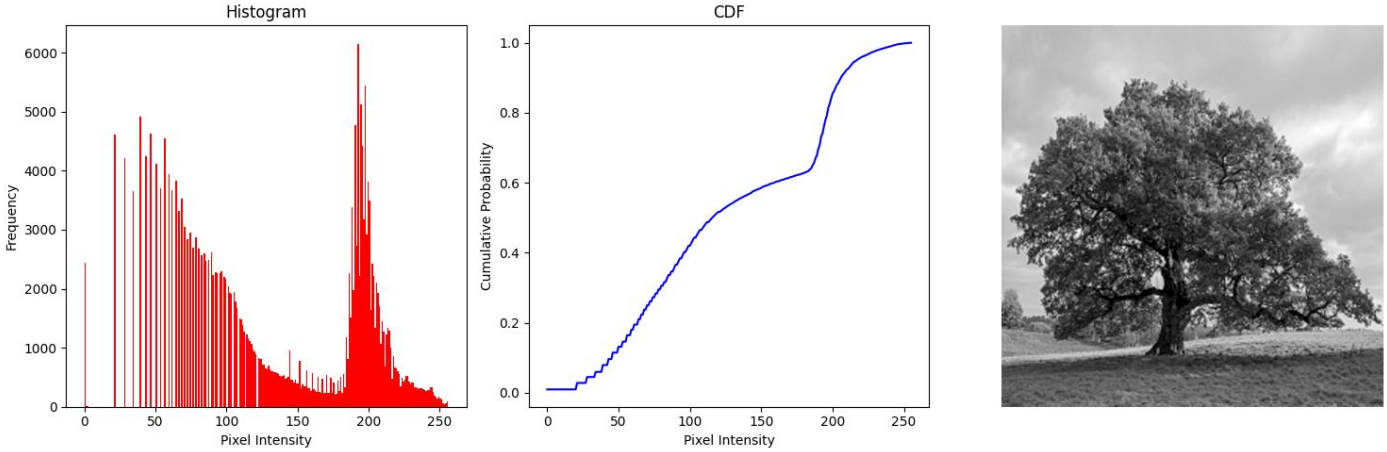


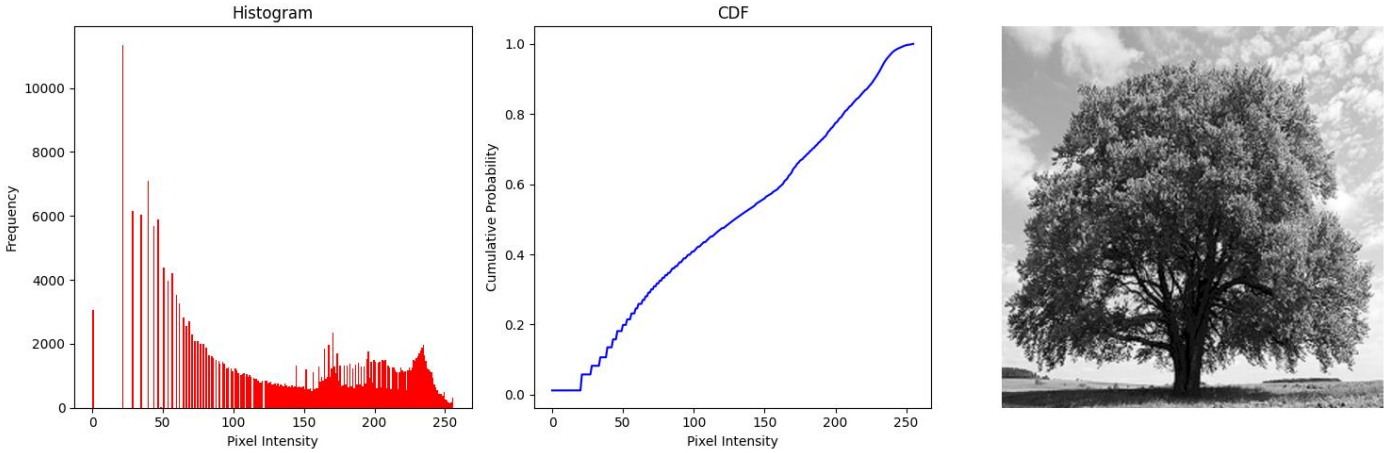
در اینجا ما از تغییر دامنه برای رنگی کردن تصویر استفاده کردیم. مشاهده می‌کنید که در بعضی نمونه‌ها پخش رنگ به خوبی نبوده و تصویر به صورت مناسبی رنگی نشده است. برای رفع این مشکل در بخش بعد این کار را با روش دیگری پیاده کردیم.

## 3.2. رنگی کردن تصاویر سیاه و سفید

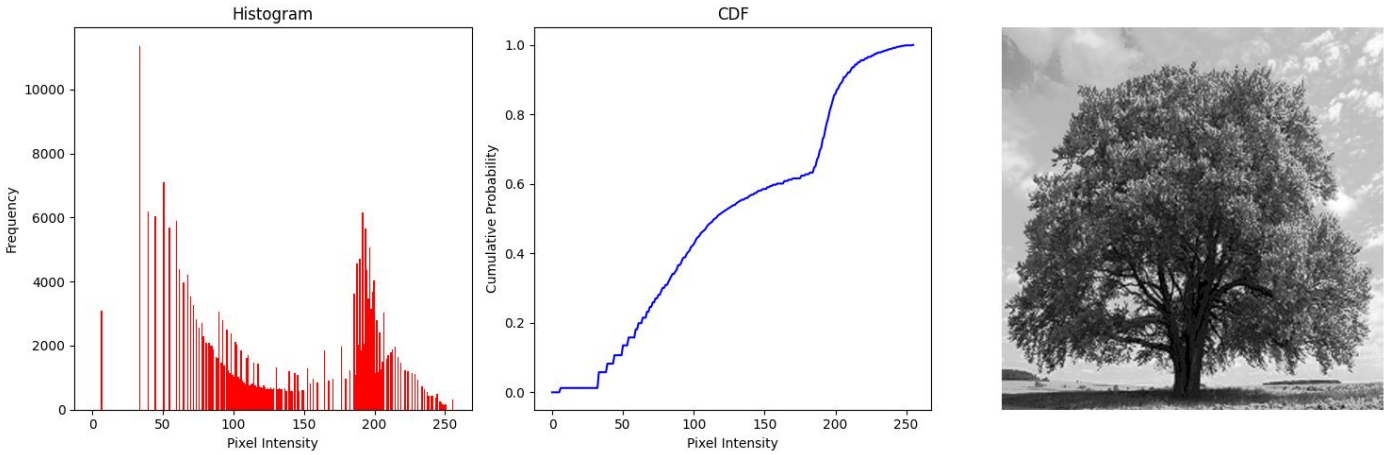
یکی دیگر از کاربرد های این پروژه رنگی کردن تصاویر خاکستری است. برای مثال فرض کنید یک تصویر خاکستری داریم و میخواهیم آن را رنگی کنیم. به این صورت عمل می کنیم که یک تصویر رنگی از همان domain را انتخاب میکنیم . برای مثال دو تصویر درخت زیر را داریم و میخواهیم تصویر سمت چپ را رنگی کنیم.


حال به این صورت عمل می کنیم که تصویر رنگی خود را نیز به صورت خاکستری در می آوریم. در تصاویر زیر تصویر خاکستری هر درخت، نمودار CDF آن و هیستوگرام آن مشخص است.



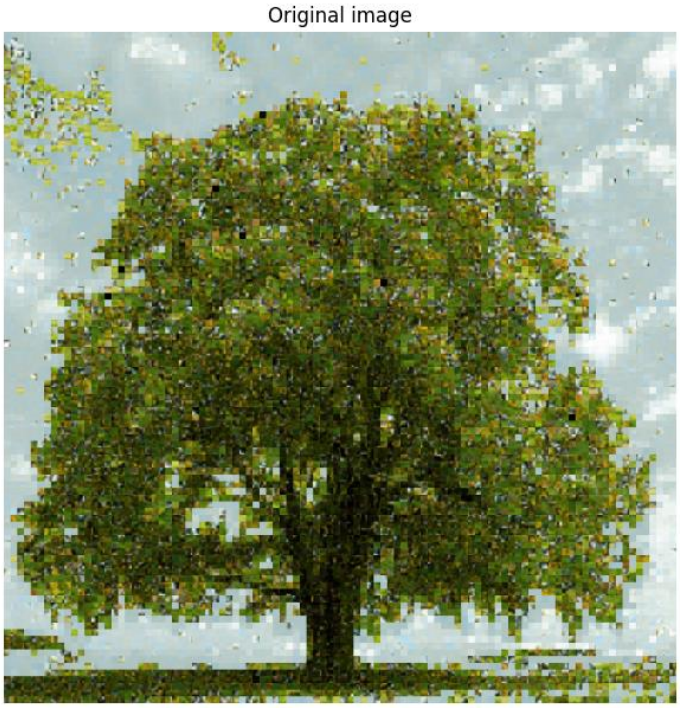



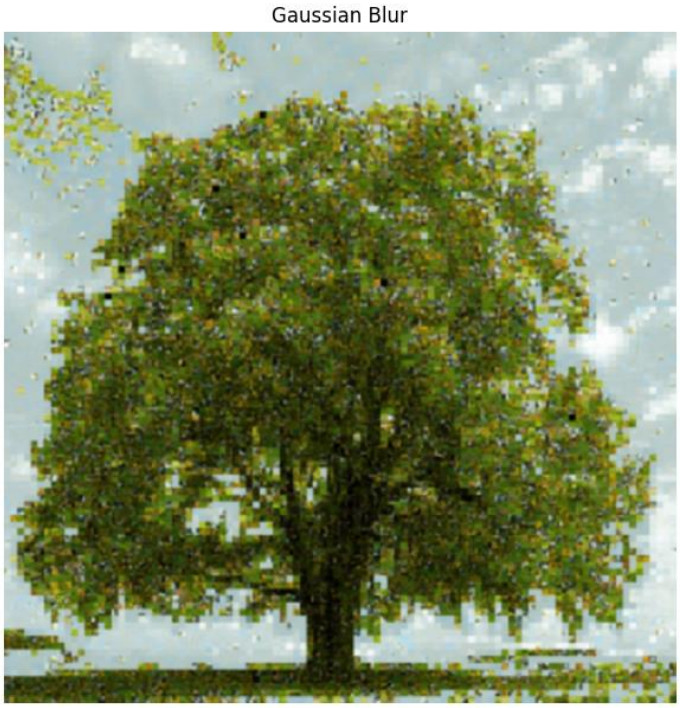

ما برای بهبود عملکرد الگوریتم خود ابتدا هیستوگرام تصویری که میخواهیم رنگی کنیم را با استفاده از الگوریتم Histogram Matching شبیه به هیستوگرام تصویری رنگی که داریم می کنیم. نتیجه آن را در تصویر زیر می بینید.




حال الگوریتم های pixel shuffle خود را روی تصاویر خاکستری اجرا می کنیم. یعنی تصویر خاکستری را با پیکسل های خاکستری شده تصویر رنگی میسازیم. حال ما ماتریس جابجایی پیکسل ها را داریم. حال این بار با استفاده از ماتریس جابجایی که بدست آمد پیکسل های رنگی را جابجا میکنیم و در نهایت به تصویر زیر می رسیم.

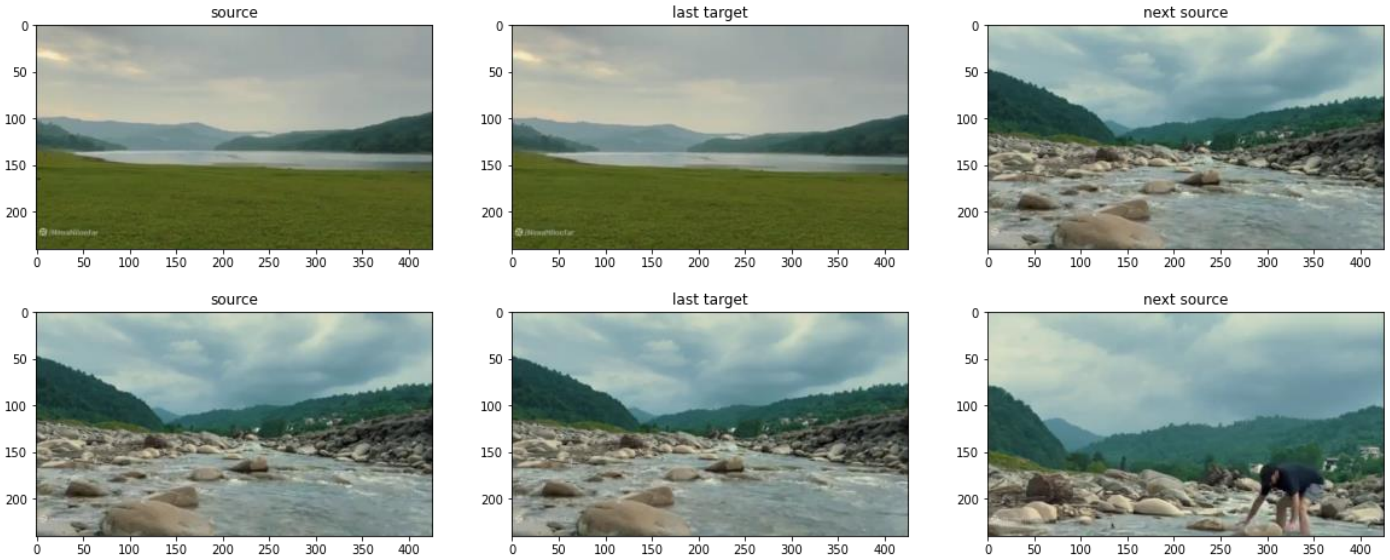
حال اگر دقت کنید نویز تصویر کمی بالاست و بلاک های آن مشخص است. برای رفع این مشکل میتوانم روی آن low pass filter استفاده کنیم که در تصویر نتیجه اعمال blur gaussian مختلف قابل مشاهده است.



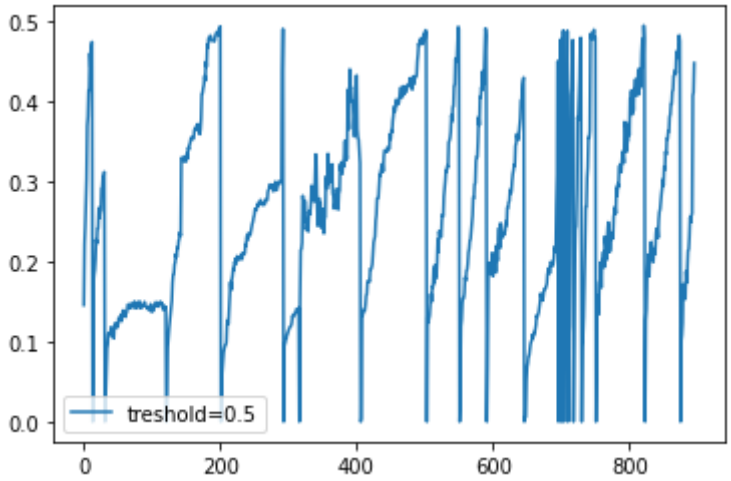



## 3.3. فشرده سازی ویدیو

کاربرد دیگری که این روش دارد این است که با استفاده از آن میتوان یک ویدیو را فشرده کرد. برای این منظور به این صورت عمل می کنیم ابتدا فریم های ویدیو را بر اساس نزدیکی histogram های آنها به هم دسته بندی میکنیم. یعنی به این صورت که ابتدا فریم اول به عنوان تصویر مبدا در نظر می گیریم و آن را در دسته اول می گذاریم و تا زمانی که اختلاف هیستوگرام های فریم های بعدی با فریم اول از یک حدی (threshold) بیشتر نشده باشد، آنها را به عنوان تصویر مقصد در همان دسته قرار می دهیم. حال زمانی که به فریمی رسیدیم که اختلاف هیستوگرام آن با تصویر مبدا دسته فعلی بیشتر از حد مشخص شده بود، یک دسته جدید میسازیم و آن فریم را به عنوان مبدا آن قرار می دهیم و همین فرآیند را مجددا تکرار می کنیم. در تصویرهای زیر به ترتیب از چپ به راست مبدا یک دسته ، آخرین عضو آن دسته و مبدا دسته بعد را می بینیم.




همچنین در تصویر زیر نمودار تغییرات اختلاف هیستوگرام فریم ها را در طول پیمایش روی آنها میبینم که مشخص است هر بار که به حد 0.5 میرسیم، تصویر مبدا و دسته عوض شده و اختلاف نزدیک صفر میشود.




حال فشرده سازی به این صورت انجام می گیرد، در هر دسته ، تصاویر را بر اساس پیکسل های تصویر مبدا آن دسته می سازیم و فقط ماتریس جابجایی را برای برای فریم های بعدی آن دسته نگهداری می کنیم. در واقع اگر تصاویر را بر اساس بلاک های 5*5 بسازیم ، هر بلاک 25 تایی از پیکسل های 3 کاناله به یک عدد int تبدیل میشود. برای مثال در ویدیو مثالی که الگوریتم را پیاده سازی کردیم، اطلاعات ویدیو اولیه شامل 275 میلیون بایت بود که پس از فشرده سازی اطلاعات به 21 میلیون بایت رسید که تقریبا 92 درصد فشرده سازی داشتیم.

## 4. جمع بندی

در این پروژه ما به بررسی ایده pixel-shuffling پرداختیم و الگوریتم آن را از لحاظ سرعت و کیفیت بهبود بخشیدیم. در ادامه بررسی کردیم که این ایده و روش چه کاربرد های جالب و مفیدی می تواند داشته باشد مانند تغییر دامنه رنگ تصویر، رنگی کردن تصویر سیاه و سفید و فشرده سازی ویدیو. محدودیتی که در این پروژه وجود داشت این بود که به دلیل سنگین بودن محاسبات اجرای الگوریتم روی تصویر های با رزولوشن بالا و اندازه بلوک های کوچک امکان پذیر نبود ولی در آینده میتوان بر بهبود عملکرد این الگوریتم ها بیشتر تحقیق و بررسی کرد. همچنین این ایده را می توان در زمینه های دیگری مانند رمزنگاری می توان به کار برد.

لینک دسترسی به کد های پروژه : https://github.com/abolfazlasad/pixel-shuffle.git

# 5. مراجع


[1] https://github.com/rishi1999/pixel-shuffle.git

[2] https://en.wikipedia.org/wiki/Histogram_matching

[3] https://en.wikipedia.org/wiki/Assignment_problem#Balanced_assignment

[4] D. F. Crouse, "On implementing 2D rectangular assignment algorithms," in IEEE Transactions on Aerospace and Electronic Systems, vol. 52, no. 4, pp. 1679-1696, August 2016, doi: 10.1109/TAES.2016.140952