بسمه تعالی

دانشکده مهندسی کامپیوتر

بینایی کامپیوتر

نام استاد: دکتر محمدی

تمرین سوم

نام دانشجو: آرمان حیدری

شماره دانشجویی: ۹۷۵۲۱۲۵۲

مهر ۱۴۰۱

فهرست

پاسخ سوال اول ۳

منابع ۴

پاسخ سوال دوم ۵

الف) ۵

ب) ۶

پ) ۶

ت) ۷

منابع ۸

پاسخ سوال سوم ۹

الف) ۹

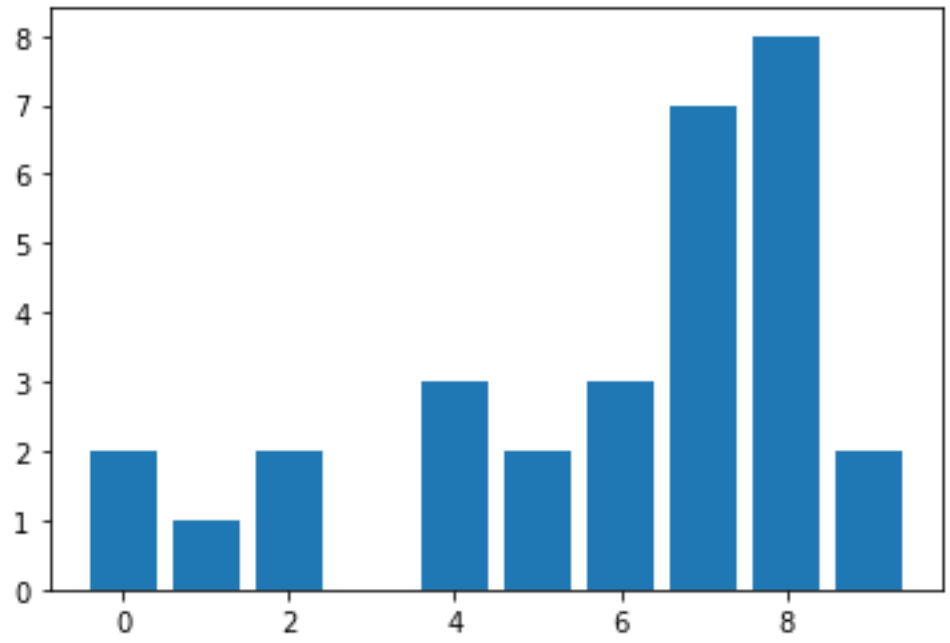
ب) ۹

پ) ۱۰

منابع ۱۱

پاسخ سوال اول

ابتدا هیستوگرام را ترسیم میکنیم:



شکل ۱: هیستوگرام تصویر فرضی

حالا تمام مقادیر را sort میکنیم تا ببینیم ۱۰ درصد بالایی و ۱۰ درصد پایینی چه اعدادی هستند:

0,0,1,2,2,4,4,4,5,5,6,6,6,7,7,7,7,7,7,7,8,8,8,8,8,8,8,8,9,9

یعنی باید ۳ عدد بالا و ۳ عدد پایین را کنار بگذاریم (۱۰ درصد ۳۰ برابر ۳ است). پس مقدار مینیمم جدید برابر ۲ و ماکسیمم برابر ۸ است. حالا با این فرمول مسئله را حل می کنیم:

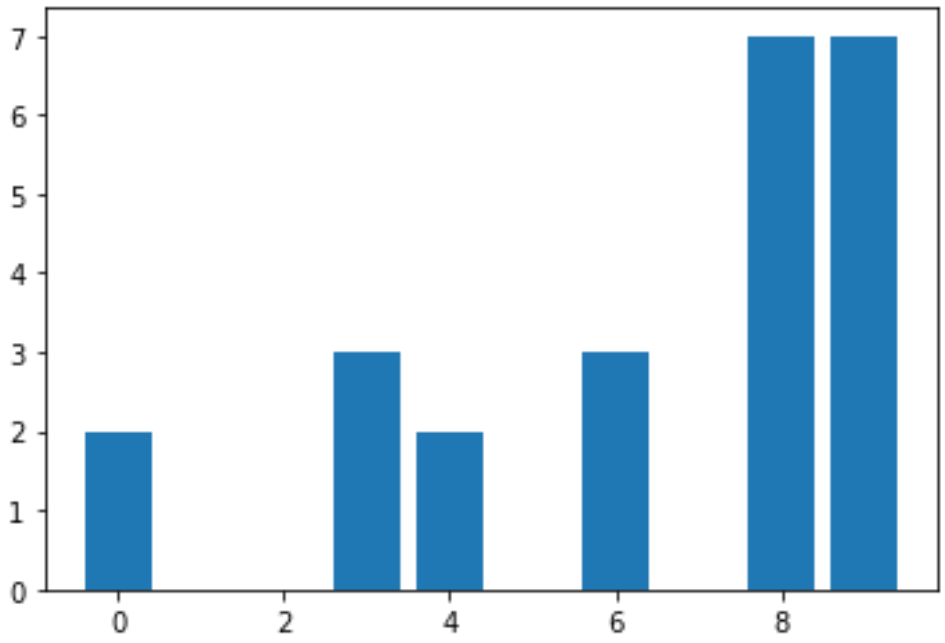
$$g(x,y) = stretch[f(x,y)] = \left(\frac{f(x,y) - f_{min}}{f_{max} - f_{min}}\right)(MAX - MIN) + MIN$$

MAX = 9, MIN = 0, $f_{min} = 2$, $f_{max} = 8$

با جایگذاری خانه های ۸۰ درصد وسط در این فرمول، و همچنین بدون تغییر گذاشتن ۱۰ درصد بالا و پایین به این تصویر میرسیم:

8	8	9	9	9	9
0	1	3	3	3	9
8	0	4	4	0	9
9	0	6	9	9	8
9	8	6	6	8	8

که هیستوگرام آن به این صورت خواهد بود:



شکل ۲: هیستوگرام پس از stretch

منابع

- ویدئو جلسه سوم کلاس

پاسخ سوال دوم

الف)

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

ابتدا با استفاده از رابطه بالا، خودم پیاده سازی را انجام دادم و شکل زیر حاصل شد.

شکل ۳: نتیجه پیاده سازی معادله histogram با numpy

و حالا با استفاده از تابع آماده از کتابخانه opencv پیاده سازی میکنیم.

شکل ۴: نتیجه پیاده سازی با opencv

حالت دوم نتایج به نظر بیشتر باعث ایجاد contrast شده اند. چون از فرمول های پیچیده تری به نسبت پیاده سازی خطی ما استفاده کرده است و کشش هیستوگرام با رابطه پیچیده تری انجام شده و بهتر نتیجه می دهد.

ب)

تابع CLAHE را با اندازه پنجره ۱۰ در ۱۰ که در نوتبوک معلوم شده بود و 5=THRESHOLD برای contrast limit اجرا می کنیم. و خروجی به این صورت می شود:

شکل ۵: نتیجه اعمال CLAHE

پ)

خروجی ها به همان ترتیب قبل:

شکل ۶: نتیجه کشش هیستوگرام خطی برای City.jpg

شکل ۷: نتیجه کشش هیستوگرام با opencv برای City.jpg

شکل ۸: نتیجه اعمال CLAHE برای City.jpg

ت)

خیر این روش درست نیست. تصاویر RGB از سه کانال تشکیل شده است. بنابراین به طور طبیعی روشی که به ذهن می رسد، این است که می توانید آنها را به سه کانال جداگانه تقسیم کنید، تساوی هیستوگرام را روی هر کدام اعمال کنید و سپس آنها را دوباره با هم ترکیب کنید. اما یکسان سازی هیستوگرام غیر خطی یک فرآیند غیر خطی است. تقسیم کانال و یکسان سازی هر کانال به طور جداگانه نادرست است. یکسان سازی شامل مقادیر شدت تصویر است، نه اجزای رنگ. بنابراین برای یک تصویر رنگی RGB ساده، یکسان سازی هیستوگرام را نمی توان مستقیماً روی

کانال ها اعمال کرد. باید به گونه ای اعمال شود که مقادیر شدت هر رنگ بدون برهم زدن تعادل رنگ تصویر برابر شوند.

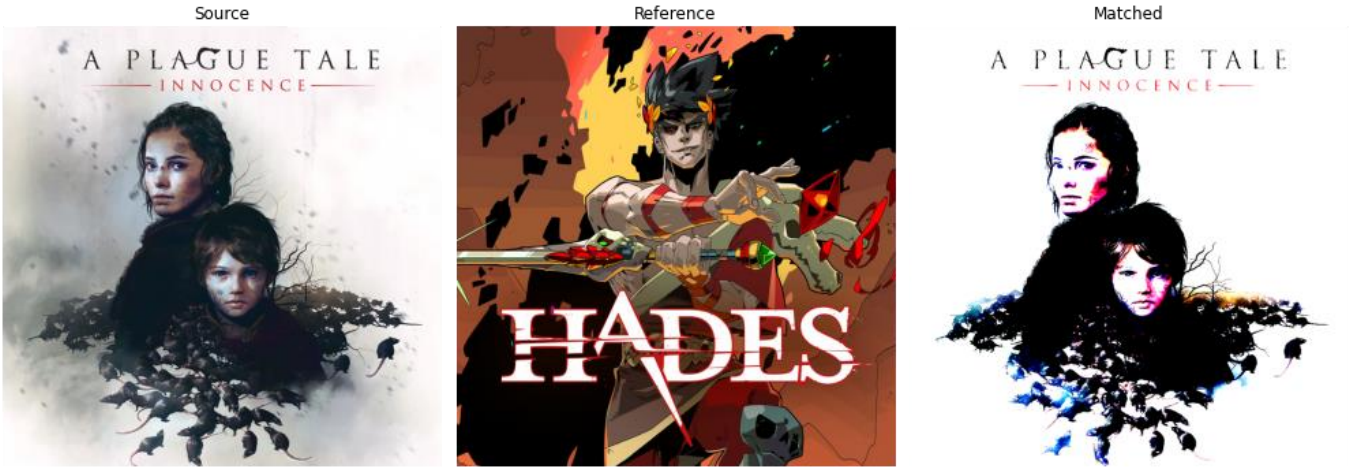
بنابراین، اولین قدم تبدیل فضای رنگی تصویر از RGB به یکی از فضاهای رنگی است که مقادیر شدت را از اجزای رنگ جدا می کند. با روشی به نام YCbCr می توانیم این کار را انجام دهیم. که روی محور Y که ترکیب سه رنگ است روش های مختلف کشش هیستوگرام را اعمال می کنیم و بعد آن را مجددا به RGB باز میگردانیم.

منابع

- اسلاید جلسه سوم کلاس
- https://docs.opencv.org/4.x/d5/daf/tutorial_py_histogram_equalization.html
- https://www.geeksforgeeks.org/clahe-histogram-eqalization-opencv/
- https://prateekvjoshi.com/2013/11/22/histogram-equalization-of-rgb-images/#:~:text=Histogram%20equalization%20is%20a%20non,applied%20directly%20on%20the%20channels.

پاسخ سوال سوم

الف)

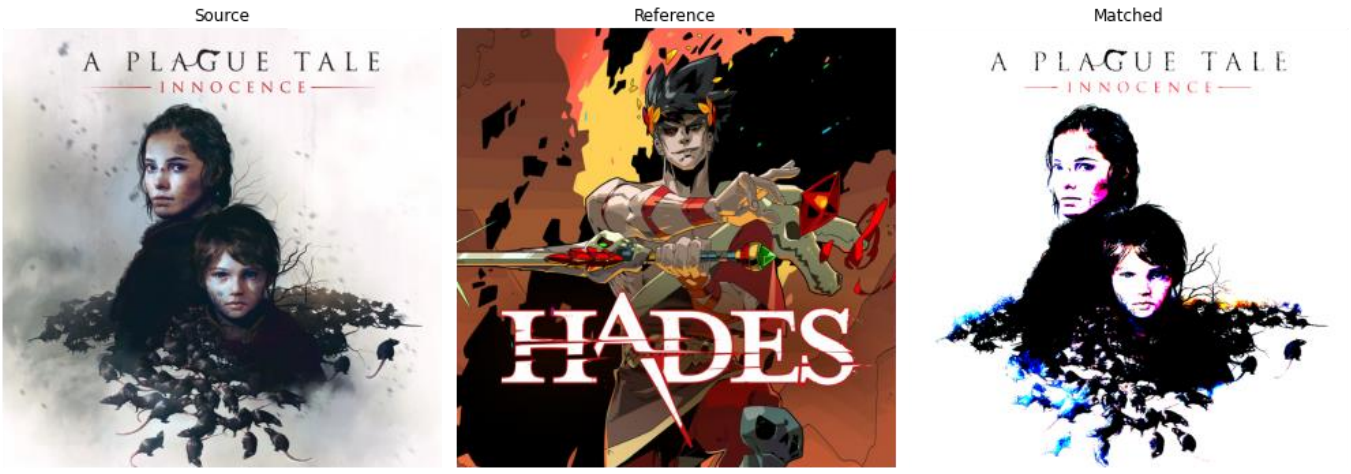


شکل ۹: تطبیق هیستوگرام یک تصویر با تصویر دیگر با تابع آماده کتابخانه scikit-image

میبینیم که رنگ های مربوط به تصویر a plague tale، تغییر کرده اند و به تصویر hades نزدیک شده اند.

ب)

ابتدا تابعی مینویسیم که این کار را برای یک بعد انجام دهد (در واقع برای تصاویر grayscale است) و بعد این کار را برای هر کدام از ۳ کانال تصویر رنگی به صورت جداگانه انجام می دهیم. در تابع نوشته شده، ابتدا چگالی احتمال تجمعی را برای تصویر source و reference محاسبه می کنیم و بعد برای هر میزان روشنایی در تصویر source، نزدیک ترین میزان روشنایی در مقدار تجمعی تصویر reference را حساب کرده و جایگزین آن می‌کنیم.



شکل ۱۰: تطبیق هیستوگرام تصاویر با تابع نوشته شده توسط خودم

میبینیم که خروجی دقیقا مشابه تابع آماده قسمت قبل است. یعنی پیاده سازی درست بوده است.

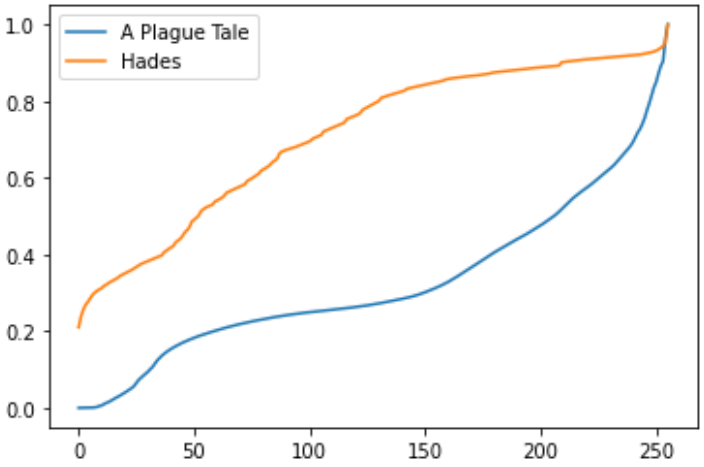
پ)

همان دو تابع قبلی را به سادگی با جابجایی source و reference صدا میزنیم و خروجی میگیریم.

شکل ۱۱: برعکس کردن source و reference با تابع آماده

شکل ۱۲: برعکس کردن source و reference با تابع خودم

بازهم خروجی ها مشابه است که یعنی تابع درست است. میبینیم تصویر حاصل تقریبا تماما سفید است و با بررسی مقادیر میبینیم که اکثرا بالای ۲۵۰ هستند. برای توجیه این موضوع میتوانیم توابع توزیع تجمعی آن ها را بررسی کنیم:



شکل ۱۳: مقایسه cdf تصاویر source و reference

همانطور که از شکل مشخص است، تصویر source اگر با reference منطبق شود، تقریبا مقادیر روشنایی بالای ۵۰ آن، با مقادیر بیش از ۲۲۰ جایگزین می شوند! از شیب نمودار نارنجی هم می توان فهمید که عمده پیکسل های تصویر source بیش از ۵۰ هستند و عملا همگی سفید می شوند و شکل ۱۱ و ۱۲ توجیه می شوند.

منابع

- https://stackoverflow.com/questions/32655686/histogram-matching-of-two-images-in-python-2-x
- https://scikit-image.org/docs/stable/api/skimage.exposure.html#skimage.exposure.match_histograms