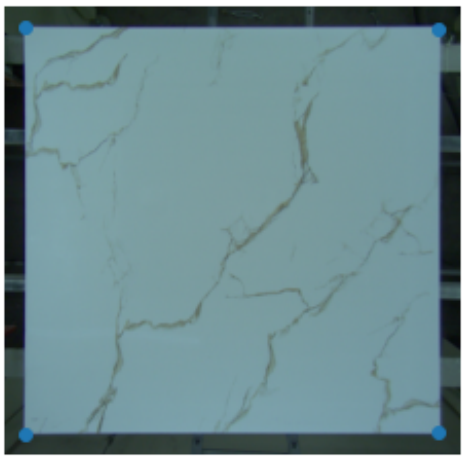
گزارش پروژه درس بینایی کامپیوتر

مهدی‌امیری‌شوکی۹۸۵۲۲۱۴۸ – فاطمه زهرا بخشنده۹۸۵۲۲۱۵۷

زمستان ۱۴۰۱

آماده‌سازی و پیش‌پردازش داده‌ها

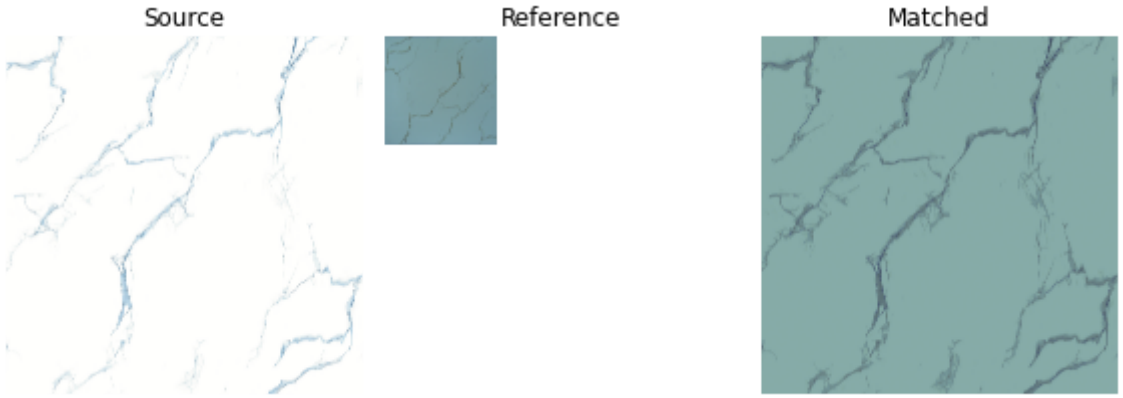


بریدن دور کاشی

برای بریدن اضافات دور کاشی ابتدا تصویر را سیاه‌وسفید کردیم و بعد از اعمال فیلتر blur و استفاده از لبه‌یاب Canny، کانتورهای موجود در نتیجه‌ی لبه‌یابی را یافته و سپس بزرگترین کانتور ۴تایی را به عنوان ۴ گوشه‌ی کاشی در نظر گرفته و با توجه به این ۴ راس بدست‌آمده دور کاشی را حذف کردیم.

تطبیق هیستوگرام تصویر و طرح

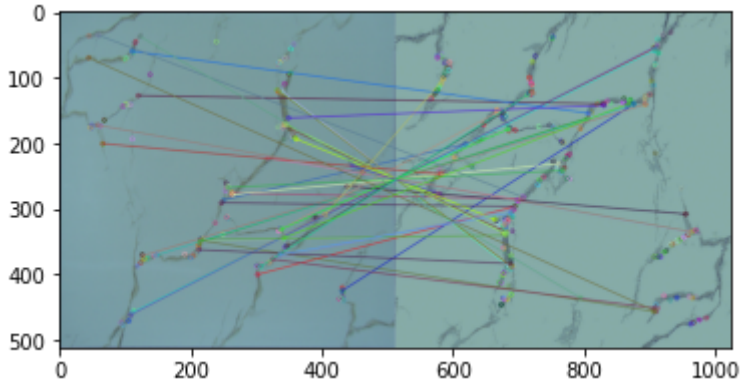
برای گرفتن نتیجه بهتر در مقایسه تصویر و طرح، رنگ‌های تصویر و طرح را از طریق تکنیک تطبیق هیستوگرام نزدیک کردیم. ابتدا رنگ‌های تصویر را به رنگ‌های طرح نزدیک کردیم که مشاهده کردیم تمایز رنگ‌های ترک تا حدودی از بین می‌رود به همین دلیل در نهایت رنگ‌های طرح را به رنگ‌های تصویر داده‌شده نزدیک کردیم تا تمایز ترک از بین نرود.



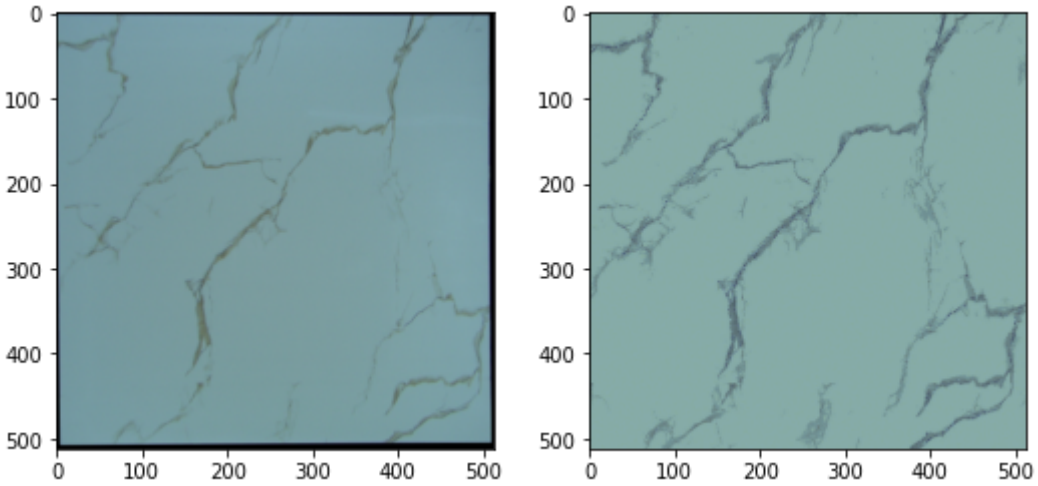
تطبیق مکانی تصویر و طرح

برای پیداکردن تبدیل هندسی از تصویر به طرح، از روش پیداکردن نقاطکلیدی با الگوریتم ORB استفاده کردیم. ORB علاوه بر اینکه سریعتر از SIFT است، بدلیل اینکه چرخش‌های تصویر مضربی از ۹۰درجه است به لحاظ عملکرد هم در این کاربرد خاص بهتر از SIFT جواب می‌دهد. سپس از BFMatcher خود OpenCV برای تطبیق نقاط کلیدی مشترک استفاده کردیم و در نهایت بعد از پیداکردن تبدیل هندسی موردنظر تصویر کاشی را به طرح منطبق کردیم.

پیدا کردن نقاط کلیدی قبل از چرخش:



بعد از اینکه ۱۸۰درجه چرخیده شده تا تصویر و کاشی منطبق شوند:



تغییر سایز

هم به جهت یکی کردن ابعاد کاشی و طرح، و هم به جهت کم کردن ابعاد تصاویر ورودی به شبکه ابعاد کاشی و طرح را به ۵۱۲x۵۱۲ تغییر دادیم.

تبدیل لیبل‌های داده‌شده به لیبل‌های مناسب برای segmentation

از مختصات نقاط داده شده برای Bounding Box ترک‌ها، یک مسیر بسته ایجاد کردیم و آن را بصورت یک لیبل‌گذاری باینری برای پیکسل‌ها تبدیل کردیم بطوری که تمام پیکسل‌های داخل ناحیه بسته 1 و بقیه پیکسل‌ها ۰ باشند تا بتوانیم در segmentation به‌عنوان لیبل استفاده کنیم.

```
# Debug
print(x_train[0].shape)
print(y_train[0])
print(np.where(y_train[0] == True))
```

```
(1024, 1024, 6)
[[False False False ... False False False]
 [False False False ... False False False]
 [False False False ... False False False]
 ...
 [False False False ... False False False]
 [False False False ... False False False]
 [False False False ... False False False]]
(array([432, 432, 432, ..., 515, 515, 515]), array([311, 312, 313, ..., 225, 226, 227]))
```

جداکردن داده‌های test و train

ابتدا همه‌ی داده‌ها (عکس‌ها و لیبل‌ها) بصورتی که تناظر بین‌شان به هم نخورد شافل می‌شوند و سپس ۸۵ درصد داده‌ها به عنوان دیتای train و بقیه به عنوان دیتای test مورداستفاده قرار می‌گیرند.

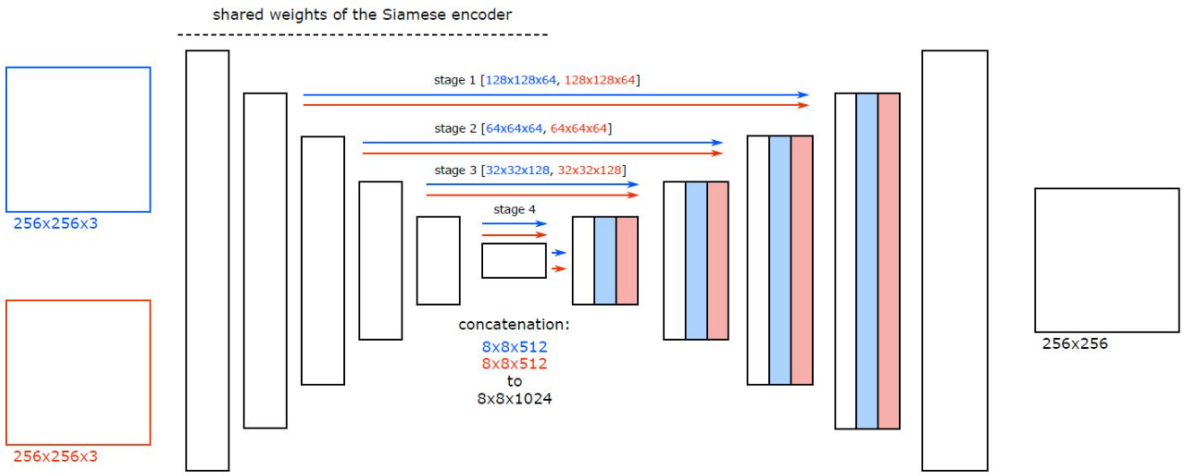
```
# pick train_set/test_set
print(len(input1_list))
cut_point = int(0.85 * len(input1_list))
x1_train = np.array(input1_list[:cut_point])
x2_train = np.array(input2_list[:cut_point])
x1_test = np.array(input1_list[cut_point:])
x2_test = np.array(input2_list[cut_point:])
y_train = np.array(label_list[:cut_point])
y_test = np.array(label_list[cut_point:])
```

ساختار شبکه و مدل یادگیری

انتخاب شبکه:

از آنجایی که با تسک segmentation روبرو هستیم، از شبکه Unet به عنوان base مدل خود، استفاده می کنیم. که برای تسک های semantic segmentation مدل محبوبی است. این شبکه به دلیل استفاده از فیچر های رزولوشن بالا در کنار فیچر های با عمق بالا، می تواند به دقت خیلی خوبی در اینگونه تسک ها برسد.

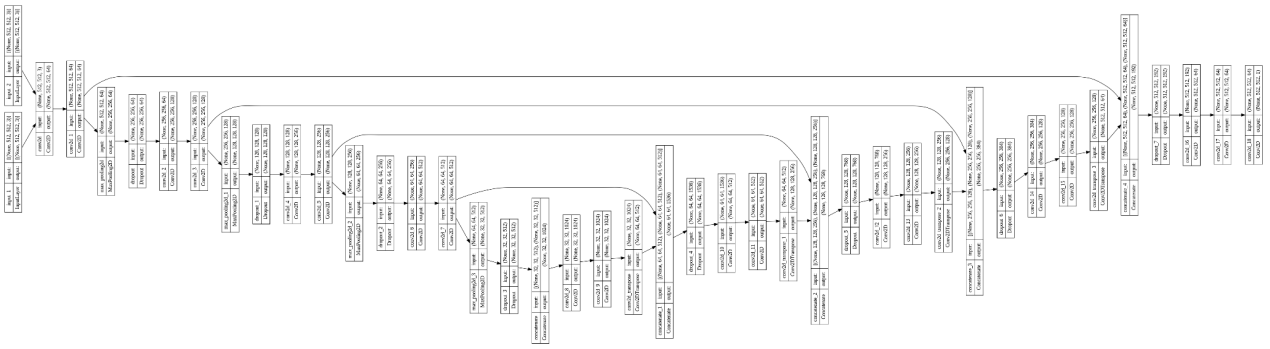
علاوه بر این، چون تسک ما یک نوع تسک change detection است، مدل ما باید دو ورودی تصویر و الگو را بگیرد. که می توان به شکل های مختلفی این دو ورودی را به شبکه داد. روشی که ما پیاده سازی کردیم یک شبکه Siam_Unet است که ساختار آن به صورت زیر است:



در واقع این شبکه دو ورودی تصویر دارد. سپس بخش encoder شبکه به طور مشترک روی هر دو تصویر اعمال می شوند. سپس خروجی ها concat شده و وارد بخش decoder می شوند. در این بخش، خروجی هر Conv2DTranspose، با خروجی رزولوشن بالاتر حاصل از تصویر و همچنین پترن، concat می شود. و به این صورت up_sampling نیز پیاده سازی می‌شود.

خروجی این شبکه، یک mask یک کاناله از تصویر می باشد، که بخش هایی که در تصویر به عنوان ترک مشخص شده اند، باید در آن 1 شوند و بقیه بخش ها باید 0 باشند. روش ساختن mask در بخش قبل توضیح داده شد.

شکل مدل:



انتخاب تابع ضرر:

انتخاب loss function برای این شبکه باید به دقت انجام شود. باید از یک loss مناسب تسک segmentation استفاده کنیم که تفاوت خروجی شبکه با mask را بسنجد و طبق آن loss را تعیین کند. در نتیجه از dice loss استفاده می کنیم. همچنین چون در یک تصویر، تعداد پیکسل های ترک بسیار کمتر از تعداد پیکسل های دیگر است، در نتیجه برای جبران این unbalance بودن، از تابع ضرر `weighted_bce_dice_loss`

استفاده می کنیم. که ترکیب weighted binary cross entropy و dice loss می باشد.

توابع loss دیگری نیز در نوت بوک پیاده سازی شده اند و می توان از آن ها نیز استفاده کرد، اما تابع loss ای که ما انتخاب کردیم، ترکیب این توابع می باشد و بهتر است.

انتخاب معیار ارزیابی:

برای انتخاب metric، باید دقت کنیم در هر خروجی (mask)، تعداد نمونه های کلاس مثبت بسیار کمتر از نمونه های کلاس منفی است. در نتیجه، استفاده از معیار accuracy برای این تسک، چندان مناسب نیست. و باید معیار های دیگری را همچون recall ،precision و f_score در نظر بگیریم.

همچنین تسک ما یک تسک عادی classification نیست، و یک تسک segmentation است که در آن، مشابهت تصویر خروجی شبکه با تصویر mask، اهمیت دارد. پس از معیار هایی مانند dice_coef و binary_iou نیز برای ارزیابی خروجی استفاده می کنیم.

در نهایت شبکه را با loss function و metric های ذکر شده، compile کرده و روی دیتاستی که از پیش ساختیم، train می کنیم.

چالش‌ها

تابع loss:

در اولین train شبکه، از تابع ضرر dice ساده استفاده کردیم. نتیجه این بود که f_score مقدار خیلی کمی شد، و پس از خروجی گرفتن predict شبکه، متوجه شدیم که شبکه سعی میکند همه پیکسل ها را 0 تشخیص دهد. تا مشابهت زیادی با mask داشته باشد. اما بخش ترک که 1 است و بخش کوچکی است، مهم نبود.

راه حل: برای بهتر شدن نتیجه و متعادل کردن کلاس ها در تابع ضرر، از focal loss و weighted_bce_dice_loss استفاده کردیم، و f_score افزایش یافت.

مشکل پر شدن RAM:

در ابتدا، اندازه تصاویر را به 1024، resize کرده بودیم، و batch_size را هم مقدار زیادی گرفته بودیم. و برای train، هر دفعه به ارور پر شدن RAM بر می خوردیم.

راه حل: اندازه تصاویر، یعنی ورودی های مدل را کوچکتر انتخاب کرده، و batch_size را هم مقدار کمتری در نظر گرفتیم. و مشکل برطرف شد.

ایده: ایده جایگزین برای حل این مشکل می تواند بریدن تصویر به بخش های کوچکتر، و train هر کدام از بخش ها باشد. یعنی به جای اینکه تصویر را کوچکتر کنیم. تصاویر بزرگ را به بخش هایی تقسیم کنیم تا مدل در هر بخش، به دنبال ترک بگردد.

ایده مورفولوژی:

ایده دیگر برای بالا بردن دقت مدل برای پیش بینی ترک ها، استفاده از عملگر های مورفولوژی است. مثلا می توانیم هم در تصویر ورودی و pattern، و هم در mask، عملگر هایی انجام دهیم که ترک ها مشخص تر شده و شبکه راحتتر از آن ها ویژگی استخراج کند. همچنین تعداد پیکسل های کلاس مثبت بیشتر شود.

ایده عوض کردن concate در بخش میانی شبکه:

چون تسک ما Change Detection است و میخواهیم به تفاوت دو تصویر، که همان ترک ها هستند برسیم، می توانیم لایه concate میانی شبکه Siam Unet را تغییر دهیم و بجای concate، دو خروجی را از هم کم کنیم.

منابع: لینک و لینک