گزارش پروژه

بینایی ماشین

یاسمین مدنی

ملیکا نوبختیان

فهرست

مقدمه

برای انجام دادن پروژه لازم بود تا چند قدم اولیه را طی کنیم.

ابتدا داده ها و پترن ها را از لحاظ نوع و کیفیت تصویر و ویژگی های آن بررسی کردیم که در این مرحله متوجه شدیم داده ها از لحاظ رنگی و شدت نور و کیفیت با پترن های موجود تفاوت زیادی دارند لذا نتیجه گرفتیم برای هرگونه راه حل احتمالی ابتدا پیش پردازشهایی برای بهبود تصویر نیاز داریم.پس از آن به ایده پردازی پرداختیم و راه حل های احتمالی خود را مکتوب کردیم که در ادامه آورده شده اند.

استخراج کاشی از تصویر ، پیدا کردن تبدیل تصویر ایده ای بود که در هر راه حل احتمالی به آن نیاز داشتیم.پس از آن در مورد شبکه ی عصبی مورد استفاده خود به چند مدل متفاوت رسیدیم.یکی استفاده از پترن ها و کم کردن طرح آنها از کاشی اصلی بود،دیگری استفاده از شبکه های عصبی siamese بود و دیگر استفاده از مدل هایی برای سگمنت کردن تصویر مانند U-net و....

در مرحله اولیه تصمیم گرفتیم تا روش کم کردن پترن از عکس را امتحان کنیم که نتایج خوبی از آن به دست نیامد از این رو کار را با شبکه های چند ورودی ادامه دادیم.

در ادامه به توضیح هر یک از مراحل میپردازیم.

جدا کردن تصویر کاشی از بک گراند

برای آن‌که بتوانیم تصویر و الگوها را بر هم منطبق کنیم و به علاوه در بخش محاسبه نقاط کلیدی نقاط ارزشمندتری پیدا کنیم نیاز است تا حاشیه‌ی تصویر را حذف کنیم .

برای انجام این کار نیز به سه روش مختلف پیش رفتیم و روش بهتر را از میان آنها استخراج کردیم.

با نقاط گوشه ای تصویر

در این روش ابتدا سعی کردیم تا چهار گوشه ی تصویر را مشابه تمرین ۷ استخراج کنیم. که این روش نتیجه‌ی چندان جالبی را به دست نیاورد و حدود ۱۰۸ داده از ۳۰۸ داده کل (تقریبا یک سوم داده ها) را نا مناسب استخراج میکرد.

با در نظر گرفتن نقطه ثابت

در این مرحله تصمیم گرفتیم از آنجا که کاشی ها تقریبا در وسط صفحه واقع شده اند با در نظر گرفتن نقاط گوشه ای به دست آمده از یکی از کاشی‌هایی که در مرحله قبل به طور مناسبی جدا شده بود

سایر کاشی ها را جدا کنیم. برای این کار یکبار نقاط به دست آمده از روش قبل را با لیبل زدن گوشه ها و مختصات آنها در labelme مقایسه کردیم و بهترین گزینه را انتخاب نمودیم. که این روش باعث بهبود وضعیت نسبت به حالت قبل شد اما هنوز بخشی از حاشیه سیاه برای برخی تصاویر باقی مانده بود.

نقاط به حالت زیر:

points = [[75, 79], [1633, 1605]]

با مورفولوژی

ابتدا تصویر را به خاکستری تبدیل کردیم پس از آن الگوریتم اوتسو را بر روی آن اعمال کرده و سپس آن را با یک واحد ساختاری ۳۹*۳۹ کلوز میکنیم تا لبه ها را به طور واضحتر و پیوسته تر به‌دست آوریم.سپس روی تصویر بدست آمده لبه یاب canny را اجرا می کنیم .در مرحله‌ی بعد با تابع findcountors کانتور ها را به دست آورده و برای آنها بیشترین مساحت را محاسبه میکنیم و کانتور با مساحت بیششینه را به تابع approxPolyDP میدهیم تا نقاط گوشه ای را برگرداند.

در مرحله آخر تابع crop_out را جهت برش نقاط از روی تصویر اصلی آماده کردیم که در ادامه توضیح میدهیم که هم تصویر هم ماسک آن را با یک مختصات کراپ میکنیم.

این روش به مراتب بهتر از روش های دیگر بود و تنها چند مورد نامناسب را تولید کرد.

محاسبه ی ماسک تصاویر

در این مرحله ماسک تصویر را نیز محاسبه کردیم.محاسبه ی ماسک تصویر به این معنی که نقاطی که در تصویر ترک بوده اند را برابر ۱ و نقاط غیر ترک را برابر ۰ قرار دهیم در مرحله آخر برای انکه بتوانیم همواره ناحیه‌ی درست ترک ها و مختصات آنها را که در فایل json هر تصویر داده شده محاسبه کنیم به این ماسک نیاز داریم.ایده کلی این است که ماسک هر تصویر را نیز با نقاط به دست آمده برای آن تصویر از مرحله ی قبل(پیدا کردن گوشه‌های کاشی)نیز کراپ کنیم و به این روش مختصات نسبی ترک ها را نسبت به تصاویر کراپ شده داریم.

محاسبه و انطباق نقاط کلیدی

پیش پردازش تصویر

برای محاسبه نقاط کلیدی با توجه به تصاویر درمیابیم که نیاز داریم تا کمی از تفاوت نور و رنگ و کیفیت الگوها و کاشی ها را بکاهیم.

از این رو اعمال زیر را روی تصویرمان انجام میدهیم :

- هیستوگرام مچینگ بین دو تصویر
- افزایش کنتراست با استفاده از کلاهه . از آنجا کنتراست بعد از هیستوگرام مچینگ کاهش پیدا کرده بود.در این بخش از فضای رنگی LABکه کامل‌ترین فضای رنگی ممکن است استفاده کردیم و کانال مورد نظر ما L بود که مربوط به شدت روشنایی است.
- به علاوه تصاویر را به حالت gray نیز تبدیل کردیم تا تفاوت رنگی مشکل جدی ای ایجاد نکند.
- لازم به ذکر است اعمال فیلتر بلر روی تصاویر الگو نیز امتحان کردیم اما تفاوت چشم‌گیری نسبت به حالت قبل را نتیجه نداد.
- توجه داریم که پیش از این الگو هارا ریسایز کرده ایم به طوری اسکیل آنها برهم نریزد.

انطباق نقاط کلیدی

Optical flow

یکی از راه های دیگری که امتحان کردیم روش sparse optical flow بود که ایده این است که در دو تصویر نقاط یا ویژگی های خوب برای دنبال کردن را با تابع goodFeaturesToTrack پیدا کند و بر اساس آن تبدیل را میچرخاندیم اما صادقانه نتوانستیم استفاده از این روش را برای تنها دو تصویر که الگو و کاشی باشد اجرا کنیم و تمامی آموزش های مربوط به فریم های یک ویدیو بود که در اسکیل ۲ تصویر جوابی حاصل نشد.

SIFT

برای به دست آوردن و مچ کردن نقاط کلیدی تصاویر از الگوریتم SIFT استفاده کردیم. و نتایج حاصله را به تابع findHomography داده ایم تا برای ما ماتریس تبدیل را به دست آورد و داده‌های پرت آن را با الگوریتم RANSACحذف میکنیم.در مرحله بعد با استفاده از warpPerspective و ماتریس تبدیل به دست‌آمده از مرحله قبل الگوها را به حالت مناسب طبق تبدیل میچرخانیم.

حذف پترن از کاشی(روش ناموفق)

در این بخش ایده این بود که الگویی که چرخش پیدا کرده و مطابق تصویر اصلی ما شده از تصویر کم کنیم تا طرح کاشی حذف شود و تنها ترک ها و بخش جزیی از نویز باقی بماند که این ایده با شکست مواجه شد چرا که تصاویر به طور مناسبی بر الگوها منطبق نمیشد و تفاضل آنها نقاط نا مناسبی را از یکدیگر می‌کاست.

شبکه با ورودی چندتایی و آماده سازی دیتای آن

پس از آنکه روش های قبلی را امتحان کردیم به مرحله انتخاب شبکه عصبی میرسیم.برای اینکار تصمیم گرفتیم که از شبکه Siamese استفاده کنیم اما با این تفاوت که به جای ورودی دادن یک تصویر بزرگ به شبکه از ایده‌ی sliding window استفاده کردیم.

تابع sliding_window براساس‌پارامترهای اندازه گام واندازه پنجره نقاط گوشه بالا سمت چپ و گوشه پایین سمت راست را به ما باز میگرداند.که از آن برای قطعه کردن الگو، ماسک و تصویر اصلی استفاده میکنیم.

```
def sliding_window(image, stepSize, windowSize):
    for y in range(0, image.shape[0], stepSize):
        for x in range(0, image.shape[1], stepSize):
            if y + windowSize[1] < image.shape[0] and x + windowSize[0] <
image.shape[1]:
                yield [[x, y],[x + windowSize[0], y + windowSize[1]]]
```

در اینجا اگر مجموع نقاط روشن موجود در یک ویندو از ماسک بیشتر از ۰ باشد به این معنی است که پنجره در ناحیه ترک‌ها بوده است و آن را به قسمت داده های ترک دار اضافه میکنیم. درغیر این صورت ویندوها شامل طرح یا پس زمینه اند.

```
step_size = 50
window_size = (50, 50)
tereshhold=0
train1_fractured = []
train2_fractured = []
train1_without_fracture = []
train2_without_fracture = []
for name, img in cropped_ims.items():
    mask = cv2.imread(mask_path + name)
    pat_name = dataset_info[name]['pattern'].split('.')[0]
    pat_t_name = pat_name + "_" + name
    pat = cv2.imread(transformed_pattern_path + pat_t_name)
    if pat is None:
```

```
            continue
        for win_coordinates in sliding_window(img, step_size, window_size):
            x1=win_coordinates[0][0]
            y1=win_coordinates[0][1]
            x2=win_coordinates[1][0]
            y2=win_coordinates[1][1]
            masked_win = mask[y1:y2,x1:x2]
            pattern_win = pat[y1:y2,x1:x2]
            image_win = img[y1:y2,x1:x2]
            win_sum=np.sum(masked_win)
            if win_sum > tereshhold :
                train1_fractured.append(image_win)
                train2_fractured.append(pattern_win)
            else:
                train1_without_fracture.append(image_win)
                train2_without_fracture.append(pattern_win)
```

با استفاده از همین دیتاست هایی که تشکیل میدهیم لیبل مخصوص جفت ها را هم انتخاب میکنیم و هر ویندو از تصویر اصلی را با ویندو آن از الگو به صورت یک جفت به شبکه میدهیم.

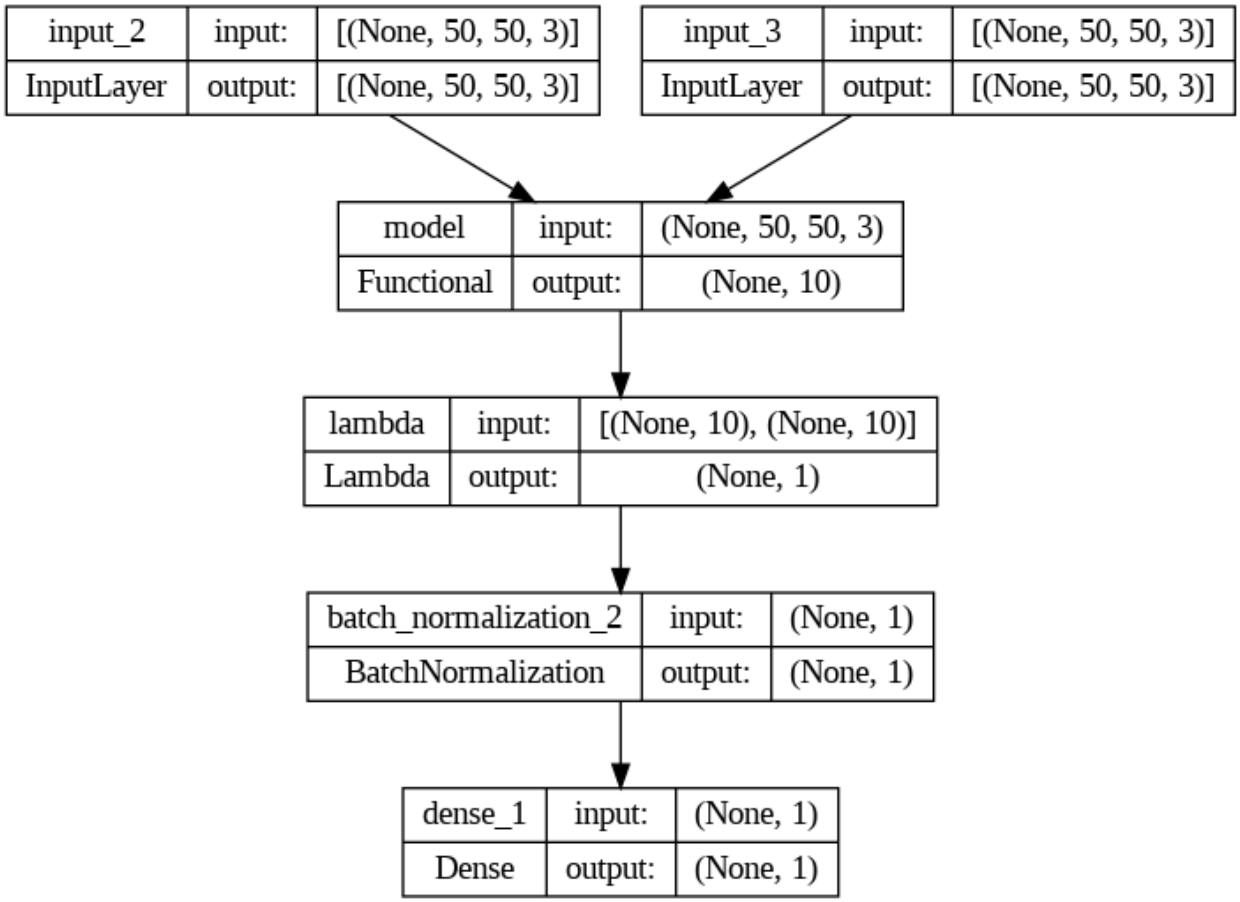
```
train1_t = train1_fractured + less_1
train2_t = train2_fractured + less_2
labels = []
for i in range(len(less_1)):
    labels += [1]

for i in range(len(train1_fractured)):
    labels += [0]
```

از آنجا که نسبت ویندوهایی که حاوی ترک اند نسبت به ویندوهایی که ترک ندارن خیلی کمتر است اگر تمامی ویندوها را در اختیار شبکه قرار دهیم موجب اور فیت شدن شبکه میشوند از این رو از داده های بخش به اصطلاح سفید یا بدون طرح نمونه برداری میکنیم و تقریبا هم اندازه با داده های ترک دار از آنها نمونه میگیریم.

```
fracture_less = list(zip(train1_without_fracture, train2_without_fracture)
)
less_1=[]
less_2=[]
for a,b in random.sample(fracture_less, 3000):
    less_1.append(a)
    less_2.append(b)
```

ساختار شبکه عصبی نیز به صورت شکل زیر است


input_2 | input: | [(None, 50, 50, 3)]
InputLayer | output: | [(None, 50, 50, 3)]
input_3 | input: | [(None, 50, 50, 3)]
InputLayer | output: | [(None, 50, 50, 3)]
model | input: | (None, 50, 50, 3)
Functional | output: | (None, 10)
lambda | input: | [(None, 10), (None, 10)]
Lambda | output: | (None, 1)
batch_normalization_2 | input: | (None, 1)
BatchNormalization | output: | (None, 1)
dense_1 | input: | (None, 1)
Dense | output: | (None, 1)

نتایج به‌دست آمده

```
Epoch 1/10
127/127 [==============================] - 14s 101ms/step - loss: 0.2362 - accuracy: 0.6781 - f1: 0.7787 - val_loss: 0.1520 - val_accuracy: 0.9941 - val_f1: 0.0000e+00
Epoch 2/10
127/127 [==============================] - 12s 91ms/step - loss: 0.2163 - accuracy: 0.7101 - f1: 0.8078 - val_loss: 0.1063 - val_accuracy: 1.0000 - val_f1: 0.0000e+00
Epoch 3/10
127/127 [==============================] - 12s 91ms/step - loss: 0.1981 - accuracy: 0.7309 - f1: 0.8266 - val_loss: 0.0936 - val_accuracy: 0.9663 - val_f1: 0.0000e+00
Epoch 4/10
127/127 [==============================] - 11s 90ms/step - loss: 0.1843 - accuracy: 0.7413 - f1: 0.8349 - val_loss: 0.3123 - val_accuracy: 0.1308 - val_f1: 0.0000e+00
Epoch 5/10
127/127 [==============================] - 11s 90ms/step - loss: 0.1735 - accuracy: 0.7584 - f1: 0.8475 - val_loss: 0.3124 - val_accuracy: 0.1843 - val_f1: 0.0000e+00
Epoch 6/10
127/127 [==============================] - 12s 95ms/step - loss: 0.1657 - accuracy: 0.7693 - f1: 0.8546 - val_loss: 0.1844 - val_accuracy: 0.7334 - val_f1: 0.0000e+00
Epoch 7/10
127/127 [==============================] - 14s 111ms/step - loss: 0.1580 - accuracy: 0.7802 - f1: 0.8608 - val_loss: 0.0011 - val_accuracy: 1.0000 - val_f1: 0.0000e+00
Epoch 8/10
127/127 [==============================] - 12s 91ms/step - loss: 0.1438 - accuracy: 0.8010 - f1: 0.8745 - val_loss: 0.4340 - val_accuracy: 0.0902 - val_f1: 0.0000e+00
Epoch 9/10
127/127 [==============================] - 11s 90ms/step - loss: 0.1376 - accuracy: 0.8080 - f1: 0.8773 - val_loss: 0.5422 - val_accuracy: 0.0813 - val_f1: 0.0000e+00
Epoch 10/10
127/127 [==============================] - 11s 90ms/step - loss: 0.1230 - accuracy: 0.8347 - f1: 0.8943 - val_loss: 0.4676 - val_accuracy: 0.0912 - val_f1: 0.0000e+00
```

نکات

- دو عدد از عکس ها به دلیل سفید بودن و مچ نبودنشان با لیبلشان از داده ها حذف شده اند.
- به دلیل مشکلات اینترنت و کمبود حافظه در کولب بخشی از زمان صرف اجرا گرفتن های چندین باره میشد که اگر این مشکلات به وجود نمی‌آمد میتوانستیم ایده‌های بیشتری را تست کنیم و مدل کامل‌تری ارائه کنیم.

References

Image projective transformation with OpenCV python | by Kai Nguyen | Medium

OpenCV: Feature Matching + Homography to find Objects

opencv - How to apply RANSAC on SURF, SIFT and ORB matching results - Stack Overflow

OpenCV: Feature Matching

opencv - How to replace a contour (rectangle) in an image with a new image using Python? - Stack Overflow

python - What's the difference between getAffineTransform(), getPerspectiveTransform() and findHomography() - Stack Overflow

python 2.7 - error: (-5) image is empty or has incorrect depth (!=CV_8U) in function cv::SIFT::operator () - Stack Overflow

https://www.youtube.com/watch?v=hfXMw2dQO4E

https://keras.io/examples/vision/siamese_network/