به نام خدا

پروژه‌ی مبانی بینایی ماشین

دکتر شهره کسائی
زمستان 1402

اعضای گروه:
محمدجواد ماهرالنقش - 99105691
امیرمحمد فخیمی - 99170531
پارسا شریفی - 99101762

راهنمای محترم پروژه:
آقای آرمان زارعیان

مقدمه‌ای بر پروژه‌ها

طبق تقسیم‌بندی انجام شده،۴ **پروژه زده شد که** اولی مربوط به Football Tracking شامل 3 زیرپروژه، دومی مربوط به track کردن اجسام کوچک و با سرعت بالا در فیلم‌ها ورزشی همچون توپ تنیس است، سومی مربوط به پردازش تصاویر Pathology و چهارمین مورد مربوط به 3D Reconstruction است.

پروژه اول: پردازش داده های فوتبالی

مقدمه

به طور کلی پردازش های حوزه تصاویر و فیلم های ورزشی، شامل Segmentation و Detection و تحلیل نتایج بازی هاست که مورد سوم ارتباطی با Computer Vision نداشته و برای همین به آن پرداخته نشده است.
دو مورد اول در قالب 3 زیرپروژه جداگانه تعریف شده است که در فایل کد به همه آنها پرداخته شده است. در جوپیترنوتبوک های گذاشته شده، به جزییات پیاده سازی، مقالاتی که پیاده سازی از آنها الهام گرفته و موارد اینچنینی پرداخته شده است.

انتخاب دیتاست

با توجه به اینکه دغدغه اصلی پیدا کردن دیتاست در حوزه فوتبال بود، در Kaggle و به طور کلی Google دیتاست های متفاوتی پیدا شد که یکی از بهترین های آن را با توجه به تعداد Upvoteها و رای های داده شده سعی کردم انتخاب کنم.

کارهای پیشین

تقسیم‌بندی بازیکن

در سال‌های اخیر، استفاده از معماری‌های یادگیری عمیق مانند یونت و نسخه‌های مختلف آن مانند یو2-نت به طور چشمگیری توانایی تقسیم‌بندی بازیکنان را در ردیابی فوتبال پیشبرد. این شبکه‌ها، الهام گرفته از زمینه تصویربرداری پزشکی، عملکرد قابل توجهی را در تفکیک بازیکنان از پس‌زمینه‌های پیچیده و در بین مخفی‌کننده‌ها نشان داده‌اند. به‌طور خاص، یونت با معماری رمزگذار-رمزگشایی و اتصالات پرش با دقتی بالا در تفکیک مرزهای بازیکنان حتی در شرایط چالشی نشان داده است. علاوه بر این، ادغام مکانیزم‌های توجه (ترنسفورمر) و ویژگی‌های چند مقیاس در نسخه‌های یو2-نت عملکرد تقسیم‌بندی را بهبود بخشیده و دقت و استحکام تقسیم‌بندی را ارتقا داده است، که کیفیت تقسیم‌بندی بازیکن در سیستم‌های ردیابی فوتبال را بهبود می‌بخشد.

شناسایی بازیکن

در زمینه شناسایی بازیکن، مدل‌های YOLO (You Only Look Once)، به‌ویژه YOLOv8، به‌عنوان یک انتخاب مشهور به دلیل قابلیت پردازش به زمان واقعی و دقت بالای شناسایی به‌وجود آمده است. YOLOv8 با به‌کارگیری پیشرفت‌های در معماری‌های پشت‌بند، تکنیک‌های ادغام ویژگی و استراتژی‌های آموزش، عملکرد بهتری در شناسایی بازیکن نشان می‌دهد. رویکرد شناسایی یک مرحله‌ای مدل‌های YOLO، به‌همراه الگوریتم‌های نهشته‌سازی کارآمد، آن‌ها را برای برنامه‌های نیازمندیسر و سریع در محلالی که نیازمند شناسایی سریع و دقیق بازیکن است، مناسب می‌سازد. علاوه بر این، قابلیت مدل YOLOv8 برای مدیریت تغییرات مقیاس و صحنه‌های شلوغ، آن را به یک ابزار ارزشمند برای سیستم‌های ردیابی فوتبال در محیط‌های پویا و چالشی تبدیل کرده است.

به طور خلاصه، ادغام مدل‌های یادگیری عمیق مثل یونت، یوتو-نت و YOLOv8 به طور قابل توجهی قابلیت تقسیم‌بندی و شناسایی بازیکنان در ردیابی فوتبال را بهبود داده است. این مدل‌ها از طرح‌های معماری پیشرفته و روش‌های آموزشی پیشرفته برای حل چالش‌های مربوط به وظایف تقسیم‌بندی و شناسایی بازیکنان استفاده می‌کنند، که تحلیل دقیق‌تر و موثرتری از حرکات و تعاملات بازیکنان در زمین را فراهم می‌سازد. با ادامه تحقیقات در این زمینه، پیشرفت‌های بیشتری در معماری‌های مدل، روش‌های آموزش و تکنیک‌های افزایش داده مورد انتظار است که به بهبودهای ادامه‌دار در سیستم‌های ردیابی فوتبال منجر می‌شود.

زیرپروژه ها

- **زیرپروژه‌ی ۱:** تشخیص بازیکنان به کمک Yolov8

  یکی از ابزارهای کاربردی در حوزه پردازش تصویر و بینایی ماشین، ultralytics است که در اینجا از همین استفاده شده
- **زیرپروژه‌ی ۲:** بخش بندی بازیکنان به کمک U-Net و U2-Net

  در یک پروژه/مقاله به بررسی بخش بندی بازیکنان به کمک 2 مدل شامل U-Net و U2-Net پرداخته اند و ما به پیاده سازی مدل آنها با توجه به ساختار این مدل ها و همچنین تست ورودی ها بر روی آن خواهیم پرداخت.
- **زیرپروژه‌ی ۳:** بخش بندی بازیکنان و تماشاگران به کمک Segment Anything

  یک پروژه‌ی نسبتاً بزرگ توسط فیسبوک (یا همان Meta) انجام شده است به نام بخش بندی همه چیز یا همان Segment Anything، در این پروژه به بررسی پیاده سازی آنها و تست عملکرد محصول آنها پرداخته‌ایم.

چالش ها

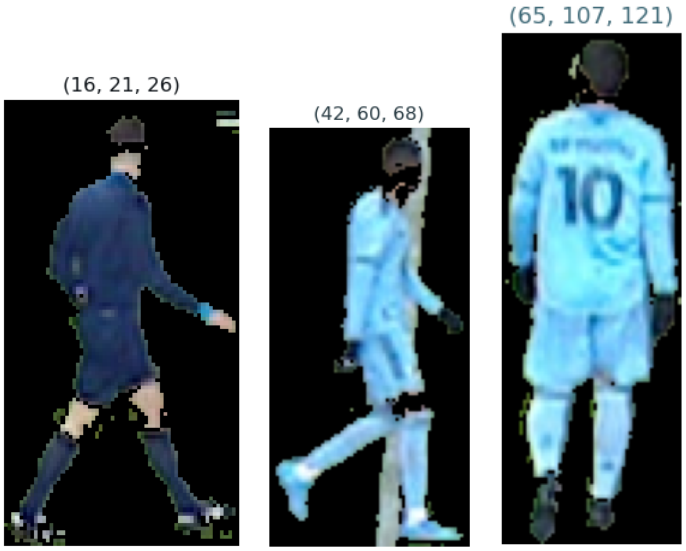
- در زیرپروژه اول از این پروژه، چالش های مختلفی درباره بررسی داده ها داشتیم که به تعدادی از آنها میپردازیم:
- یکی از موارد load کردن دیتاست بود که در Kaggle انجام شد و مشکل آن چنین بود که باید Notebook مربوطه ذیل همان Dataset تعریف میشد که ابتدا Notebookی جداگانه تعریف شده بود و سپس بدین شکل تغییر کرد.
- چالش بعدی، پردازش داده ها بود که به یک ارور عجیبی بدین صورت برخوردیم:

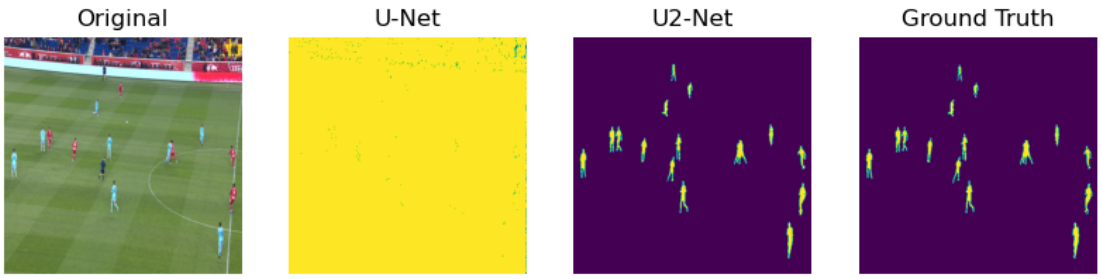
TypeError: can't convert cuda:0 device type tensor to numpy. Use Tensor.cpu() to copy the tensor to host memory first.

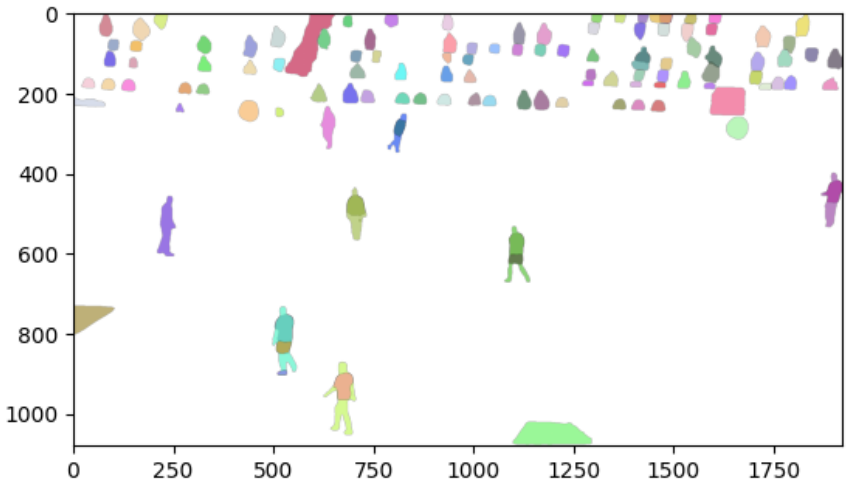
بدین منظور دستور .()cpu().numpy را زدیم و این موضوع حل شد.

تحلیل نتایج

در انتهای نوتبوک، نتایج هر یک از پروژه ها به همراه مقایسه آنها آمده است(مثلا در پروژه U-Net و U2-Net عملکرد U2-Net بسیار بهتر است.






0
200
400
600
800
1000
0
250
500
750
1000
1250
1500
1750

پروژه‌ی دوم: دنبال کردن توپ تنیس

مقدمه

تحلیل فیلم‌های ورزشی انجام شده ولی همچنان دنبال کردن (track) اجسامی که ریز هستند و سرعت بالایی در فیلم دارند، مشکل است. در مقاله‌ی گفته شده به دنبال ایجاد یک مدل برای track کردن توپ‌های ورزشی است. به طور خاص ما به دنبال track کردن توپ در ورزش تنیس هستیم و از PyTorch برای پیاده‌سازی استفاده کرده‌ایم.

دیتاست

برای train کردن مدل از داده‌های بازی‌های ورزشی تنیس استفاده شده است. داده‌های ما شامل یک سری game هستند. هر game شامل یک سری clip یا به اصطلاح rally است. در ادامه در هر clip (از شروع سرویس تا گرفتن امتیاز) frameهای مرتبط با آن rally قرار دارند. در نهایت برای هر rally که شامل تعدادی frame عکس است، یک فایل csv قرار دارد. فایل csv دارای ۵ ستون است. ستون اوّل file name یا نام آن frame در آن clip مورد نظر است. ستون دوم visibily است (این ستون برای افزایش دقّت مدل اضافه شده است.) که مربوط به پیدا بودن توپ می‌باشد که دارای ۴ حالت زیر است:

- عدد صفر: توپ در آن frame نیست.
- عدد یک: توپ به راحتی در آن frame قابل تشخیص است.
- عدد دو: توپ در عکس است ولی به راحتی قابل تشخیص نیست. برای مثال ممکن است blur شده باشد:

- عدد سه: توپ در تصویر است ولی توسط بازیکن، تور یا ... مسدود شده است:

برای توپ‌هایی که visibility آن‌ها غیر از صفر است، ستون‌های بعدی پر می‌شوند.
دو ستون بعدی، X و Y هستند که مختصات توپ در تصویر را نشان می‌دهند. توجّه کنید که در یک frame ممکن است دنباله‌ای از توپ در تصویر باشد که در این صورت انتهای مسیر حرکت توپ را به عنوان مختصات آن در نظر می‌گیریم. ستون آخر، status است. این ستون نشان‌دهنده‌ی وضعیت توپ است. status سه حالت درد:

- عدد صفر: توپ flying است.
- عدد یک: توپ hit شده است.
- عدد دو: توپ bouncing است.

مدل (شامل ایده‌ها و پیاده‌سازی)

ما مدل را با استفاده از PyTorch پیاده کرده‌ایم. در ادامه ایده‌ی اصلی پشت مدل، مدلی شبیه به U-Net است که ابتدا عکس‌ها را به فضای latent می‌برد و سپس upsample می‌کند. همچنین برای بهبود performance ما batch normalization انجام داده‌ایم و وزن‌ها را به شکل زیر initial کرده‌ایم:

- لایه‌ی Conv2d: وزن‌ها از توزیع uniform هستند و bias صفر است.
- لایه‌ی BatchNorm2d: وزن‌ها برابر با یک و bias صفر است.

نکته‌ی اصلی این کار آن است که برای ورودی به مدل، از سه frame پشت سر هم استفاده می‌کنیم تا هم مواردی مثل occlusion را بتوانیم handle کنیم و هم بتوانیم توپ را track و مسیر حرکت آن را پیدا کنیم. به طور شهودی در هر مرحله خروجی مدل یک heat map شامل مکان توپ است.

استفاده از مدل و inference

در نهایت کافی است فایل پایتون infer_on_video.py را با دستور زیر اجرا کنید تا توپ تنیس در هر فیلم دلخواهی را track کند:

```
python infer_on_video.py --model_path ./model_best.pt --video_path ./input.mp4 --video_out_path ./output.mp4
```

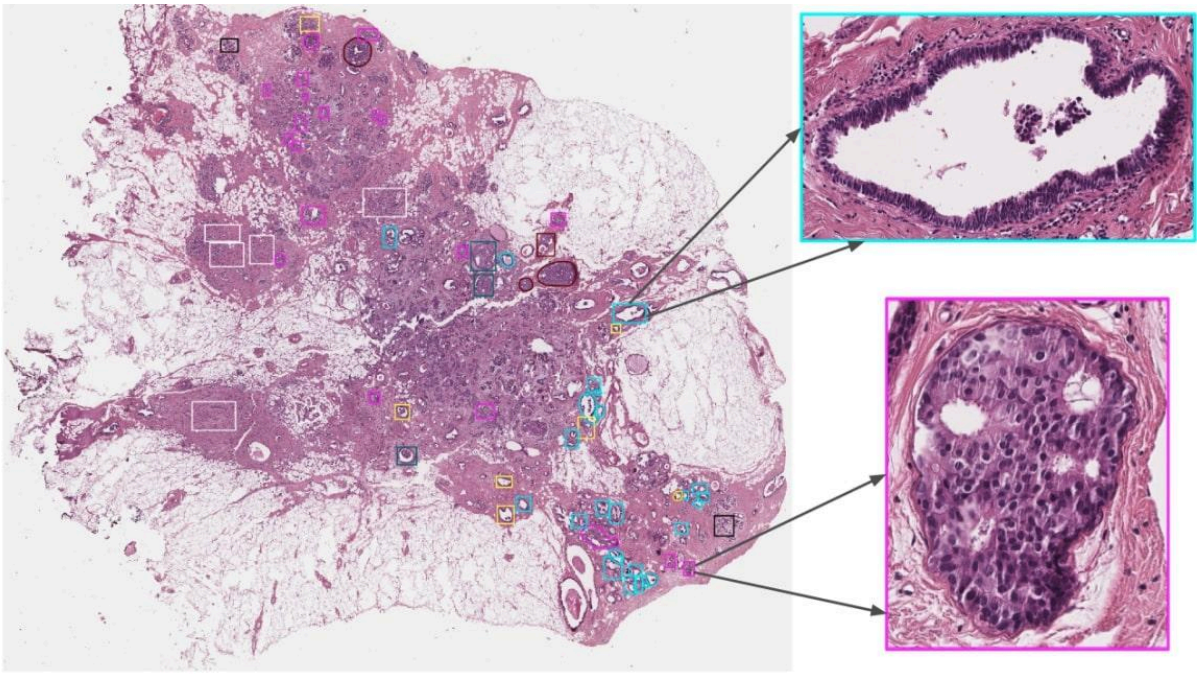
ورودی model_path نشان‌دهنده‌ی path برای مدل train شده است. video_path مربوط به فیلمی است که توپ هنوز به روی آن track نشده و عملاً فیلم ورودی است. video_out_path نیز محلی است که خروجی برنامه (توپ track شده) باید در آن ذخیره شود.

یک نمونه از ورودی و خروجی در این لینک در دسترس است.

پروژه سوم: پردازش تصاویر پزشکی

تعریف صورت مسئله

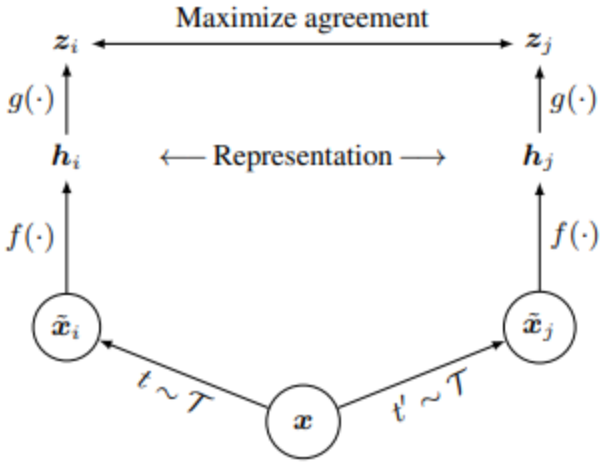
در این پروژه می خواهیم با استفاده از روش های pre-training مبتنی بر contrastive-learning و representation-learning عملکرد مدل خود را بهبود ببخشیم.
کار اصلی ما روی داده های تصویری پزشکی است که میخواهیم با استفاده از این نوع pre-training از یک نسخه کوچک شده از دیتاست BRACS استفاده میکنیم.



در تصویر فوق نمایی از این دیتاست را مشاهده میکنید که یک نمونه به قسمت های متفاوتی تقسیم بندی میشود تا بتوان آن را به عنوان ورودی به مدل داد زیرا که حجم اطلاعات موجود در عکس بسیار بالا میرود.
این دیتا ست به 6 کلاس تقسیم میشود که هدف اصلی ما بالابردن دقت مدل برای شناسایی کلاس تصویر ورودی به مدل است.

تحقیقات پیشین:

این زمینه یکی از مهمترین قسمت هی مربوط به تحلیل تصاویر پزشکی است که پژوهش های زیادی بر روی آن انجام شده.
از جمله مقالات مهم در این زمینه می توان به SimCLR اشاره کرد که یکی از مهمترین مقالات در زمینه contrastive learning میباشد.



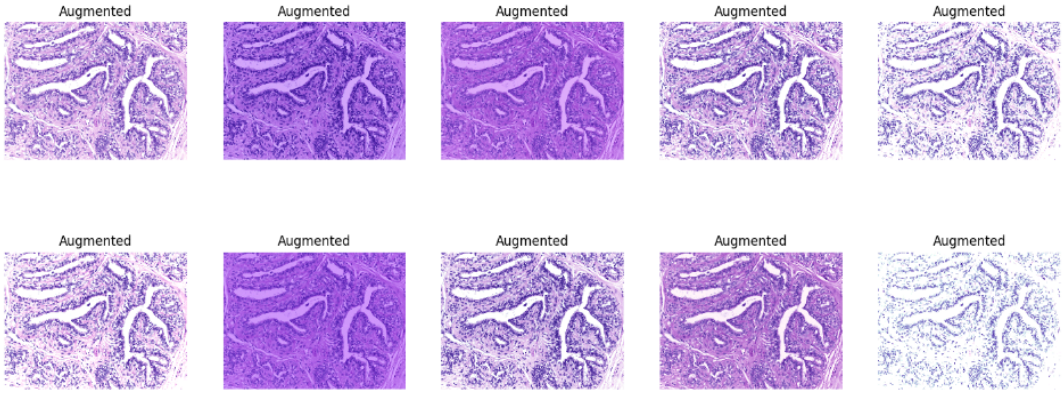
در تصویر بالا نمای کلی از این مدل را میبینید که با اعمال دو **augmentation** روی یک داده ورودی از آن دو داده متفاوا گرفته و سعی در نزدیک کردن **agreement** ها در **representation** های عکس دارد تا بتواند مدل را بتواند مدل را متوجه اشتراکات تصویر کند و قسمت اصلی مدل یا همان **backbone** آن را تعلیم دهد تا بتواند از آن در تسک های دیگر استفاده کرد و این کار را میتوان با استفاده از **transfer learning** انجام داد.

تسک های متفاوتی بر اساس این نوع تعلیم مدل طراحی شده که میتوان به **jigsaw puzzle** اشاره کرد که در آن مدل مجبور به یادگیری شکل و اطلاعات موجود در تصویر می شود تا بتواند تسک خود را انجام دهد.

در این تصویر می توان تسک **jigsaw puzzle** را مشاهده کرد که تصویر اولیه که **a** است به **9** قسمت تقسیم شده و از مدل میخواهیم که تصویر **b** را مرتب کند تا به تصویر اولیه برسیم.
واضح است که در این تسک ها انواع **augmentation** ها نقش مهمی در انجام تسک داشته و با اعمال تغییرات مناسب در عکس میتوان به یادگیری اطلاعات موجود در عکس کمک کرد.
از جمله تغییراتی که می تواند به یادگیری اطلاعات در تصاویر کمک کند اعمال انواع تغییرات در ساختار شکلی و رنگی تصاویر است که هر کدام منجر به یادگیری جنبه خاصی از اطلاعات موجود در تصویر می شود.

ایده پروژه:
در اینجا ما از مدل های **resnet18** به عنوان قسمت های اصلی مدل استفاده میکنیم.
در ابتدا، تصاویر را به صورت سیاه و سفید در آورده و عکس اصلی را گرفته و یک پچ از آن را جدا میکنیم و دو تصویر به دست می آوریم که میخواهیم ویژگی های استخراج شده از این دو تصویر را به هم نزدیک کنیم.

پس یک کلاس dataset تعریف میکنیم که با دریافت هر عکس یک پچ از آن جدا کرده و روی تصاویر حاصل یک سری augmentation رندوم انجام دهد.
مدل ما یک resnet-18 است که در انتهای آن یک هدر برای نمایش representation ها قرار داده ایم و با استفاده از Normalized Temperature-Scaled Cross-Entropy Loss سعی در کم کردن loss داریم تا فیچر ها به هم نزدیک شوند.
در ادامه این backbone که در واقع یک resnet18 است را برداشته و آن را فریز میکنیم که در ادامه فرآیند training تغیری نکند و یک resnet18 دیگر در ادامه قبلی اضافه کردیم و به نوعی یک aouto encoder تشکیل میدهیم و دوباره فرآیند contrastive learning را پیاده میکنیم و با این تفاوت که دیگر تصاویر را سیاه و سفید نکرده و پچ بندی نمیکنیم، در عوض بر روی رنگ متمرکز میشویم و علاوه بر augmentation های رندوم stain normalization را روی داده ها پیاده میکنیم و از متد های vahadane و macenko را بر روی یک تصویر اعمال کرده تا طیف رنگی آن ها را نیز تغییر داده باشیم، سپس میتوانیم پس از این مرحه training اصلی را انجام داده و نتایج نهایی را مقایسه کنیم.



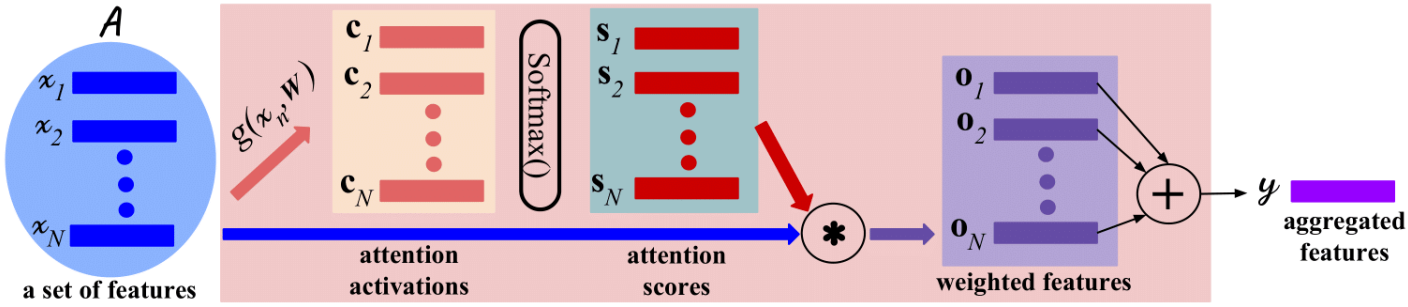
در تصویر بالا تاثیر انواع stain normalization ها را مشاهده میکنید که چگونه بر روی رنگ ها تاثیر میگذارند.

پروژه چهارم: بازسازی تصاویر سه‌بعدی از روی تصاویر ۲ بعدی

این مورد آخرین پروژه است و مرتبط با 3D reconstruction است. این پروژه، مقاله‌ای با نام به همراه نویسندگان زیر است:

Robust Attentional Aggregation of Deep Feature Sets for Multi-view 3D Reconstruction, By Bo Yang, Sen Wang, Andrew Markham, Niki Trigoni. IJCV, 2019

در این مقاله از معماری زیر استفاده شده است:



همچنین الگوریتم آن برای optimize کردن به شکل زیر است:

Algorithm 1 Feature-Attention Separate training of an AttSets enabled network. M is batch size, N is image number.

Stage 1:
for number of training iterations **do**

- Sample M sets of images $\{\mathcal{I}_1, \cdots, \mathcal{I}_m, \cdots, \mathcal{I}_M\}$ and sample N images for each set, *i.e.*, $\mathcal{I}_m = \{\boldsymbol{i}_m^1, \cdots, \boldsymbol{i}_m^n, \cdots, \boldsymbol{i}_m^N\}$. Sample M 3D shape labels $\{\boldsymbol{v}_1, \cdots, \boldsymbol{v}_m, \cdots, \boldsymbol{v}_M\}$.
- Update the base network by ascending its stochastic gradient:

$$\nabla_{\Theta_{base}} \frac{1}{MN} \sum_{m=1}^{M} \sum_{n=1}^{N} \left[\ell(\hat{\boldsymbol{v}}_m^n, \boldsymbol{v}_m)\right], \ \textit{where}\ \hat{\boldsymbol{v}}_m^n \text{ is the estimated 3D shape of single image } \{\boldsymbol{i}_m^n\}.$$

Stage 2:
for number of training iterations **do**

- Sample M sets of images $\{\mathcal{I}_1, \cdots, \mathcal{I}_m, \cdots, \mathcal{I}_M\}$ and sample N images for each set, *i.e.*, $\mathcal{I}_m = \{\boldsymbol{i}_m^1, \cdots, \boldsymbol{i}_m^n, \cdots, \boldsymbol{i}_m^N\}$. Sample M 3D shape labels $\{\boldsymbol{v}_1, \cdots, \boldsymbol{v}_m, \cdots, \boldsymbol{v}_M\}$.
- Update the AttSets module by ascending its stochastic gradient:

$$\nabla_{\Theta_{att}} \frac{1}{M} \sum_{m=1}^{M} \left[\ell(\hat{\boldsymbol{v}}_m, \boldsymbol{v}_m)\right], \textit{where}\ \hat{\boldsymbol{v}}_m \text{ is the estimated 3D shape of the image set } \mathcal{I}_m.$$

The gradient-based updates can use any gradient optimization algorithm.

کد معماری (Network در کد) و کد train آن در فایل main_AttSets.py موجود است. همچنین demoای از اجرای آن (اجرای فایل demo_AttSets.py) به روی تعدادی عکس ورودی به شکل زیر است:

- تعدادی از عکس‌های ورودی:

- خروجی و مقایسه با نسخه‌ی اصلی: