پروژه نهایی شبکه عصبی
شایان بمانیان

بخش صفر)

تفاوت TransferLearning و FineTuning این است که در TransferLearning ما دقیقا همان مدل train شده روی یک دیتاست را برای حل مسئله مشابهی استفاده می‌کنیم، اما FineTune کردن آن زمانی اتفاق می‌افتاد که لایه خروجی شبکه pre-trained را تغییر میدهیم که مناسب انجام تسک موردنظر ما بشود. در واقع FineTuning یکی از روش های TransferLearning می‌باشد.

ObjectDetection یکی از مهم‌ترین ترین مفاهیم در دنیای بینایی کامپیوتر است که پایه ی مسائلی همچون ImageCaptioning و Object Tracking می‌باشد. محبوب‌ترین الگوریتم های Object Detection از CNN ها استفاده می‌کنند و شامل دو مرحله هستند، یکی Region Propsal Stage که مجموعه‌ای از جاهایی که اشیا با احتمالا می‌توانند در آن جا باشد، و مرحله دوم با استفاده از CNN اشیا را تشخیص می‌دهد و اشتباهات مرحله گذشته را برطرف می‌کند.

محبوب‌ترین متود های تشخیص اشیا عبارتند از:

- FASTER R-CNN
- RETINANET
- YOLO (YOU ONLY LOOK ONCE)
- MASK R-CNN
- SSD (SINGLE SHOT MULTIBOX DETECTOR)

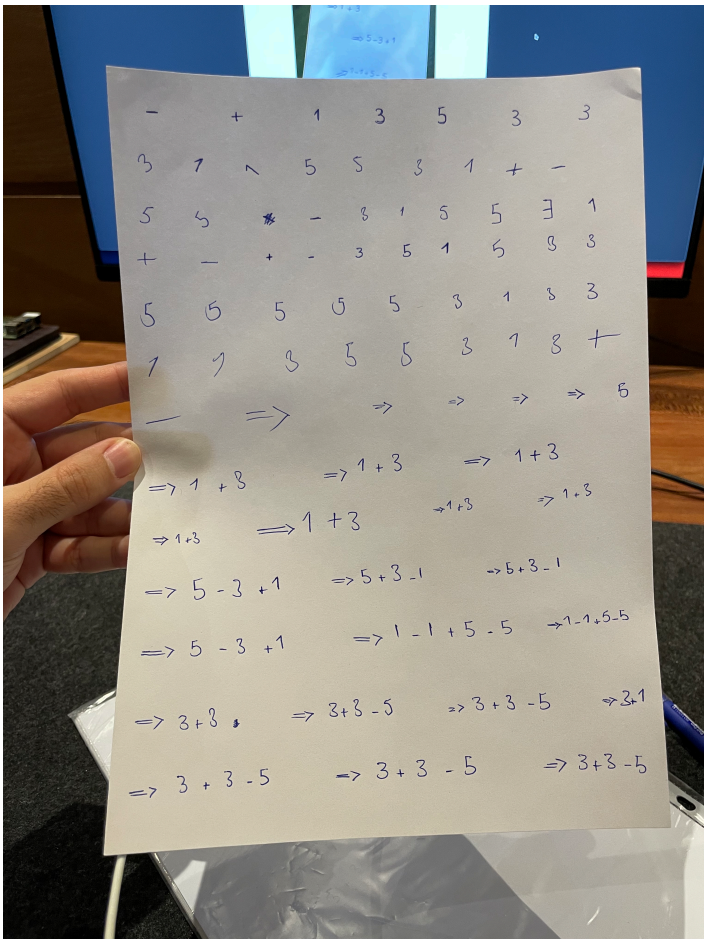
منابع:

https://viso.ai/deep-learning/object-detection

https://medium.com/@pedroazevedo6/object-detection-state-of-the-art-2022-ad750e0f6003

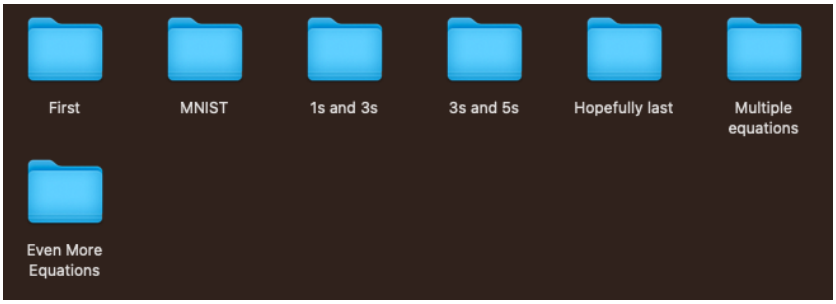
بخش یک)

تعداد زیادی از اعداد روی کاغذ نوشته شد و سپس هر یک کراپ شد و به روبوفلو منتقل شد.

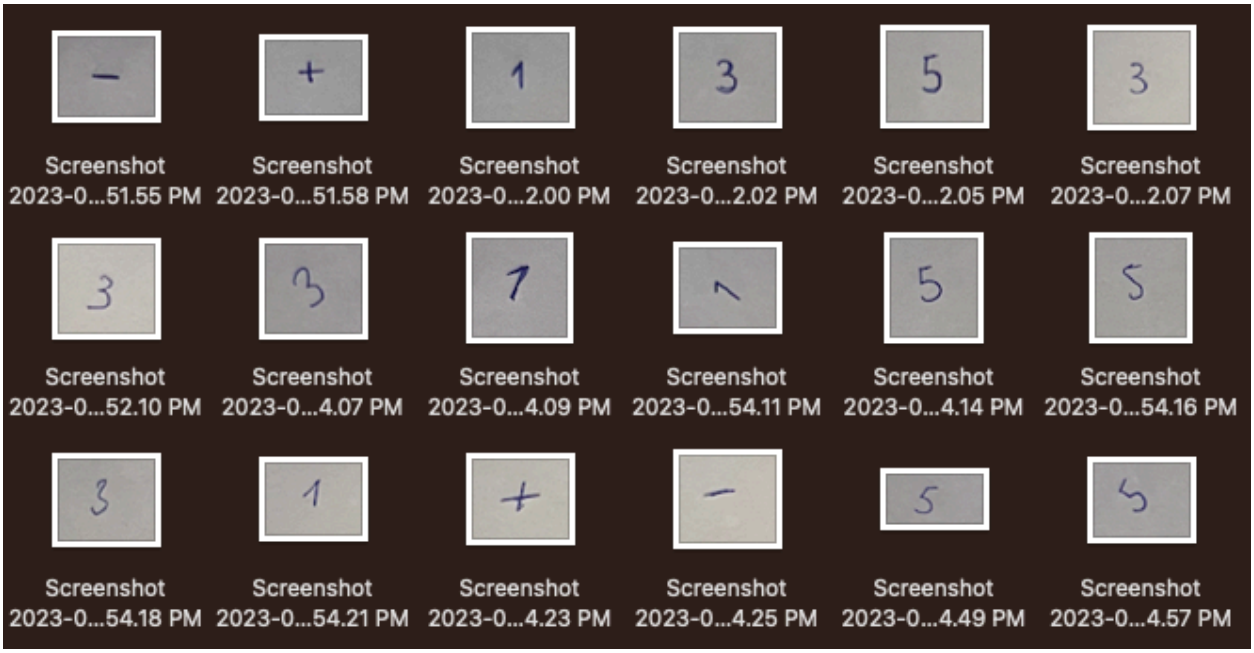


نمونه برگه استفاده شده

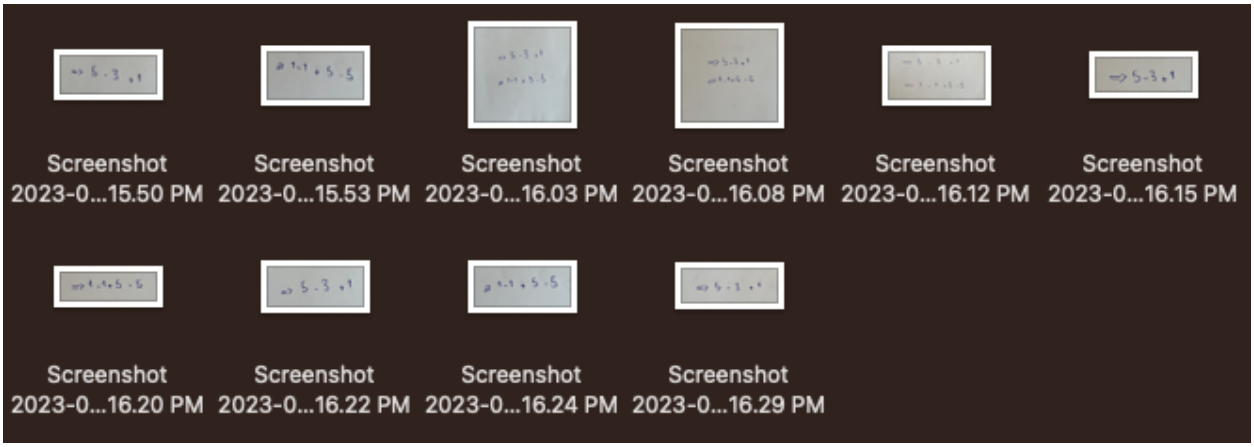
برای کمک به دیتاست نمونه‌ای کوچک از دست‌خط های دیتاست MNIST هم استفاده شد و در روبوفلو لیبل زده شد.
در کل تعداد بچ ها به این صورت می‌باشد:



داخل هر بچ چندین تصویر کراپ شده به صورت زیر وجود دارد:



نمونه بچ شامل اعداد



نمونه بچ شامل معادله

بخش دو)

برای تقویت دیتاست از روشی استفاده می‌شود که به آن دیتا آگمنتیشن گفته می‌شود.
در این روش تعداد تصاویر دیتاست ما با تغییرات کوچکی مثل افزایش روشنایی تصویر، اضافه کردن نویز، چرخاندن یا سیاه سفید کردن تصویر چند برابر می‌شود.
به‌طور مثال ممکن است دیتا هدف ما در محیطی با نور بسیار زیاد عکس‌برداری شده باشد ولی دیتاست آموزشی ما در محیطی به نور کم تصویر برداری شده باشد. در این شرایط، شاید پیش‌بینی مدل طبق انتظار نباشد. آگمنتیشن به ما کمک می‌کند این مشکلات را حل کنیم. درواقع به ما کمک می‌کند تا داده های بیشتری ببیند که این موضوع به طور مستقیم دقت آن را افزایش می‌دهد.

در این تمرین از سایت roboflow برای لیبل زدن و آگمنتیشن استفاده شده است. متاسفانه به علت تحریم ها، سرویس فایربیس گوگل در دسترس کاربران ایرانی نیست و سایت روبوفلو که از این سرویس استفاده می‌کند به سختی در دسترس است و زمان بسیار زیادی صرف کار کردن با این وب‌سایت شده است.

دیتاست استفاده شده در این پروژه در آدرس زیر قابل مشاهده می‌باشد:
https://universe.roboflow.com/shayan-bemanian/equation-solver-slfis

با آزمون و خطا متوجه شدیم که بهتر است آگمنتیشن را ساده نگه داریم. و تنها از grayscale و brightness استفاده شده. دلیل استفاده از این دو در بالا آورده شده ولی دلیل استفاده نکردن از بقیه گزینه ها، مثل اضافه کردن نویز یا چرخاندن، این بود که نه تنها تاثیر مثبتی در خروجی نداشت، آن را بدتر هم می‌کرد. مثل اگر 3 رو بچرخانیم شبیه 5 می‌شود و این موضوع تشخیص را برای مدل سخت می‌کند.
پس با افزایش یا کاهش روشنایی و اضافه کردن نسخه سیاه و سفید، دیتاست را آماده شرایط تست متفاوت کردیم.

بخش سه)

در این بخش توضیحی درباره چگونگی حل این پروژه خواهم داد. همچنین برای شفافیت بیشتر کد پروژه کامنت گذاری شده است.
پس از اینستال کردن detectron2 و ایمپورت کردن ملزومات، فایل دیتاست را به صورت زیپ را دریافت و آن را داخل یک فولدر روی دیسک کولب ذخیره کردیم.

دیتاست را که به صورت COCO است را در detectron2 رجیستر می‌کنیم.
در صورتی که بخواهیم نوت‌بوک را بدون ریست کردن ران‌تایم دوباره ران کنیم، خطایی مبنی بر اینکه دیتاست قبلا تعریف شده است دریافت می‌کنیم که کد رفع این مشکل آورده شده است.

نگاهی به دیتای train و لیبل های آن می‌اندازیم.

با استفاده از یک مدل از پیش‌تعریف شده که در Model Zoo وجود دارد می‌توانیم آموزش را شروع کنیم.
در این بخش مدل های زیادی تست شده اند، یکی از کاندید ها با عمق ۱۰۱ بود.
در نهایت مدل انتخاب شده Faster RCNN با عمق ۵۰ است. مدل های دیگری نیز موجود هستند که لیست کامل آن اینجا آورده شده. هرچند که می‌گویند The deeper, the better اما کلا ۶ کلاس نه چندان پیچیده داریم. پس به همان مدل سبک‌تر با عمق ۵۰ رضایت می‌دهیم.
پارامتر های BASE_LR و MAX_ITER با آزمون و خطای زیاد (و قطع چندین شدن باره ران‌تایم به علت استفاده بیشتر از حد رایگان) به دست آمده‌اند.
معیار سنجش ما بجز تست کردن روی داده های تست، استفاده از TesnorBoard بود.
پارامتر NUM_CLASSES هم شامل تعداد کلاس ها می‌باشد. بعضی منابع می‌گویند تعداد منابع به علاوه یک، برخی می‌گویند همان تعداد کلاس هاست. ما در اینجا تعداد کلاس ها به علاوه یک را آورده‌ایم.

مرحله بعدی لود کردن مدل بود و باز هم با آزمون و خطا با SCORE_THRESH_TEST روی ۰.۷ نتیجه مطلوبی حاصل شد.

کدی قرار داده شده تا اگر مدلی روی Google Drive سیو شده، دوباره از آن استفاده شود. این مرحله نیز لازم است، چون هربار ترین شدن مدل زمان زیادی می‌برد و با توجه به محدودیتی زمانی کولب ممکن است همه کار های انجام شده از بین برود. در خروجی همه کد ها تمیز شده اند و مدل از اول ران شده، به این دلیل این سل کامنت شده و نیازی به استفاده از آن نبود.

قدم بعدی چک کردن کارایی مدل روی دیتاست تست ماست. که از نتیجه رضایت حاصل شد.

حال تصویر داده شده در pdf را برای تست به مدل می‌دهیم. برای اینکار این فایل را در یک سرویس آپلود عکس آپلود کردیم و با wget آن را در کولب دانلود کرده و نمایش دادیم.
برای اجرای مدل روی آن را تابع predictor استفاده کردیم و خروجی را نمایش دادیم.
می‌توان دید که مدل برخی از اعداد را تشخیص نداده است، اما این مسئله در این قدم برای ما اهمیت زیادی ندارد و از کلیت تشخیص راضی هستیم.
رویکرد ما به مسئله به نوعی divide and conquer است.
حالا باید هر ۴ معادله را از تصویر استخراج کنیم، این کار را با استفاده از تشخیص فلش <= انجام میدهیم.
چون هر معادل با یک فلش شروع می‌شود، هر فلش را نماینده یک معادله در نظر می‌گیریم.
موقعیت هر فلش را داریم. پس می‌توانیم چند پیکسل اطراف آن را هم ببینیم. اینکار را انجام دادیم و هر معادله را به صورت جداگانه کراپ کردیم.
در قدم بعد، تابع predictior روی هر معادله اجرا شده و توضیحات آن (که کمی با جزئیات است) به طور دقیق در کامنت های داخل کد توضیح داده شده است و نوشتن توضیحات آن ها در این فایل واضح نخواهد بود.
در آخرین مرحله هم با استفاده از opencv باندینگ باکس های دور معادلات رسم شده و حاصل معادله در گوشه آن نوشته شده است.

در این پروژه موفق شدیم که هر چهار معادله را به طور کامل تشخیص داده و حل کنیم.

آدرس نوت‌بوک کولب:

https://colab.research.google.com/drive/1e3ZmcdKNqOhjiUeExB20IN748dZYCMU?usp=sharing