تمرین سری هشتم بینایی کامپیوتر

دکتر محمدی

هلیا شمس زاده

۴۰۰۵۲۱۴۸۶

سوال اول).....

بخش الف-

روش پنجره لغزان(Sliding Window)

در این روش، یک پنجره با اندازه ثابت در سراسر تصویر حرکت می‌کند و هر بار که پنجره روی یک قسمت از تصویر قرار می‌گیرد، آن بخش از تصویر به عنوان ورودی به CNN داده می‌شود. این فرآیند به طور مکرر برای کل تصویر انجام می‌شود تا تمامی بخش‌های تصویر بررسی شوند.

پنجره با یک گام ثابت در عرض و طول تصویر حرکت می‌کند. هر بار که پنجره بر روی یک بخش از تصویر قرار می‌گیرد، آن بخش به عنوان ورودی به شبکه عصبی داده می‌شود. شبکه عصبی هر بخش را جداگانه پردازش می‌کند و نتایج را ذخیره می‌کند.

پیاده‌سازی با کانوولوشن

در این روش، کل تصویر به عنوان ورودی به شبکه عصبی داده می‌شود. شبکه عصبی به صورت همزمان تمام ویژگی‌ها و نواحی مختلف تصویر را پردازش می‌کند و خروجی نهایی را تولید می‌کند.

تصویر به شبکه عصبی داده می‌شود. شبکه عصبی تمامی ویژگی‌ها و نواحی تصویر را به طور همزمان پردازش می‌کند. خروجی نهایی بر اساس کل تصویر تولید می‌شود.

مقایسه تفاوت‌ها

۱. **زمان پردازش:**
   - **پنجره لغزان:** زمان‌بر است، زیرا باید پنجره به صورت پیاپی در سراسر تصویر حرکت کند.
   - **با کانوولوشن**: سریع‌تر است، زیرا کل تصویر به صورت همزمان پردازش می‌شود.
۲. **دقت در شناسایی نواحی کوچک:**
   - **پنجره لغزان:** می‌تواند نواحی کوچک و جزئیات را با دقت بیشتری شناسایی کند.
   - **با کانوولوشن**: ممکن است دقت کمتری در شناسایی نواحی کوچک داشته باشد.
۳. **پیچیدگی و تنظیمات:**
   - **پنجره لغزان:** نیاز به تنظیم پارامترهای مربوط به اندازه و گام پنجره دارد.

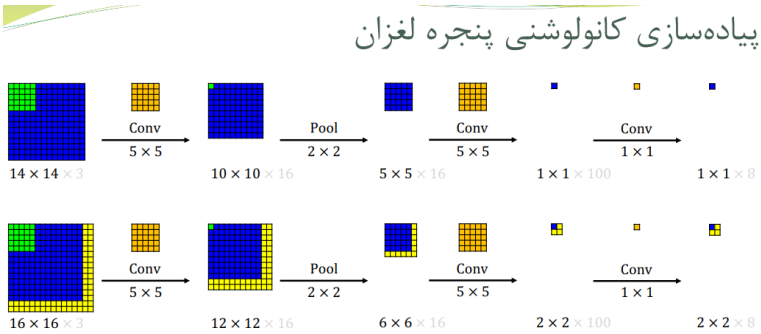
- **با کانوولوشن**: ساده‌تر است و نیاز به تنظیمات خاصی ندارد.

٤. **استفاده از منابع سخت‌افزاری:**

- **پنجره لغزان:** مصرف منابع بیشتری دارد به دلیل پردازش مکرر.
- **با کانوولوشن:** مصرف منابع بهینه‌تر است و محاسبات کمتری دارد چون کل تصویر یکبار پردازش می‌شود.

٥. **وابستگی به اندازه تصویر:**

- **پنجره لغزان:** با تصاویر با اندازه‌های مختلف به خوبی کار می‌کند.
- **استفاده مستقیم:** ممکن است نیاز به تنظیمات خاص برای تصاویر با اندازه‌های مختلف داشته باشد.



مثلا در مثال بالا، اگر یک پنجرهٔ ۱۴ در ۱۴ را با for ۴ بار روی تصویر ۱۶ در ۱۶ بلغزانیم، باید به تعداد ۴ بار این لغزش را انجام داده و نتیجه را حساب کنیم که یعنی

$2 \times 2 \times (10 \times 10 \times 5 \times 5 \times 3 \times 16)$ بار محاسبات که برابر است با ۴۸۰,۰۰۰ بار ضرب.

اما اگر با کانوولوشن پیاده‌سازی شود، برابر می‌شود با $12 \times 12 \times (5 \times 5 \times 3 \times 16)$ که یعنی ۱۷۲,۸۰۰ بار ضرب. یعنی میزان محاسبات تقریبا ۳ برابر کمتر شد.

بخش ب-

Yolo یک مدل شبکه عصبی کانوولوشنی برای تشخیص اشیا است که قادر است چندین کلاس اشیا را در یک سلول شبکه واحد مدیریت کند.
این مدل تصویر به یک شبکه S×S تقسیم می‌کند.
هر سلول شبکه چندین جعبه مرزی (Bounding Box) و احتمال‌های کلاس مربوطه را پیش‌بینی می‌کند. (حداکثر به تعداد Anchor box ها)
Yolo برای هر سلول شبکه (grid) اطلاعات زیر را پیش‌بینی می‌کند:

- مختصات جعبه‌های مرزی: هر جعبه مرزی با چهار مقدار (bx, by, bw, bh) مشخص می‌شود:

Bx, by: مختصات مرکز جعبه در سلول

Bw, bh: عرض و ارتفاع جعبه نسبت به اندازه کل تصویر

- احتمال اشغال بودن جعبه توسط شیء: هر جعبه مرزی یک مقدار confidence score دارد که احتمال وجود شیء در آن جعبه را نشان می‌دهد.
- احتمال کلاس‌ها: برای هر جعبه مرزی، Yolo احتمال تعلق آن جعبه به هر یک از کلاس‌های موجود را پیش‌بینی می‌کند. (یک عدد بین ۰ و ۱ برای هر کلاس)

همچنین استفاده از جعبه‌های Anchor به Yolo کمک می‌کنند تا اشیاء با اندازه‌ها و اشکال مختلف را تشخیص دهد. جعبه‌های Anchor مجموعه‌ای از جعبه‌های مرزی پیش‌تعریف شده با اندازه‌ها و نسبت‌های مختلف هستند. مدل، مختصات آفست‌های جعبه را نسبت به این جعبه‌های Anchor پیش‌بینی می‌کند.
استفاده از جعبه‌های Anchor باعث می‌شود که مدل بتواند اشیاء با ابعاد و نسبت‌های مختلف را بهتر تشخیص دهد، زیرا جعبه‌های پیش‌بینی شده می‌توانند به شکلی نزدیک‌تر به ابعاد واقعی اشیاء در بیایند و بصورت نسبی بیان شوند.
به طور کلی، Yolo با استفاده از شبکه‌های سلولی، پیش‌بینی‌های متعدد برای هر سلول و استفاده از جعبه‌های Anchor به تشخیص اشیا با دقت و کارایی بالا می‌پردازد.

سوال دوم).....

بخش الف-

۱. معماری

YOLO (You Only Look Once)

- **معماری شبکه:** YOLO از یک شبکه عصبی پیچشی (CNN) با لایه‌های مختلف برای استخراج ویژگی‌ها و تشخیص اشیاء استفاده می‌کند. YOLO تصویر ورودی را به یک شبکه S×S تقسیم می‌کند و برای هر سلول شبکه چندین جعبه مرزی و احتمال کلاس‌ها را پیش‌بینی می‌کند. مبتنی بر پنجرهٔ لغزان است.
- **خروجی:** برای هر سلول، YOLO تعدادی مختصات جعبه‌های مرزی، confidence score و احتمال کلاس‌ها را پیش‌بینی می‌کند.

SSD (Single Shot MultiBox Detector)

- **معماری شبکه:** SSD از یک شبکه عصبی پیچشی همراه با چندین لایه تشخیص استفاده می‌کند. هر یک از این لایه‌ها مسئول تشخیص اشیاء با اندازه‌های مختلف است. مبتنی بر پنجرهٔ لغزان است.
- **خروجی:** SSD از مجموعه‌ای از جعبه‌های پیش‌فرض (default boxes) با نسبت‌های ابعاد و اندازه‌های مختلف استفاده می‌کند و پیش‌بینی‌های مختصات و احتمال کلاس‌ها را برای هر جعبه پیش‌فرض انجام می‌دهد.

۲. سرعت

- **YOLO:** به دلیل اینکه YOLO تمام پیش‌بینی‌ها را در یک مرحله انجام می‌دهد، سرعت اجرای بالایی دارد و مناسب برای کاربردهای زمان واقعی (real-time) است.
- **SSD:** SSD نیز به دلیل استفاده از شبکه‌های عصبی عمیق با چندین لایه تشخیص، سرعت بالایی دارد و می‌تواند در کاربردهای real-time مورد استفاده قرار گیرد. با این حال، سرعت آن ممکن است کمی کمتر از YOLO باشد.

۳. دقت

- **YOLO**: YOLO به دلیل رویکرد ساده‌اش و تعداد کمتر پیش‌بینی‌ها ممکن است دقت کمتری نسبت به SSD داشته باشد، به خصوص برای اشیاء کوچک یا متراکم. مثلا تعداد کل اشیا پیدا شده با این الگوریتم در یک تصویر حداکثر ۹۸ تا است.
- **SSD**: SSD به دلیل استفاده از چندین لایه تشخیص با اندازه‌های مختلف، دقت بالاتری در تشخیص اشیاء با اندازه‌های متفاوت و کوچک دارد. با این روش می‌توان حداکثر ۸۷۳۲ تا شی از تصویر پیدا کرد که بسیار از yolo بیشتر است، البته تعداد زیادی از آن هم می‌تواند background باشد.

تحلیل سناریوها

سناریوهایی که YOLO بهتر عمل می‌کند:

- **کاربردهای زمان واقعی:** به دلیل سرعت بالای YOLO، در سناریوهای real-time مانند نظارت ویدئویی و ماشین‌های خودران بهتر عمل می‌کند.
- **اشیاء بزرگ و متوسط:** YOLO در تشخیص اشیاء با اندازه‌های بزرگ و متوسط عملکرد بهتری دارد.

سناریوهایی که SSD بهتر عمل می‌کند:

- **تشخیص اشیاء کوچک:** به دلیل استفاده از لایه‌های چندگانه با اندازه‌های مختلف، SSD در تشخیص اشیاء کوچک دقت بیشتری دارد.
- **تصاویر با اشیاء متراکم:** SSD در تشخیص اشیاء در تصاویر با تراکم بالا و اشیاء نزدیک به هم عملکرد بهتری دارد. مثلا در مکانی شلوغ که می‌خواهیم تمام افراد را تشخیص بدیم و تراکم افراد زیاد است.

بخش ب-

در برخی روش‌های کلاسیک تشخیص یک مرحله‌ای مانند SSD که تقریباً 10^4 تا 10^5 پروپوزال را در هر تصویر ارزیابی می‌کنند، اما تنها تعداد کمی از مکان‌ها حاوی اشیا (به عنوان مثال اکثرا پیش‌زمینه) هستند و بقیه فقط اشیاء پس‌زمینه هستند. این منجر به مشکل عدم تعادل طبقاتی می شود. یعنی مثلا در ۸۷۳۲ تا پروپوزال پیدا شده شاید حداکثر ۲۰ تای آن مربوط به یک شئ باشند و بقیه مربوط به پس‌زمینه باشند.

RetinaNet با هدف رفع این مشکل، مفهوم جدیدی به اسم focal loss ارائه کرد که نوعی تابع هزینه cross entropy است.

مشکل عدم تعادل کلاس به دو روش زیر بر مدل تشخیص شی تأثیر می گذارد:

الف) عدم یادگیری به دلیل نکات منفی آسان

اگر یک شبکه عصبی بسازیم و کمی آن را آموزش دهیم، به سرعت یاد می‌گیرد که منفی‌ها (آنهایی که شئ نیستند) را در سطح پایه طبقه‌بندی کند. از این نقطه به بعد، بیشتر نمونه‌های آموزشی کار چندانی برای بهبود عملکرد مدل انجام نمی‌دهند، زیرا مدل در حال حاضر کار قابل قبولی را انجام می‌دهد. این باعث می‌شود آموزش ناکارآمد باشد زیرا اکثر مکان‌ها در تصویر easy negative هستند (یعنی می‌توان آن‌ها را به راحتی توسط آشکارساز به عنوان پس‌زمینه طبقه‌بندی کرد) و از این رو هیچ یادگیری مفیدی ایجاد نمی‌کند. یعنی مدل مثلا ۵۰۰۰ تا پروپوزال پیدا کرده است که ۴۹۰۰ تای آن پس‌زمینه و ۱۰۰ تای دیگر اشیا هستند. این مدل شاید در تشخیص پس‌زمینه بودن آن ۴۹۰۰ پروپوزال دقیق باشد و همه را درست تشخیص دهد، ولی در آن ۱۰۰ تا که شی هستند فقط ۴۰ تای آن را درست تشخیص دهد. در این حالت مثلا از ۵۰۰۰ تا مدل ما ۴۹۵۰ تا درست پیش‌بینی کرده است ولی ۴۹۰۰ تای آن کار آسانی است و دقت مدل را با آن ۱۰۰ شی باید سنجید که در اینجا می‌بینیم تشخیص درست ۵۰ تا ۱۰۰ تا اصلا دقت مناسبی نیست، ولی بدون مفهوم focal loss این مدل، مدل خوبی تلقی می‌شود.

ب) اثر تجمعی بسیاری از easy negatives

دیده‌ایم که تصاویر دارای نگاتیو (کلاس‌های پس‌زمینه) بالایی هستند. وقتی تلفات کوچک از چنین نگاتیوهای آسانی در بسیاری از تصاویر جمع می‌شود، ضرر کلی را تحت تأثیر قرار می‌دهد و باعث ایجاد مدل‌های منحط می‌شود. منفی‌هایی که به راحتی طبقه بندی می شوند اکثر ضرر را تشکیل می دهند و بر گرادیان غالب هستند. این امر تمرکز مدل را بر طبقه بندی صحیح فقط نمونه های منفی تغییر می‌دهد. به عبارت دیگر وزن‌ها به گونه‌ای به روز می‌شوند که عملکرد فقط روی نگاتیوها بهتر و بهتر می شود.

برای مثال، یک پیش‌بینی با اطمینان بالا از کلاس پس‌زمینه

$(Y=0) = -\log(1-p) = -\log(1-0.05) = -\log(0.95) = 0.05$ به ضرر کمک می‌کند. و یک پیش‌بینی با اطمینان بالا از کلاس پیش‌زمینه $(Y=1)$ نیز $= -\log(p) = -\log(0.95) = 0.05$ به تابع ضرر کمک می‌کند.

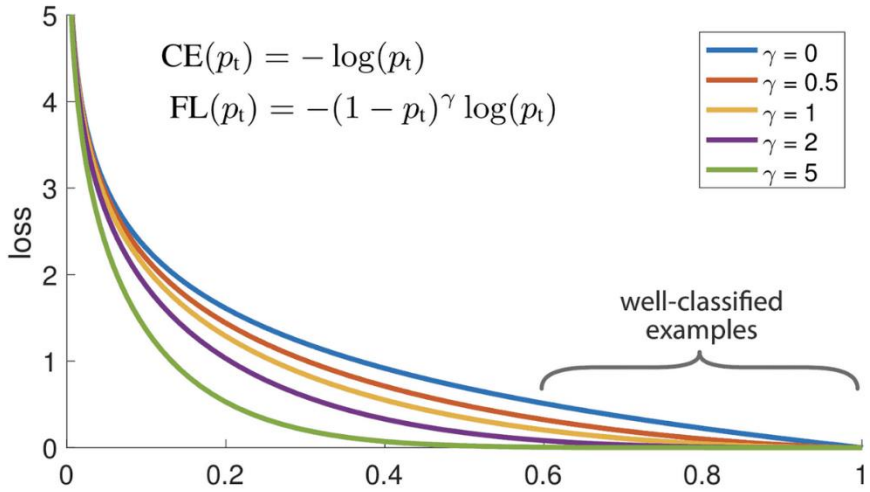
هر دو به طور مساوی به ضرر کمک می کنند، اما از آنجایی که تعداد کلاس های پس زمینه بیشتر است، ضرر ناشی از آن بر ضرر کلی غالب خواهد بود و مدل به سمت بهبود آنها پیش می‌رود.

یک راه برای غلبه بر مشکل نامتعادل بودن نمونه‌ها این است که به کلاس‌های easy negative که تعداد بیشتری هستند وزن کمتری بدهیم تا کمتر loss را تحت تاثیر قرار دهند. این ایدهٔ اصلی در focal loss است.

Focal loss یک عامل تعدیل کننده را معرفی می‌کند که سهم هر نمونه را در loss کل تنظیم می‌کند. این به یادگیری از طریق نمونه های چالش برانگیز وزن بیشتری می‌دهد و در نتیجه عملکرد بهتری دارد.

$$L_{focal}(p) = -\alpha(1-p)^{\gamma}\log p$$

که p با احتمال پیش‌بینی‌شده با ground truth مطابقت دارد، a نشان‌دهنده عامل تعدیل‌کننده برای مقابله با عدم تعادل است، و گاما به عنوان یک پارامتر اضافی برای کنترل سرعت پردازش نمونه‌های آسان عمل می‌کند. در زیر، می‌توانیم یک تصویر بصری از افت کانونی را در مقایسه با آنتروپی متقاطع سنتی در زمانی که alpha = 1 مشاهده کنیم:



این نشان می‌دهد هر چه گاما بزرگتر انتخاب شود و هر چه پیش‌بینی‌ها به ground truth نزدیک‌تر می‌شوند، اثر آن نمونه روی loss کلی کمتر می‌شود، و اینگونه loss های نمونه‌های کلاس easy negative اثر تجمعی کمتری گرفته و مدل را به سمت بهبود نمونه‌های hard positive می‌برد.

مزایا:

در مقایسه با از cross entropy، مزایای بسیاری را ارائه می دهد.

الف- اول، همانطور که قبلا ذکر شد، مدل ML را قادر می سازد تا ویژگی های معناداری را از مجموعه داده های نامتعادل یاد بگیرد، حتی زمانی که اندازه کلاس اقلیت بسیار کوچک است. همچنین، می توان از آن برای

زمانی استفاده کرد که مقادیر ground truth ممکن است حاوی اشتباهاتی به دلیل حاشیه نویسی انسانی باشد.
ب- علاوه بر این، مفهوم focal loss بدون اضافه کردن بار محاسباتی اضافی در روش آموزشی قابل درک و پیاده سازی است. در نهایت، شامل دو فراپارامتر (alpha و gamma) است که انعطاف‌پذیری زیادی را در حوزه‌های مختلف فراهم می‌کند.

این ویژگی‌ها باعث شده که خصوصا در الگوریتم‌های تشخیص اشیا که احتمال نامتعادل بودن نمونه‌ها و کلاس‌ها زیاد است مورد استفاده قرار بگیرد.

سوال سوم).....

کاربرد: در مدل‌های تشخیص اشیاء، معمولاً تعدادی Bounding Box برای یک شیء خاص با امتیازات اطمینان (pc) متفاوت پیش‌بینی می‌شود. این جعبه‌ها ممکن است به شدت همپوشانی داشته باشند، که این همپوشانی براساس معیاری به نام IoU که میزان اشتراک مساحت دو باکس به اجتماع‌شان است، سنجیده می‌شود. تعداد زیاد این همپوشانی‌ها به صورت همزمان می‌تواند نشان‌دهندهٔ یک شیء واحد باشند. بدون استفاده از تکنیک NMS ممکن است یک شئ چند بار شمرده شود.

مراحل: مراحل آن به این صورت است که ابتدا تمام جعبه‌های پیدا شده براساس pc شان سورت می‌شوند. باکس‌هایی که pc آن‌ها از یک threshold ای کوچکتر باشند، یعنی احتمال وجود شئ در آنها بسیار پایین است، پس حذف می‌شوند. حال باکس‌های باقیمانده در هر کلاس را دو به دو با هم مقایسه کرده و آن دویی که IoU شان از یک threshold ای (مثلا ۰.۵ – که این مقدار از مسئله به مسئلهٔ دیگر فرق می‌کند) بیشتر باشد احتمالا مربوط به یک جسم هستند، پس وارد مرحلهٔ NMS می‌شوند و باکس با pc بیشتر نگه داشته شده و دیگری حذف می‌شود.

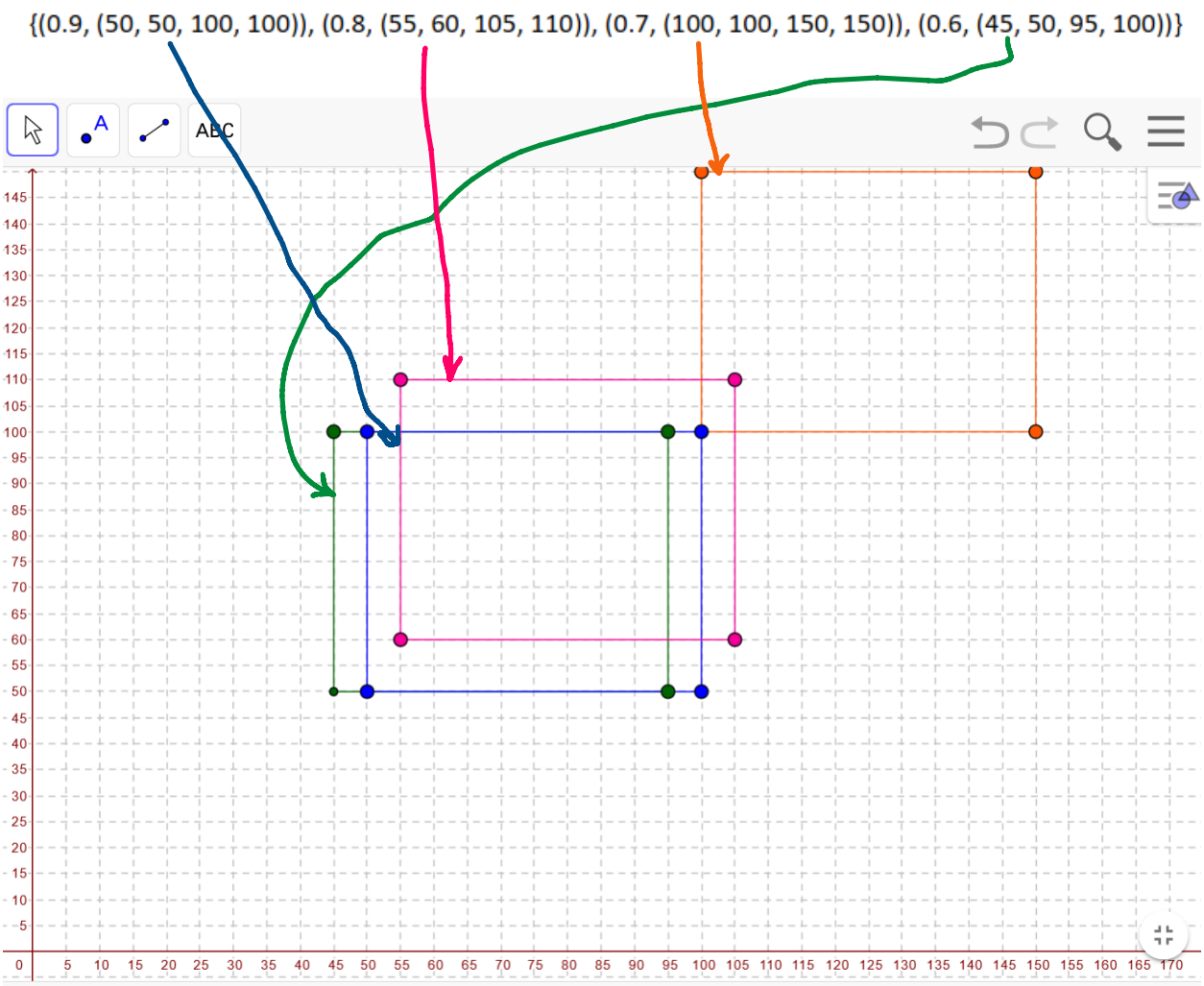
سطح آستانه: مقدار سطح آستانهٔ NMS بسیار اهمیت دارد و بر نتیجه تاثیر می‌گذارد. در برخی مسائل سطح آستانهٔ بالا و در برخی مسایل پایین انتخاب می‌شود. هر چه این threshold کوچکتر باشد، باکس‌ها با احتمال بیشتری وارد NMS می‌شوند زیرا نسبت به همپوشانی آنها سختگیری بیشتری دارد. مثلا در مسائلی که می‌دانیم معمولا فاصلهٔ بین اشیا زیاد است و اشیا به حد کافی بزرگ یا متوسط هستند از یک حد آستانهٔ کوچک استفاده می‌کنیم. ولی احتمال اشتباه زیادی دارد. این می‌تواند باعث حذف برخی از جعبه‌های صحیح و کاهش دقت شود، اما در عین حال باعث کاهش تعداد جعبه‌های مرزی نهایی می‌شود و نتیجه نهایی را تمیزتر می‌کند.

یعنی ممکن است به FN ها اضافه کند و از TP ها کم کند چون از تعدادی که درست تشخیص داده کم شده و در اصل به اشتباه می‌گوید برخی جاها شئ نداریم.

هر چه threshold بیشتر باشد، احتمال اینکه دو باکس که همپوشانی دارند وارد NMS شوند کمتر است. این مقدار برای مسائلی مناسب است که تعداد زیادی آبجکت در یک کلاس با فاصلهٔ کمی قرار دارند، مثلا وقتی می‌خواهیم همهٔ آدم‌ها را در یک جای شلوغی پیدا کنیم. البته سطح آستانهٔ بالا تعداد کمتری از باکس‌ها را وارد مرحلهٔ NMS می‌کند. این ممکن است باعث شود که برخی جعبه‌های مرزی نزدیک به هم که واقعاً نشان‌دهنده یک شیء واحد هستند، حذف نشوند. در نتیجه، ممکن است تعداد بیشتری جعبهٔ مرزی در خروجی باقی بماند و برای یک شئ چند جعبه مرزی ممکن است باقی بماند. یعنی ممکن است به FP ها اضافه کند چون برای یک شئ بیشتر از یک باکس در نظر می‌گیرد. به تعداد باکس‌های اضافه‌ای که شئ تشخیص داده است FP اضافه می‌شود.

سوال چهارم).....

مستطیل‌ها را در صفحهٔ مختصات رسم می‌کنیم تا راحت‌تر اشتراکشان را پیدا کنیم:



مساحت همهٔ مستطیل ها ۵۰ در ۵۰ یعنی ۲۵۰۰ است.

دو به دو اشتراک به اجتماع را حساب کرده و اگر از ۰.۵ بیشتر بود، وارد مرحلهٔ NMS می‌شود.

- **مستطیل نارنجی با مستطیل آبی و سبز اشتراکی ندارد، پس IoU آنها صفر است و به NMS احتیاجی ندارد.**

- **مستطیل نارنجی و صورتی:**
  - اشتراک: ۵۰ = (۱۰۰ - ۱۰۰) x (۱۱۰ - ۱۰۵)
  - اجتماع: ۴۹۵۰ = ۵۰ − ۲۵۰۰ + ۲۵۰۰
  - IoU: ۵۰/۴۹۵۰ << ۰.۵

چون IoU آنها بسیار از ۰.۵ کمتر است پس این دو مستطیل وارد مرحلهٔ NMS نمی‌شوند.

- **مستطیل سبز و آبی:**
  - اشتراک: ۲۲۵۰ = (۱۰۰ - ۵۰) x (۹۵ - ۵۰)
  - اجتماع: ۲۷۵۰ = ۲۲۵۰ − ۲۵۰۰ + ۲۵۰۰
  - IoU: ۲۲۵۰/۲۷۵۰ = ۰.۸۱

IoU این دو مستطیل از ۰.۵ بیشتر است، پس وارد مرحلهٔ NMS شده و بین آنها آنی که امتیاز بیشتری دارد انتخاب شده و دیگری حذف می‌شود. امتیاز مستطیل سبز ۰.۶ و امتیاز مستطیل آبی ۰.۹ است؛ پس مستطیل سبز حذف می‌شود.

- **مستطیل صورتی و آبی:**
  - اشتراک: ۱۸۰۰ = (۱۰۰ - ۶۰) x (۱۰۰ − ۵۵)
  - اجتماع: ۳۲۰۰ = ۱۸۰۰ − ۲۵۰۰ + ۲۵۰۰
  - IoU: ۱۸۰۰/۳۲۰۰ = ۰.۵۶

چون IoU این دو از ۰.۵ بیشتر است، وارد مرحلهٔ NMS شده و چون امتیاز آبی از صورتی بیشتر است، مستطیل آبی مانده و صورتی حذف می‌شود.

پس در آخر فقط مستطیل آبی و نارنجی باقی می‌مانند.

لیست جعبه‌های باقی‌مانده:

(0.9, (50, 50, 100, 100)), (0.7, (100, 100, 150, 150))

سوال پنجم).....

$$b_x\ in\ pixels = b_x \times\ width = 0.5\ \times 416 = 208\ pixels$$

$$b_y\ in\ pixels = b_y \times height = 0.6\ \times 416 = 249.6 = 250\ pixels$$

$$b_w\ in\ pixels = b_w \times\ width = 0.3\ \times 416 = 124.8 = 125\ pixels$$

$$b_h\ in\ pixels = b_h \times height = 0.4\ \times 416 = 166.4 = 166\ pixels$$

یعنی مرکز این باکس در مختصات (208, 250) قرار دارد و عرض آن 125 پیکسل و ارتفاعش 166 پیکسل است.

سوال ششم).....

شئ مورد بررسی کتاب است. لیبل‌گذاری برای تصاویر با کمک roboflow انجام شده است.

برای تهیهٔ دیتاست از برخی از تصاویر این دیتاست استفاده شده است:

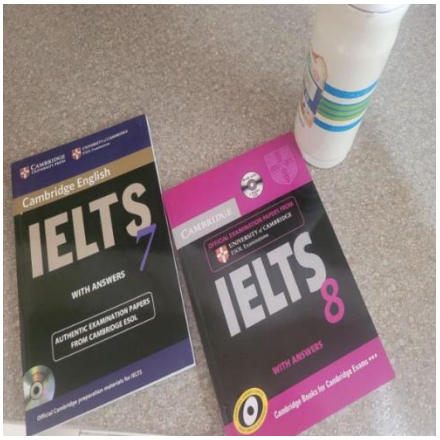
https://universe.roboflow.com/yolov8-xddjz/buku

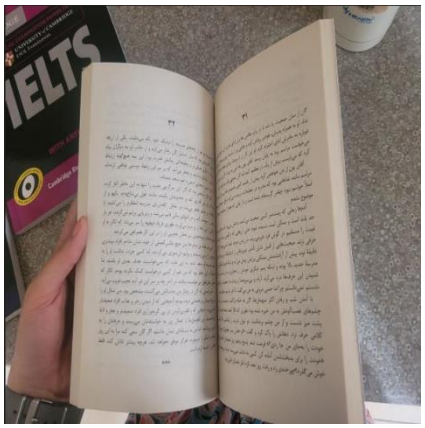
همچنین حدود ۵۰ تا تصویر نیز خودم تهیه کردم که مجموع آن با برخی از تصاویر دیتاست بالا و اعمال data augmentation دیتاست زیر را تولید کرد که از آن برای train مدل استفاده شده است که در google drive آپلود شده است و public است و فولدر train آن دارای تقریبا ۱۱۲۰ تا تصویر است:

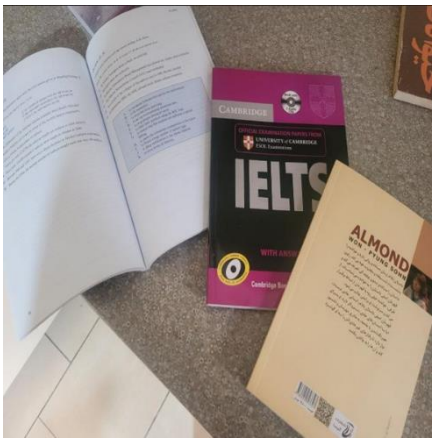
https://drive.google.com/file/d/1O4HK9K84TK9-I1g32j0UT7sNE-0zfkRt/view?usp=sharing

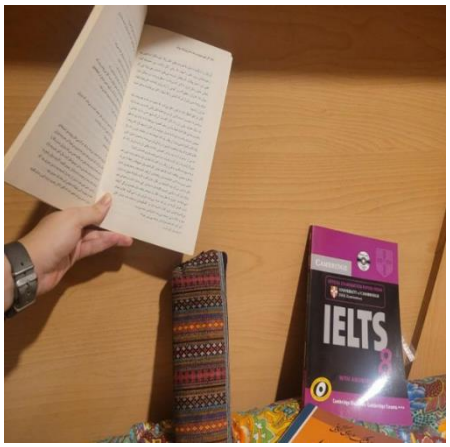
لیبل باکس‌های تصاویر خودم را buku گذاشتم تا مثل لیبل‌های تصاویر دیتاست ذکر شده شود.

برخی از تصاویری که خودم گرفتم :

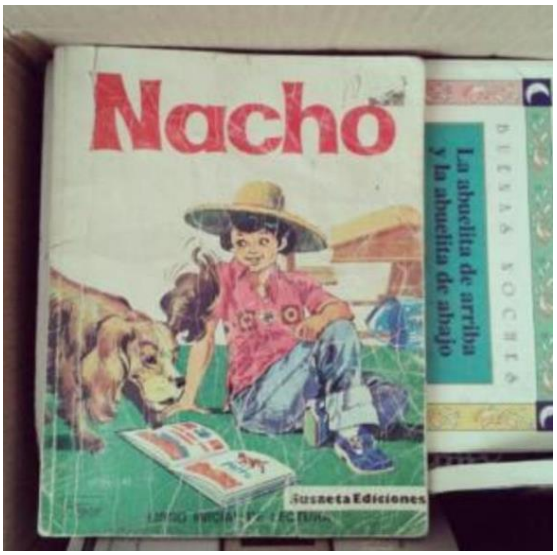


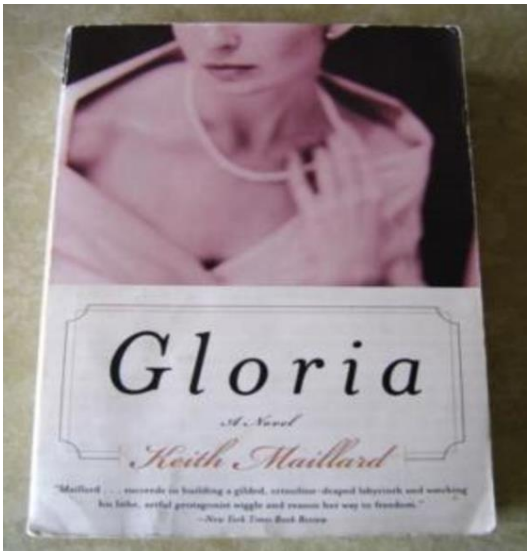


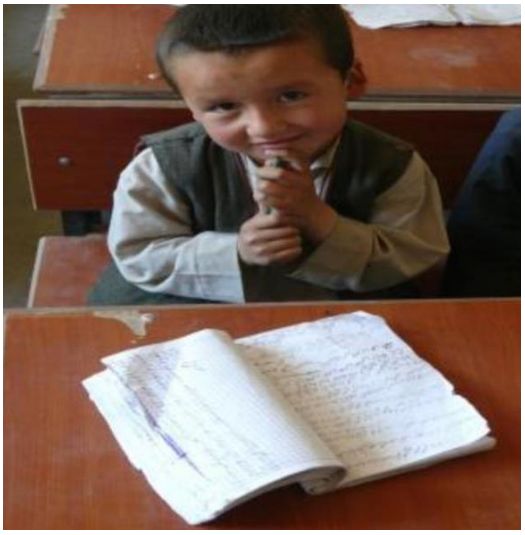




برخی از تصاویر دیتاست آماده:







مشکلات اولیه: ابتدا از کل تصاویر یک دیتاست بزرگتر استفاده شد، ولی چون خود دیتاست augmentation داشت و yolov8 هم موقع train خودش هم یک سری داده‌افزایی اعمال می‌کند دچار مشکل می‌شد و نمی‌توانست به خوبی تشخیص دهد. برای همین از این دیتاست که نسبت به قبلی کمتر بود استفاده کردم و با تصاویر خودم ترکیب کردم و مقدار augmentation هم کمتر کردم تا مدل دچار underfit نشود. سپس دیتاست نهایی به صورت فشرده شده را در drive آپلود کردم و با کمک کد زیر آن را داخل colab لود کردم:

```
# mount Google Drive
from google.colab import drive
```

```
 # necessary libraries
 !pip install ultralytics
 !pip install opencv-python-headless
 !pip install google-auth google-auth-oauthlib google-auth-httplib2
```

فایل زیپ شده را با کد زیر unzip کردم:

```
import zipfile
import os

# path to zip file
zip_file_path = '/content/drive/MyDrive/bookdata.zip'

# directory to extract to
extraction_path = '/content/datasets'

# create extraction directory if not exists
os.makedirs(extraction_path, exist_ok=True)

# unzip the file
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(extraction_path)
```

سپس با کد زیر، مدل YOLO را استفاده کرده و با کمک فایل yaml داخل دیتاست، مدل مسیر تصاویر و لیبل‌ها را پیدا کرده و آموزش می‌بینند. چون تعداد داده‌ها زیاد نبود، تعداد epoch ها را بیشتر کردم:

```
import torch
torch.cuda.empty_cache()
from ultralytics import YOLO

# path to the dataset
data_path = extraction_path
yaml_path = f'{data_path}/buku/data.yaml'


# initialize and train the model with the modified configuration
model = YOLO("yolov8n.pt")

# combine the dictionary with the path to the data
results = model.train(data=yaml_path, epochs=60, imgsz=640)
```

نتیجه آموزش:

```
 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
57/60      2.25G     0.4939     0.3304      1.149         12        640: 100%|██████████| 71/71 [00:27<00:00,  2.58it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 7/7 [00:02<00:00,  2.71it/s]
             all        202        267      0.821      0.805      0.844      0.673

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
58/60      2.24G     0.4801     0.3186      1.135         17        640: 100%|██████████| 71/71 [00:26<00:00,  2.64it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 7/7 [00:04<00:00,  1.69it/s]
             all        202        267       0.82      0.813      0.855      0.672

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
59/60      2.25G     0.4706     0.3086      1.122         16        640: 100%|██████████| 71/71 [00:23<00:00,  2.97it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 7/7 [00:02<00:00,  3.24it/s]
             all        202        267      0.857      0.787       0.86      0.674

 Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
60/60      2.25G     0.4644     0.3157      1.129         14        640: 100%|██████████| 71/71 [00:25<00:00,  2.83it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 7/7 [00:03<00:00,  1.81it/s]
             all        202        267      0.829      0.815      0.861      0.682

ochs completed in 0.514 hours.
izer stripped from runs/detect/train2/weights/last.pt, 6.3MB
izer stripped from runs/detect/train2/weights/best.pt, 6.3MB

ating runs/detect/train2/weights/best.pt...
lytics YOLOv8.2.48 🚀 Python-3.10.12 torch-2.3.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
 summary (fused): 168 layers, 3005843 parameters, 0 gradients, 8.1 GFLOPs
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 7/7 [00:05<00:00,  1.25it/s]
             all        202        267      0.829      0.815       0.86      0.684
: 0.3ms preprocess, 3.2ms inference, 0.0ms loss, 7.0ms postprocess per image
ts saved to runs/detect/train2
```

بهترین وزن پیدا شده هم (معیار سنجش AP است) داخل دیرکتوری runs/detect ذخیره می‌شود. با این کد مدل آموزش دیده را با وزن‌های پیدا شده load می‌کنیم. همانطور که در کد هم کامنت شده، دیکتوری آخرین

ران را باید لحاظ کنیم. من چون دو بار ران گرفتم دو بار مدلم آموزش دید، وزن‌های ران آخر را که در دیرکتوری runs/detect/train2 است لود می‌کند.

```
import cv2
import numpy as np
from IPython.display import display, Javascript, Image
from google.colab.output import eval_js
from base64 import b64decode, b64encode

# path to the trained model
trained_model_path = '/content/runs/detect/train2/weights/best.pt' # it should be set to the last train values, the default
                                                                   # is 'train', but because I ran it 2 times it is set to 'train2'
                                                                   # should open the runs/detect directory and set it to last folder name in it.

# load the trained model
model = YOLO(trained_model_path)

def js_to_image(js_reply):
    """
    Convert image from JavaScript as Base64 to OpenCV image
    """
    image_bytes = b64decode(js_reply.split(',')[1])
    jpg_as_np = np.frombuffer(image_bytes, dtype=np.uint8)
    img = cv2.imdecode(jpg_as_np, flags=1)
    return img

def bbox_to_bytes(bbox_array):
    """
    Convert bounding box image to bytes
    """
    bbox_PIL = Image.fromarray(bbox_array, 'RGB')
    iobuf = io.BytesIO()
    bbox_PIL.save(iobuf, format='jpeg')
    bbox_bytes = '' + b64encode(iobuf.getvalue()).decode('utf-8')
    return bbox_bytes
```

حال برای اینکه مدل را روی وبکم به صورت لایو تست کنیم، به یک کد javascript نیاز داریم. کد مربوط به این بخش با کمک این لینک زده شده است.

```
# JavaScript code embedded in Python to handle video streaming and image capture
def video_stream():
    js = Javascript('''
        // initialization of variables and elements for video streaming and image capture
        var video;
        var div = null;
        var stream;
        var captureCanvas;
        var imgElement;
        var labelElement;

        var pendingResolve = null;
        var shutdown = false;

        // function to remove DOM elements and stop video stream
        function removeDom() {
            stream.getVideoTracks()[0].stop();  // Stop the video stream
            video.remove();  // Remove the video element from DOM
            div.remove();   // Remove the div container from DOM
            video = null;   // Clean up variables
            div = null;
            stream = null;
            imgElement = null;
            captureCanvas = null;
            labelElement = null;
        }

        // function to handle animation frame updates
        function onAnimationFrame() {
            if (!shutdown) {
                window.requestAnimationFrame(onAnimationFrame);  // request next animation frame
            }
            if (pendingResolve) {
                var result = "";
                if (!shutdown) {
                    captureCanvas.getContext('2d').drawImage(video, 0, 0, 640, 480);  // capture frame from video stream
                    result = captureCanvas.toDataURL('image/jpeg', 0.8);  // convert captured frame to JPEG format
                }
                var lp = pendingResolve;
                pendingResolve = null;
                lp(result);  // resolve promise with captured image data
            }
        }
```

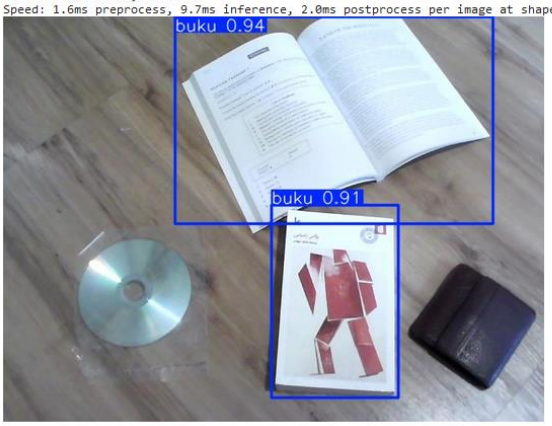
چون کدش زیاد است داخل داک نذاشتم ولی داخل فایل کد به صورت کامل و با کامنت موجود است.

این کد بعد از روشن شدن وبکم، هر چند میلی ثانیه یک frame از ویدیو را به مدل می‌دهد تا کتاب‌ها را پیدا کند. برخی از فریم‌ها در ادامه آورده شده است.

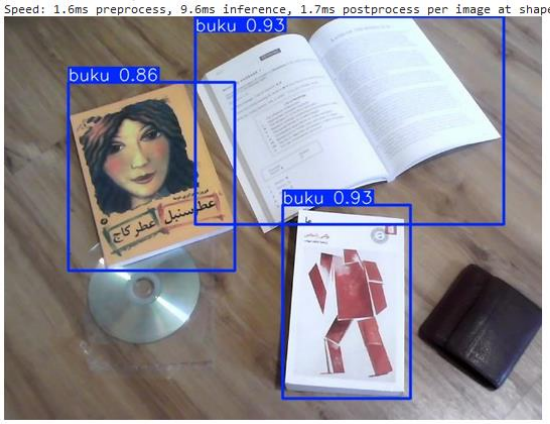
ران دوم:

0: 480x640 1 buku, 9.4ms
Speed: 1.6ms preprocess, 9.4ms inference, 2.5ms postprocess per image at shape (1, 3, 480, 640)

0: 480x640 2 bukus, 9.7ms
Speed: 1.6ms preprocess, 9.7ms inference, 2.0ms postprocess per image at shape (1, 3, 480, 640)

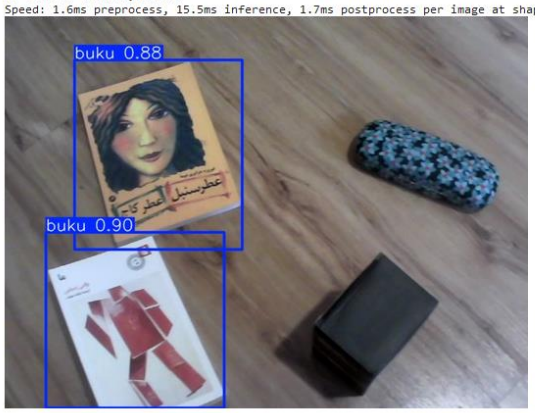


0: 480x640 3 bukus, 9.6ms
Speed: 1.6ms preprocess, 9.6ms inference, 1.7ms postprocess per image at shape (1, 3, 480, 640)

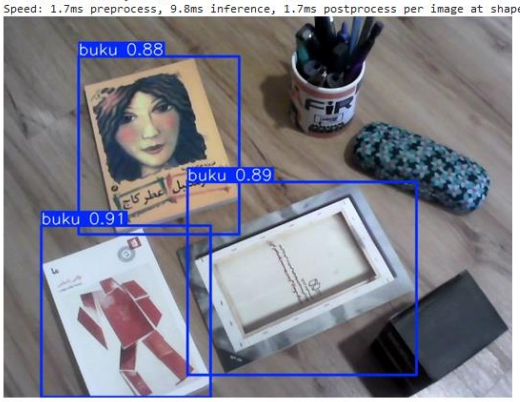


0: 480x640 2 bukus, 12.2ms
Speed: 2.7ms preprocess, 12.2ms inference, 1.8ms postprocess per image at shape (1, 3, 480, 640)

0: 480x640 2 bukus, 15.5ms
Speed: 1.6ms preprocess, 15.5ms inference, 1.7ms postprocess per image at shape (1, 3, 480, 640)



0: 480x640 3 bukus, 9.8ms
Speed: 1.7ms preprocess, 9.8ms inference, 1.7ms postprocess per image at shape (1, 3, 480, 640)



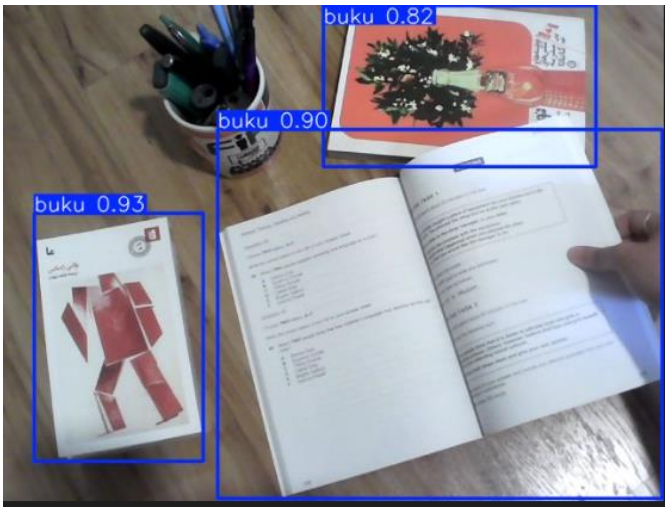
ران اول:

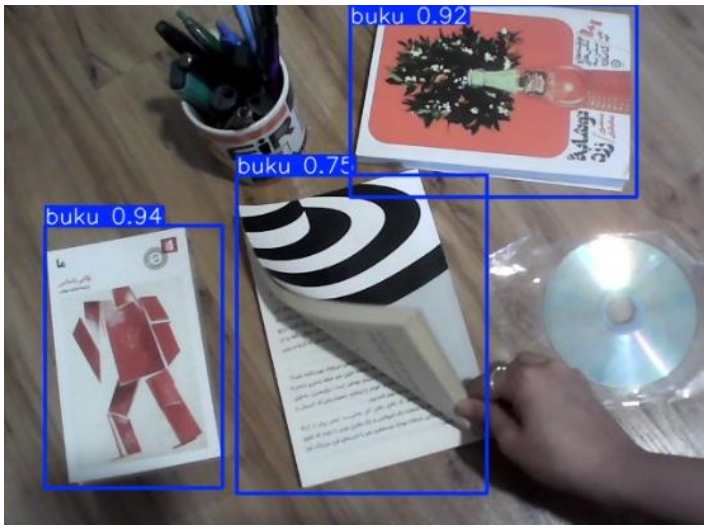

buku 0.88
buku 0.89
جورج اورل
قلعه حیوانات
عطرسنبل
عطر کاج


buku 0.91
buku 0.93
buku 0.92
buku 0.91
جورج اورل
قلعه حیوانات
عطرسنبل
عطر کاج


buku 0.93
buku 0.90
buku 0.92
buku 0.76
جورج اورل
قلعه حیوانات
عطرسنبل
عطر کاج


buku 0.93
buku 0.91
buku 0.88
جورج اورل
قلعه حیوانات


buku 0.82
buku 0.90
buku 0.93


buku 0.92
buku 0.75
buku 0.94

دو تا فایل Q6_firstrun و Q6_secondrin دارای کد مشابه هستند ولی چون مربوط به دو تا ران مختلف هستند فریم‌های خروجی‌شان فرق می‌کند که هر دو در فولدر تمرین گذاشته شده اند.

منابع:

- https://www.youtube.com/watch?v=IHbJcOex6dk
- https://www.youtube.com/watch?v=uvhPfKL6hOo&t=72s
- https://www.youtube.com/watch?v=ebAykr9YZ30&t=306s
- https://www.youtube.com/watch?v=YjWh7QvVH60&t=122s
- https://www.baeldung.com/cs/focal-loss