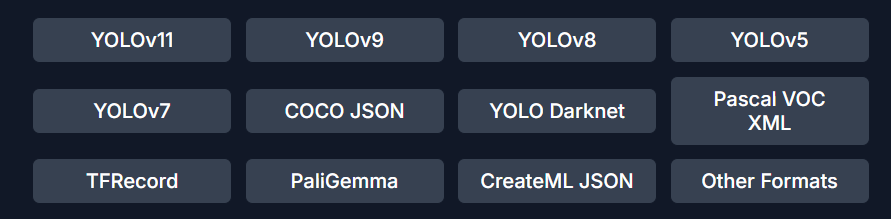
گزارش فاز دوم پروژه درس یادگیری عمیق – امیرحسین ملک‌ثابت، علیرضا نوری

1. **آماده‌سازی دیتاست و پیش‌پردازش داده‌ها**

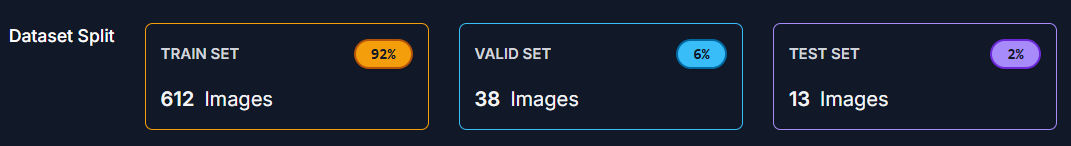
برای انجام این پروژه از دیتاست football-players-detection v1 در سایت Robowflow استفاده شده است. این دیتاست از نظر حجم و دقت برای پروژۀ پیش رو کاملا مناسب است و نتایج به خوبی این را نشان می‌دهد. لینک دیتاست جهت بررسی بیش‌تر:

<https://universe.roboflow.com/roboflow-jvuqo/football-players-detection-3zvbc/dataset/1>

جهت معرفی مختصر، اولا این دیتاست به طور خاص برای انواع ورژ‌ن‌های YOLO قرار داده شده است:



به همین دلیل با توجه به معماری (مثلا در پروژۀ ما YOLOv5) صرفا فرمت مورد نیاز را انتخاب کرده و دانلود می‌کنیم و نیاز به تغییر خاص دیگری ندارد. دیتاست شامل پوشه‌های train، test و valid است و هر کدام دو پوشۀ images و labels را دارا هستند. تعداد هر set در جدول زیر آمده است:



ثانیا، خود دیتاست augmentationهای مفیدی برای آموزش بهتر مدل ارائه کرده است که شامل horizontal flip، تغییر Saturation و تغییر brightness هستند. با این حال خود ما نیز پس از دانلود دیتاست به augmentation بیش‌تر پرداخته‌ایم. مثلا تعدادی از تصاویر را noisy کرده و همچنین به دلیل اهمیت brightness در سناریوی فوتبالی (برگزاری بازی‌ها در ساعات مختلف شبانه‌روز یا شرایط آب و هوایی متفاوت یا دوربین‌های با کیفیت متفاوت) دوباره تعدادی از setها را با brightness تغییر یافته در داده قرار داده‌ایم. این کار به آموزش بهتر مدل کمک زیادی کرده است.

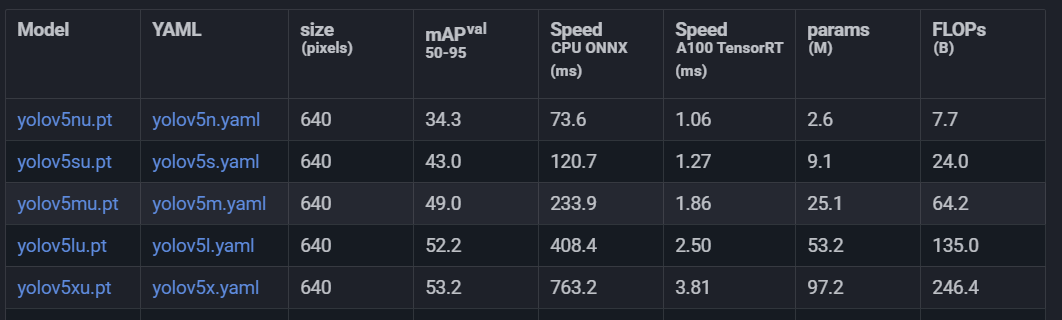
1. **پیاده‌سازی الگوریتم ردیابی اشیا**

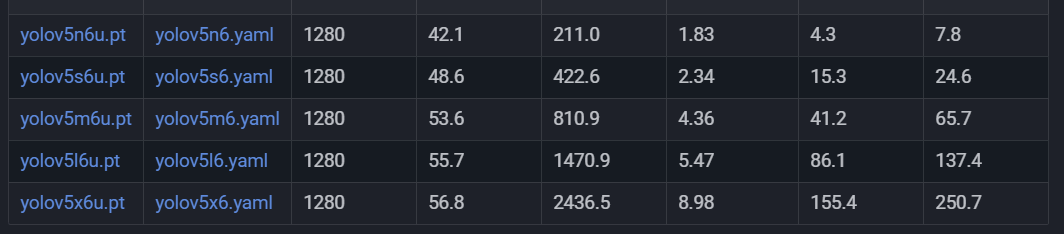
**1.2. تشخیص اشیاء**

**1. در این بخش الگوریتم تشخیص منتخب خود را معرفی می‌کنیم**. الگوریتم تشخیص اشیاء انتخابی ما در این پروژه، مدل YOLO است. YOLO مخفف You Only Look Once است. دلیل اصلی ما برای انتخاب این مدل در حقیقت سناریویی است که در این پروژه روی آن کار می‌کنیم. YOLO به دلیل سرعت بالای تشخیص خود، به طور خاص برای کاربردهای آنلاین و یا سناریوهایی که در آن‌ها objectها سرعت بالایی دارند و محیط بسیار dynamic است مناسب است. تشخیص بازیکنان، داور و سخت‌ترین مورد یعنی توپ در یک زمین فوتبال نیاز سرعت بالای پردازش است که مدل‌هایی مانند Faster R-CNN با اینکه دقیق‌تر هستند، در آن به مشکل می‌خورند. بنابراین برای tracking مناسب بازیکنان یا توپ یک detector سریع نیاز داریم که tracker بتواند با دریافت به موقع detectionها پایدار باقی بماند.

مدل YOLO در حقیقت یک مدل single-shot است. به این معنا که **یک** شبکۀ عصبی کانوولوشنی مستقیما تصویر ورودی را به predictionها در یک forward path تبدیل می‌کند. در حالی که مدل‌های کلاسیک مانند Faster R-CNN پردازشی دو stageای دارند. مثلا خود Faster R-CNN یک معماری است که بر اساس region proposalها عمل می‌کند. بدین صورت که ابتدا region proposalها تولید می‌شوند، سپس طبقه‌بندی شده و در نهایت bounding boxها بهبود می‌یابند. جدای از جزییات، معنای این کار این است که تصویر دوبار پردازش می‌شود به همین دلیل سرعت پردازش مناسب سناریویی مانند سناریوهای ورزشی (و به طور خاص ردیابی توپ) نیست. انتخاب YOLO انتخاب معقول‌تری است.

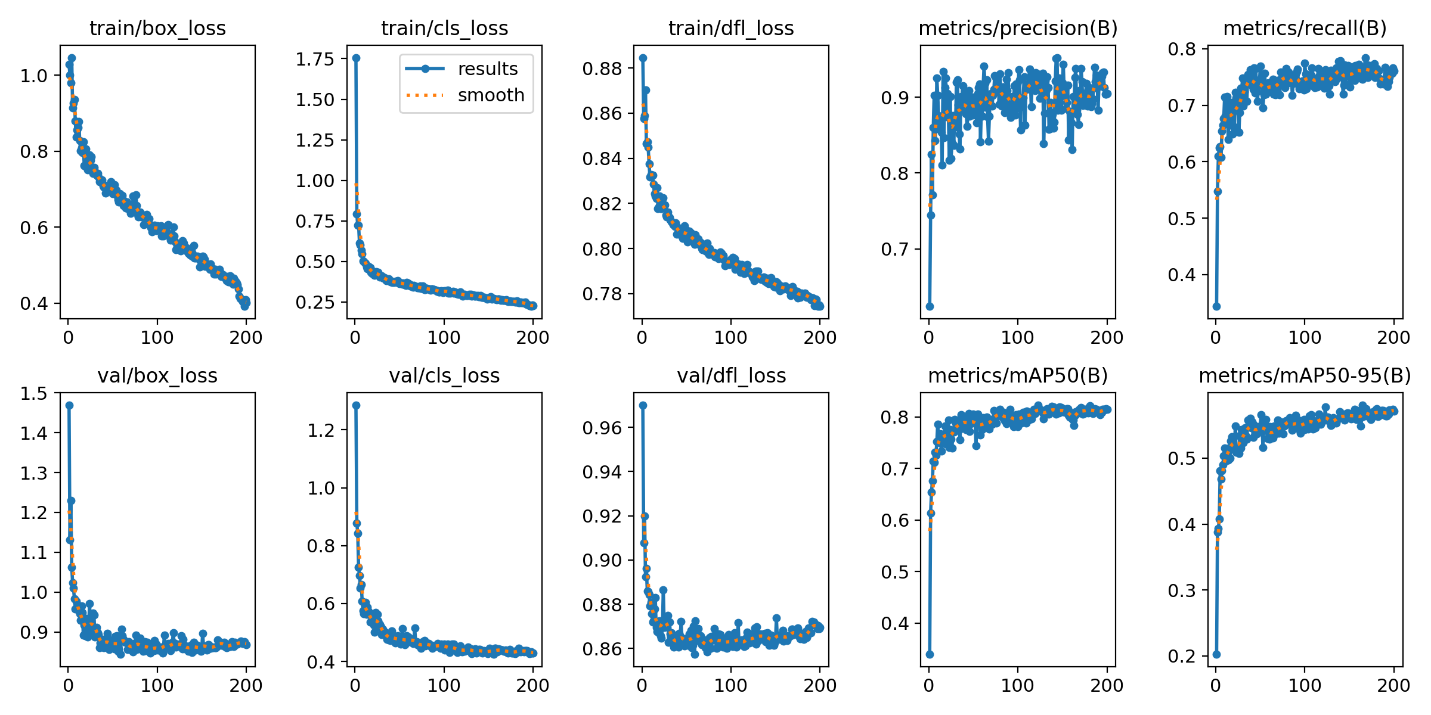
**2. در این بخش مدل خود را آموزش داده و نمودار Loss را می‌بینیم.** مدل اجراشدۀ ما مدل YOLO v5 است. برای پیاده‌سازی و آموزش مدل از کتابخانۀ ultralytics استفاده کرده‌ایم. کدهای مربوط به دانلود دیتاست و آموزش yolo در yolo.ipynb در پوشۀ main codes قابل مشاهده است. معماری backbone این مدل از نظر تعداد پارامترها می‌تواند کوچک، متوسط یا بزرگ باشد. لیست مدل‌ها از نظر تعداد پارامتر و دقت در جدول زیر آمده است:



****

ما چندین مدل مختلف را انتخاب کردیم و روی دیتاست خود آموزش دادیم. معماری انتخابی نهایی ما yolov5s6u با تعداد 15m پارامتر است. طبعا هر چقدر مدل بزرگ‌تری انتخاب کنیم نتیجۀ بهتری می‌گیریم ولی از نظر محدودیت منابع، بهترین مدلی که توانستیم آموزش بدهیم این مدل بود. که البته، نتیجه بسیار خوب است و به مدل بزرگ‌تری هم در حد این پروژه نیازی نبود.

مدل در دویست epoch آموزش داده شد در ذیل می‌توان نمودار Loss و دقت‌ها در فرآیند آموزش را دید. خود کتابخانۀ ultralytics نمودارهای زیر را (و همینطور سایر تصاویر که در سوالات بعدی می‌آید) را برای ما در پوشۀ runs ذخیره کرده است. در yolo.ipynb نیز نمودارها plot شده‌اند.



توجه شود که مدل نهایی ذخیره و استفاده شده، مدل بعد از epoch آخر نیست. کتابخانۀ ultralytics دو مدل last و best را تحویل می‌دهد که last آخرین مدل به دست آمده و best بهترین مدل موجود در کل فرآیند training است. به همین دلیل در انتخاب تعداد epoch دست باز عمل کرده‌ایم و در نهایت بهترین مدل را برداشته‌ایم.

**3. در این بخش خروجی مدل را روی چند فریم از ویدئو نمایش می‌دهیم.** خروجی yolo را روی یک ویدئوی سی ثانیه‌ای از بوندیس لیگا بررسی می‌کنیم و ببینیم آیا مدل آموزش دیده می‌تواند بازیکنان، توپ یا داور را به خوبی تشخیص دهد یا خیر. لینک ویدئو:

http://\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

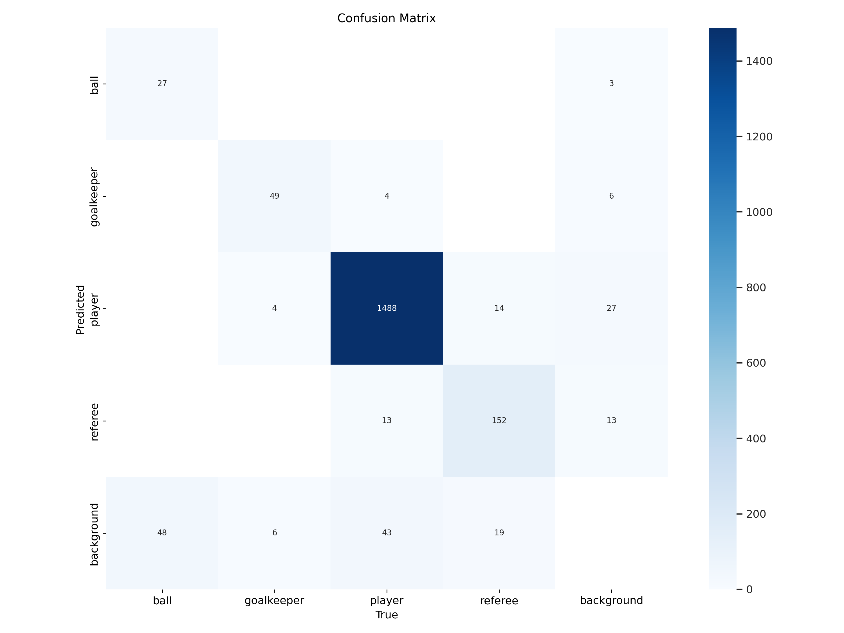
پس از عبور frameهای آن از YOLO و ذخیرۀ خروجی در پوشۀ predict دو فریم خروجی را با هم می‌بینیم (در notebook نیز پلات شده‌اند.)



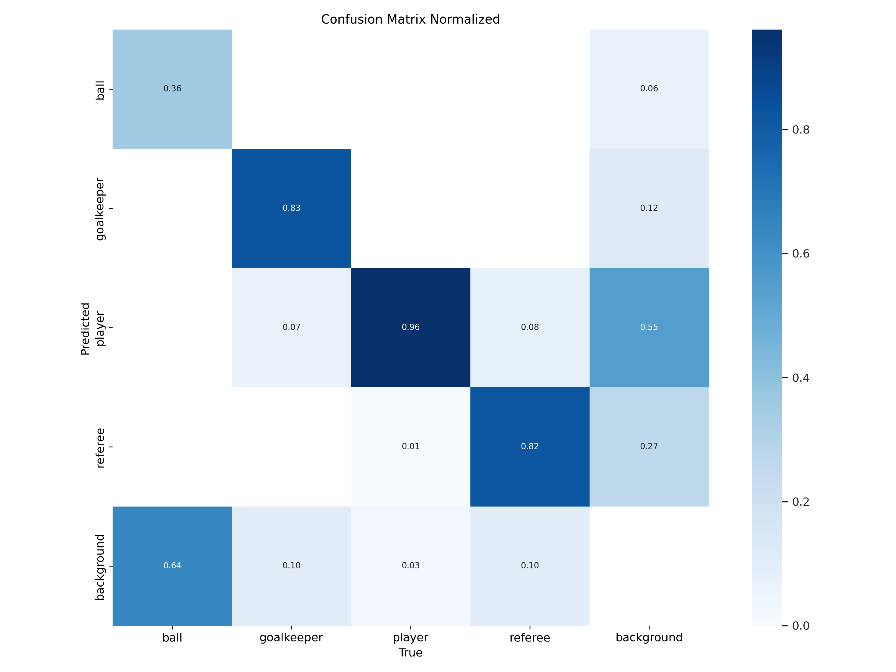
مشخص است که به خوبی detection انجام می‌شود. لینک دانلود خروجی با prediction روی تمامی فریم‌ها:

http://\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

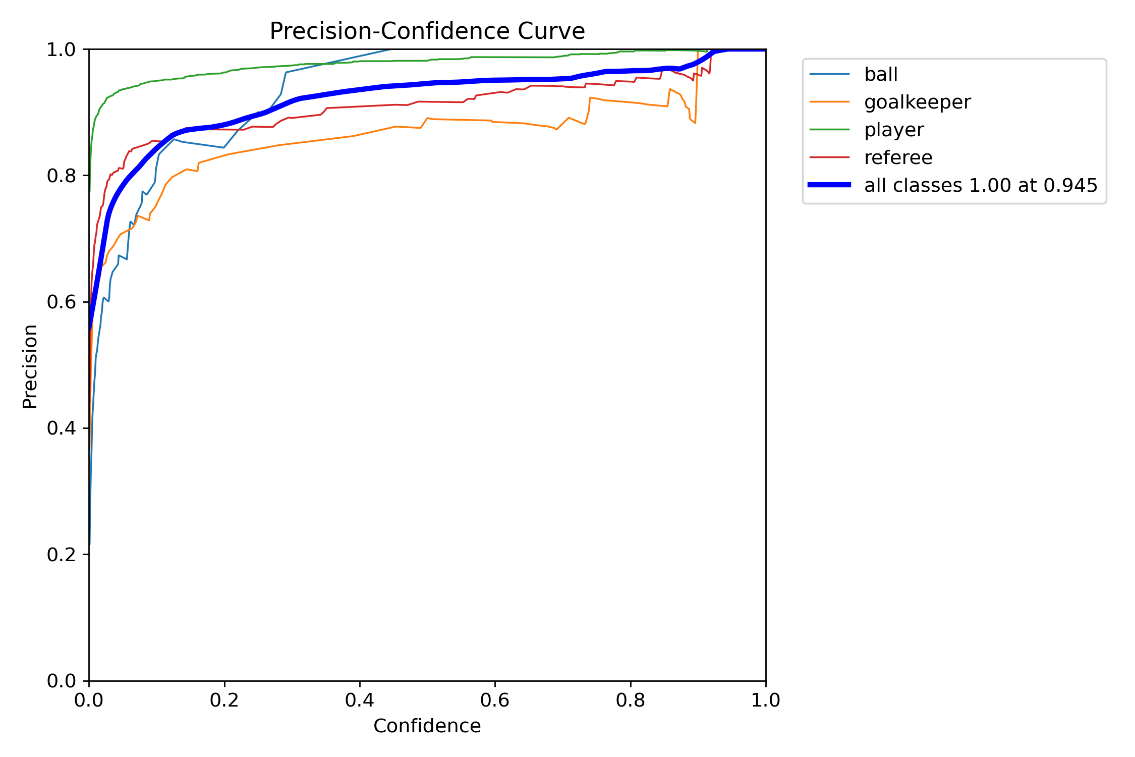
**4. در این قسمت دقت و عملکرد مدل را با استفاده از معیارهای مختلف بررسی می‌کنیم.** تمام نمودارها و شکل‌های این قسمت در پوشۀ runs ذخیره شده‌اند. اولین معیار ماتریس confusion است:

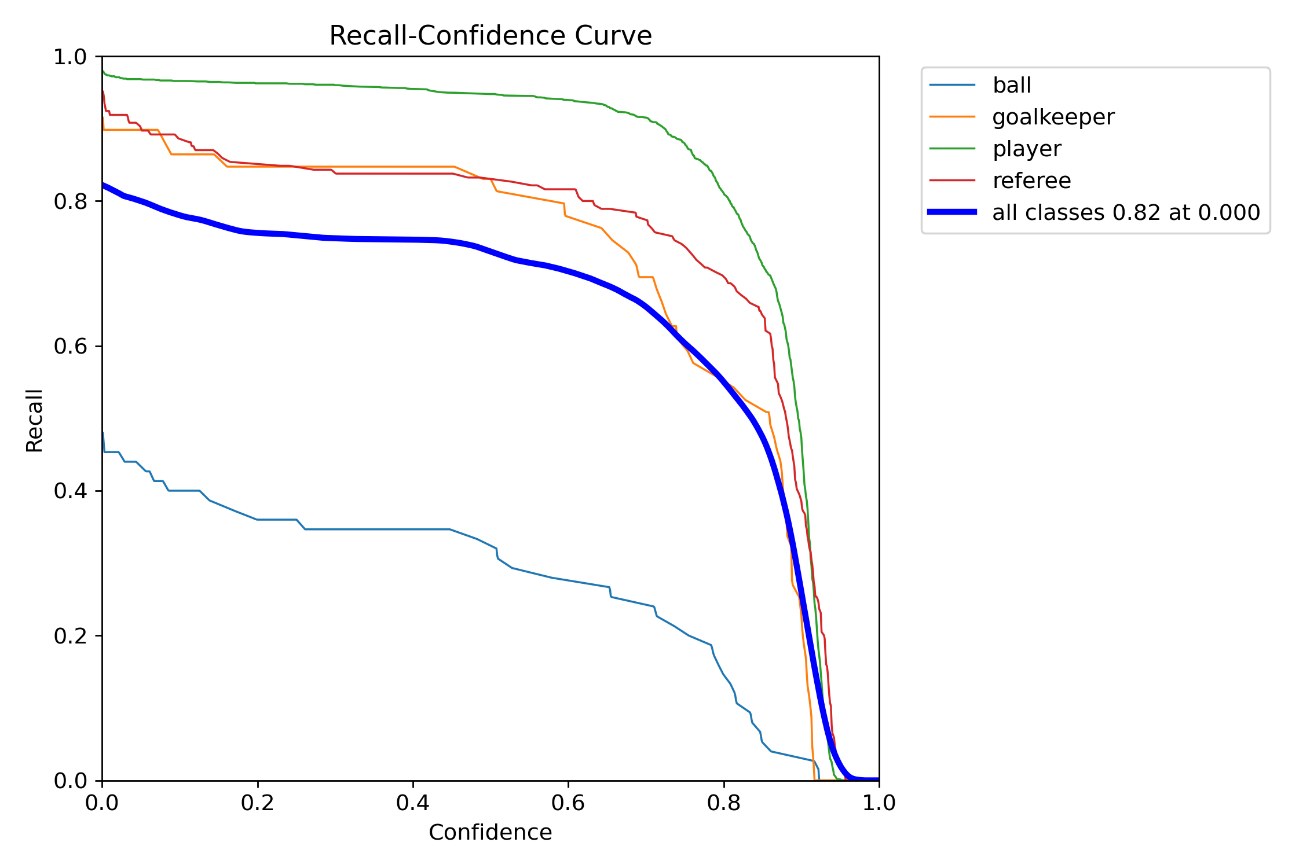


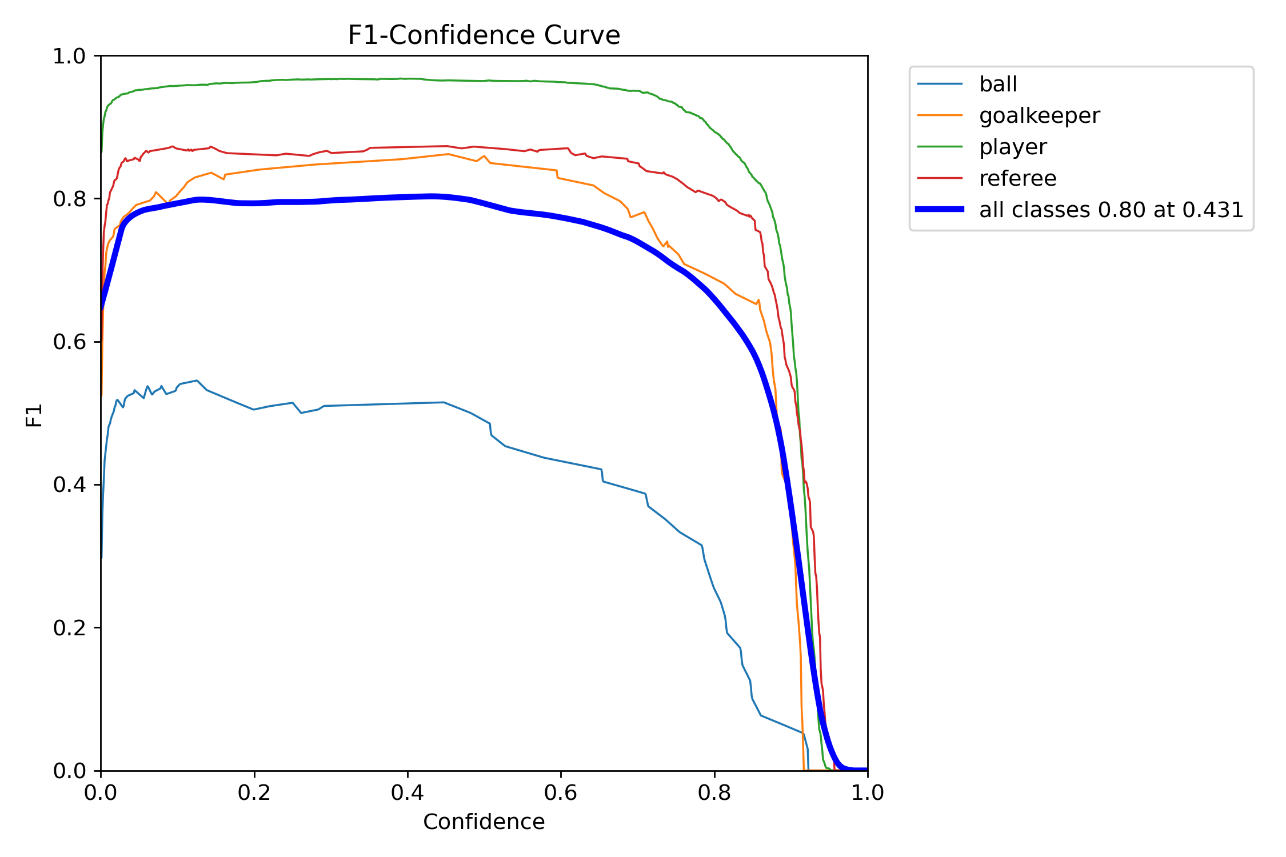
دقت مدل خوب است و در بسیاری از موارد تشخیص به درستی انجام شده. اما بررسی اشتباهات آموزنده است. همانطور که مشخص است بیش‌ترین چالش توپ است که 48 بار به اشتباه تشخیص داده نشده (تشخیص background). این مورد به دلیل سایز کوچک توپ نسبت به بقیۀ objectها است که باعث می‌شود به راحتی گم شود. توپ در بخش Trackerها نیز مشکلاتی دیگر ایجاد می‌کند و سرعت بالای آن باعث می‌شود Track کردن آن سخت شود. مورد قابل توجه دیگر داورها هستند که چندین بار به جای بازیکن یا background اشتباه گرفته شده‌اند. تعداد تشخیص‌ اشتباه background به حای یکی از objectها نیز وجود دارد که در برابر تشخیص‌های درست (مثل تعداد تشخیص درست بازیکنان) ناچیز هستند. ماتریس confusion نرمالیزه شده را نیز می‌بینیم:



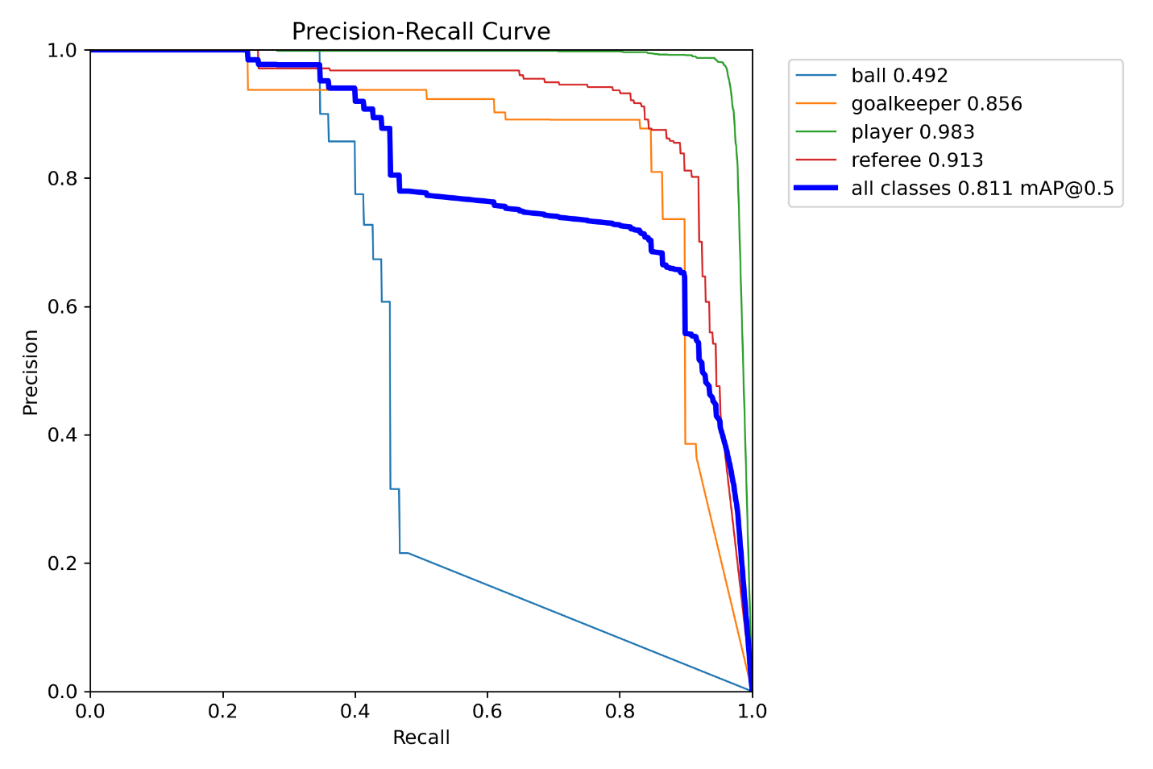
در زیر نمودارهای precision، recall و F1-score که ترکیبی از دو مورد اول است را بر حسب confidence threshold می‌بینیم.







در نهایت، نمودار precision بر حسب recall:



**2.2. ردیابی تک‌شیء (Single Object Tracking)**

1.

2.

3.

4.

5.

6.

**3.2. ردیابی چند شیء (Multiple Object Tracking)**

**1. در این بخش الگوریتم ردیابی چند شیء خود را انتخاب می‌کنیم.** الگوریتم انتخابی ما برای ردیابی چند شیء الگوریتم ByteTrack است. این الگوریتم در حال حاضر الگوریتم state-of-the-art است. این الگوریتم در ادامۀ الگوریتم‌های SORT و DeepSORT تکامل یافته است و ایدۀ اصلی آن استفاده از تشخیص‌های low-confidence در کنار تشخیص‌های high-confidence است. تغییری کوچک ولی بسیار کمک‌کننده برای ردیابی اشیاء. این ویژگی الگوریتم را بسیار برای سناریوهایی مانند سناریوهای ورزشی که اشیاء سریع حرکت می‌کنند و مرتبا دچار پوشیدگی می‌شوند مناسب می‌کند. به همین دلیل انتخاب ما برای این پروژه است. نکتۀ دیگر این است که الگوریتم به اندازۀ الگوریتم SORT سریع است. با این فرق که همزمان دقت بسیار خوبی نیز دارد و همچنین به دلیل ماهیت الگوریتمی خود، وزنی برای یادگیری ندارد که باعث می‌شود سرعت خوبی برای سناریوهای Real-Time داشته باشد.

**2. در این بخض ارتباط بین الگوریتم تشخیص و ردیابی را توضیح می‌دهیم.** مدل detection ما که مدل YOLO بود، با آمدن هر فریم در یک single shot در خروجی bounding boxed، class label و Confidence Score را تولید می‌کند. که ByteTrack از هر دو bounding box با confidence score پایین و بالا با دو آستانۀ مختلف و انتخابی استفاده می‌کند و فرآیند association را بین تشخیص‌های فریم جدید و قبلی انجام می‌دهد.

**3. در این بخش نحوۀ Assignment را توضیح می‌دهیم.** هدف assignment این است که detectionهای فریم جدید با detectionهای فریم قبلی match شوند تا عمل tracking انجام شود. ابتدا به کمک کالمن فیلتر قدم بعدی اشیاء پیش‌بینی می‌شود. سپس با داشتن bounding boxهای فریم جدید و فریم پیش‌بینی، IoU (میزان همپوشانی دو bounding box یا همان نسبت اشتراک به اجتماع) آن‌ها محاسبه می‌شود و در نهایت نتیجه گرفته می‌شود که دو box با بیش‎‌ترین IoU احتمالا مربوط به یک شیء هستند. و این کار مرتبا تکرار می‌شود تا زمانی که فرآیند ردیابی به پایان برسد. برای انجام مراحل اشاره شده از تکنیک Hungarian Method استفاده می‌شود که در حقیقت یک روش بهینه‌سازی برای حل مسائل assignment است. در مسئلۀ ردیابی، detectionهای فریم جدید باید با trackهای فریم قبلی match بشوند. و چون این کار با آمدن هر فریم جدید (یا بعد از تعدادی فریم) باید انجام شود، ممکن است از نظر محاسباتی بسیار هزینه‌بر باشد. به همین دلیل به Hungarian method نیاز داریم تا فرآیند assignment را به صورت بهینه حل کند. الگوریتم، مسئله را با کمینه کردن یک تابع هزینه ماتریسی با کم کردن سطرها/ستون‌ها به شکلی که از نظر محاسباتی بهینه باشد انجام می‌دهد.

**4. در این قسمت پیاده‌سازی الگوریتم را انجام می‌دهیم و نتیجه را در یک ویدئو نمایش می‌دهیم.** پیاده‌سازی الگوریتم در فایل انجام شده و نتیجه در output\_video.avi در لینک زیر قابل مشاهده است:

http://\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. **بهبود الگوریتم و ارائۀ روش‌های پیشنهادی**

**1.3. بخش اول – تحقیق**

**1. در این بخش روش‌هایی برای بهبود الگوریتم ردیابی پیشنهاد می‌دهیم.**

روش اول: همانطور که گفتیم، روش ByteTrack به کمک دو سطح آستانۀ انتخاب شده از detectionهایی با confidence score بالا و confidence score پایین بهره می‌گیرد تا tracking را انجام دهد. در ادبیات computer vision اینکه سطوح آستانه از قبل انتخاب می‌شوند و تا پایان اجرا ثابت می‌مانند خیلی جذاب نیست. تنها کاری که می‌شود در این شرایط انجام داد این است که چندین بار پروسه را تکرار کرد و سطوح آستانه‌ای که بهترین نتیجه را داشتند انتخاب کرد. که این کار با شرایط دنیای واقعی سازگار نیست. روش‌هایی پیشنهاد شده که بتوان به صورت داینامیک و تطبیقی در طول ویدئو سطوح آستانۀ مختلفی متناسب با شرایط انتخاب شوند که به نتیجۀ بهتری برسیم.

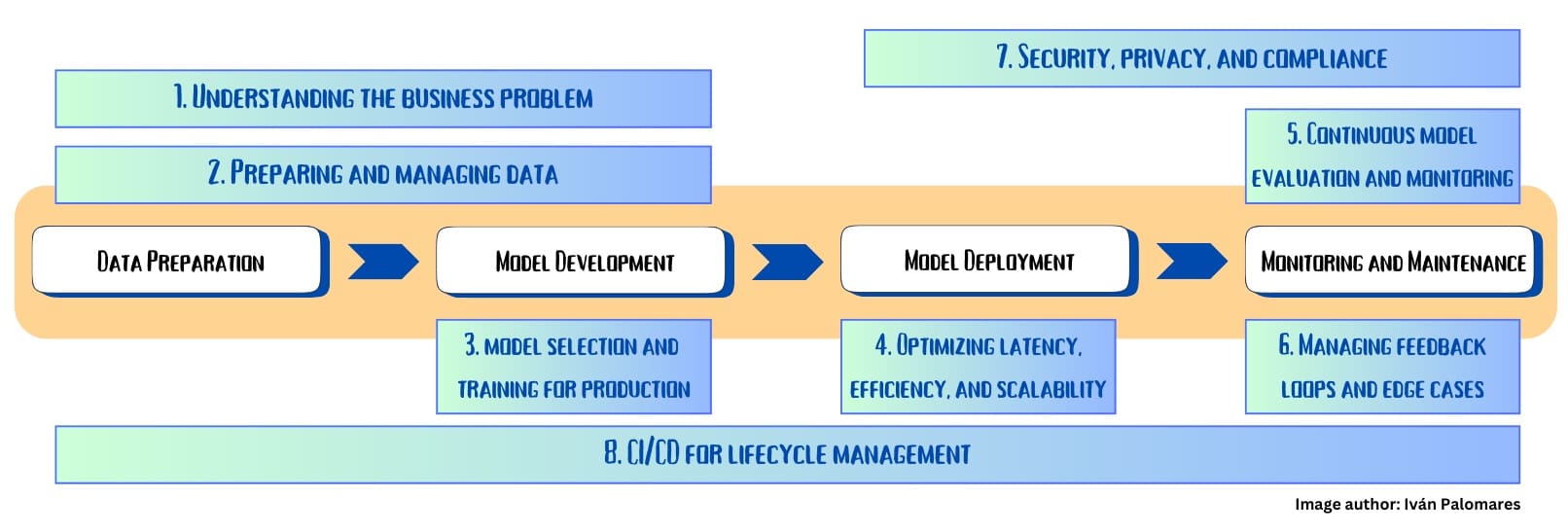
همانطور که قبلا گفتیم، در خروجی YOLO ما تمام confidence scoreها را داریم. بنابراین اگر الگوریتمی داشته باشیم، باید پس از دریافت خروجی‌های YOLO سطوح آستانه را انتخاب کند. یک کار برای انتخاب دینامیک سطح آستانه پیشنهاد شده است و آن این است که ابتدا confidence rateها را به صورت نزولی مرتب کنیم سپس به نحوی نقطۀ مناسب را بین این مجموعه پیدا کنیم. پس در ابتدا مجموعۀ نزولی زیر را داریم:

حال برای انتخاب سطح آستانه ابتدا یک تابع به این نقاط فیت می‌کنیم (مثلا در ساده‌ترین حالت ممکن، اتصال نقطه به نقطه) و سپس روی این تابع نقطۀ مناسبی را پیدا می‌کنیم. روش‌های مختلفی برای این کار می‌توان متصور شد، اما یک روش ساده که پیچیدگی و زمان‌بری زیادی نداشته باشد و به سرعت الگوریتم آسیب نزند این است که نقطه‌ای که در آن **بیش‌ترین سقوط** اتفاق می‌افتد را پیدا کنیم. یعنی:

و سطح آستانه را برابر قرار دهیم. بدین صورت که confidence rateهای بالاتر از این سطح آستانه high و سایر confidenceها low در نظر گرفته شوند. بدینصورت می‌توان زحمت انتخاب سطح آستانه را نداشت و همزمان به دقت بهتری در محیط‌های دینامیک پرچالش مانند سناریوهای ورزشی رسید و در برابر چالش‌هایی مانند پوشیدگی بهتر عمل کرد. زیرا انتخاب سطح آستانۀ مناسب در روش ByteTrack مستقیما به این مهم بستگی دارد.

روش دوم: روشی که در بخش بعدی پیاده‌سازی می‌کنیم در حقیقت این روش است. در اینجا تصمیم گرفتیم که با توجه به اینکه کاربرد مدل مشخص است (ردیابی اشیاء در فوتبال) سعی کنیم که خروجی بهتری با در نظر گرفتن خود سناریو داشته باشیم. تا این لحظه، الگوریتم ما بازیکنان، داور و توپ را شناسایی می‌کند و دور بازیکنان دایرۀ قرمز رنگ و دور داوران دایرۀ زرد رنگ کشیده و بالای توپ یک مثلث کوچک قرار می‌دهد. بنابراین بازیکنان هر دو تیم برای الگوریتم فرقی ندارند. به ذهن ما رسید که می‌توان مکانیزمی در الگوریتم قرار داد که الگوریتم اولا بازیکنان دو تیم را بشناسد، ثانیا دایره‌ای که بعد از تشخیص آن‌ها دور آن‌ها می‌کشد هم‌رنگ پیراهن تیم باشد. بدین ترتیب، خروجی بهتر و تمیزتر یا به عبارتی محصول بهتری خواهیم داشت. برای اینکار به طور خلاصه ابتدا bounding box مربوط به بازیکن را استخراج کرده‌ایم و سپس قسمت پایینی آن را حذف کرده‌ایم (زیرا در اکثر مواقع پیراهن بازیکن در بالای bounding box قرار دارد و برای تشخیص رنگ آن نیازی به نیمۀ پایینی bounding box) نداریم. حال که bounding box کات شده را داریم، اینگونه تحلیل می‌کنیم که تصویر فوق دارای دو خوشه است: 1. بازیکن و 2. پس‌زمینه. پس اگر یک الگوریتم خوشه‌بندی روی پیکسل‌های این bounding box جداشده اجرا کنیم، پیکسل‌های مربوط به بازیکن در یک خوشه و پیکسل‌های مربوط به پس‌زمینه در خوشۀ دیگر قرار می‌گیرند. برای این کار از KMeans کتابخانۀ sklearn استفاده کرده‌ایم. پس از استخراج خوشه‌ها می‌توان میانگین پیکسل‌های خوشۀ پیراهن بازیکن را به عنوان رنگ پیراهن بازیکن استفاده کرد. جزییات پیاده‌سازی دیگری نیز وجود دارند، مثلا مشخص نیست KMeans دو عدد صفر و یک را چگونه به دو خوشه اختصاص می‌دهد که باید بررسی شود. و سایر جزییات که تماما در کد در نظر گرفته شده‌اند و در کلاس TeamAssigner پیاده شده‌اند.

**2. در این بخش در مورد روش‌های optimization model صحبت می‌کنیم.** برای optimize کردن مدل و آماده کردن آن برای ارائه به عنوان یک محصول کارهای زیادی باید انجام داد که در stageهای مختلف فرآیند آماده‌سازی، تولید، آموزش و تست مدل انجام می‌شوند. اشتباه رایجی وجود دارد و آن این است که optimize کردن محصول فقط در زمانی انجام می‌شود که محصول می‌خواهد عملیاتی شود. اما در حقیقت، بهینه‌سازی از همان قدم‌های ابتدایی ساخت مدل یادگیری ماشین شروع می‌شود. درست از زمانی که یک مشکل برای حل شناسایی می‌شود. در نتیجه، ارتباط نزدیکی بین فازهای مختلف آماده‌سازی یک مدل یادگیری ماشین (مثلا در کیس ما، یک مدل MOT) و قدم‌هایی که می‌توان برای بهینه کردن مدل برداشت وجود دارد. منظور از فازهای مختلف 1. آماده‌سازی دیتاست 2. آموزش مدل 3. عملیاتی کردن پروژه 4. بررسی و نگه‌داری است. به طور خلاصه نمودار زیر می‌تواند در مورد روش‌های مختلف بهینه‌کردن محصول که در stageهای مختلف قابل اجرا هستند دید خوبی به ما بدهد. در ادامه بر بررسی بیش‌تر هر کدام خواهیم پرداخت.



1. درک مسئلۀ اقتصادی که در تلاش برای حل آن هستیم: درک عمیق مسئلۀ تعریف‌شده برای حل و تعریف اهداف اقتصادی بر اساس آن اولا مشخص می‌کند که آیا یک مدل یادگیری ماشین راه‌حل مناسبی هست یا خیر، و مهم‌تر از آن، اگر جواب سوال مثبت است، کمک می‌کند که مدل مناسب برای حل مسئله را انتخاب کنیم. آیا می‌خواهیم فروش را پیش‌بینی کنیم؟ مشتری‌ها را طبقه‌بندی کنیم؟ ویدئوی فوتبالی را آنالیز کنیم؟ ترافیک را بررسی کنیم؟ به محض اینکه این سوالات جواب داده شدند، می‌توانیم بفهمیم که کدام متریک‌ها در مسئله‌مان اهمیت بیش‌تری دارند: دقت، سرعت، هزینه و... مثلا در کیس پزشکی دقت دارای اهمیت بسیار بالایی است ولی در کیس ردیابی اشیا در سناریوهای ورزشی شاید ترجیح بدهیم کمی از دقت را فدای سرعت کنیم. این اطلاعات، ما را برای مهم‌ترین جنبه‌های مدل که باید بهینه باشند آماده می‌کند.
2. آماده‌سازی و مدیریت داده‌ها: بعد از درک عمیق مسئله، باید دنبال دیتاست‌های مرتبط باشیم. دیتاست‌هایی که مدل ما قرار است به آن‌ها تکیه کند. برای بهینه کردن مدل به عنوان یک محصول، مهم است که مدیریت دیتا در همان ابتدای امر انجام شود. این کار شامل automated data pipelines، استفاده از feature stores برای راحت کردن بازتولید و قابلیت scale کردن، و پیاده‌سازی مکانیسم‌هایی برای بررسی تضمین کردن کیفیت خوب داده‌ها و پیوستگی بین آن‌ها است.
3. انتخاب مدل و آموزش برای تولید محصول: تصمیماتی که برای انتخاب مدل، تنظیمات hyperparameters و سایر جزییات می‌گیریم بسیار در این مهم که مدل ما به عنوان یک محصول خوب عمل کند اهمیت دارند. سه ملاک با توجه به کاربرد باید balance شوند: 1. سادگی 2. کیفیت اجرا 3. تفسیرپذیری. با توجه به اولویتی که هر کدام از این موارد می‌توانند داشته باشند، مدل باید انتخاب شود. مثلا برای یک مسئلۀ طبقه‌بندی که کیفیت اجرا برای آن بسیار مهم است، شاید یک شبکۀ عصبی عمیق انتخاب خوبی باشد ولی در صورتی که سادگی و تفسیرپذیری مهم‌تر باشند، رگرسیون لجستیک یا درخت تصمیم‌گیری انتخاب بهتری باشد. جدای از انتخاب مدل، بسیار مهم است که مدل قابلیت تعمیم‌پذیری نیز داشته باشد و از مدل‌های بسیار پیجیده با واریانس بالا استفاده نشود.
4. بهینه کردن Latency، Efficiency، و Scalability مدل: بعد از اینکه یک ورژن خوب از مدل آموزش دید و validation و test روی آن انجام شد، حالا زمان عملیاتی کردن آن به عنوان یک محصول است! در این بخش باید به مواردی مانند کیفیت اجرا، سایز و حجم مدل و میزان تاخیر مدل توجه کرد. مثلا می‌توان از روش batch prediction استفاده کرد تا Efficiency مدل را بررسی کرد و آن را بهبود داد. برای کاهس سایز و تاخیر نیز روش‌هایی مانند model quantization وجود دارند که به هدف حذف وزن‌های زائد و بی‌اهمیت معرفی شده‌اند. به طوری که بتوان سایز و تاخیر مدل را به میزان قابل توجهی کاهش داد بدون آنکه در دقت کاهش قابل توجهی داشته باشیم. از آنجا که model quantization به خصوص در مدل‌های خیلی بزرگ امروزی مانند LLMها اهمیت زیادی دارند مهم‌ترین آنها را معرفی می‌کنیم:

* Post-Training Quantization: بعد از آموزش کامل مدل، می‌توان تعدادی از وزن‌ها را حذف کرد. نحوۀ انتخاب حذف وزن‌ها (مثلا برای شبکه‌های عصبی) به کمک داده و تست قابل انجام است. بدین صورت که مثلا نورون‌های که در اکثر موارد غیرفعال هستند را از شبکه حذف می‌کنیم. یکی از مشکلات این روش این است که در صورتی که نتوان به درستی نورون‌ها یا وزن‌های دارای اهمیت کمتر را تشخیص داد، با حذف وزن‌هایی که ممکن است بیش‌تر از تصور ما اهمیت داشته باشند دقت زیادی را از دست بدهیم. پس باید محتاطانه از این روش استفاده کنیم. نکته دیگر هم این است که حتی اگر کاملا توانایی مقایسه اهمیت وزن‌ها را داشته باشیم، باز هم انتخاب بالانس مناسب بین دقت و سایز کاریست Ad-Hoc و بدون ریاضیات خاص و دقیقی. بسیار هم با کاربردهای مختلف تغییر می‌کند.
* Quantization-Aware Training: این روش عمل Quantization را وارد خود پروسۀ آموزش می‌کند. بدین صورت که در فاز forward بخشی از وزن‌ها حذف شده و مدل خودش یاد می‌گیرد که با این کار تطبیق پیدا کند. این کار معمولا به دقت بالاتری نسبت به روش قبلی دست پیدا می‌کند.
* روش‌های دیگری نیز وجود دارند. مثلا روش‌های Uniform Quantization و Non-Uniform Quantization که بسیار در آموزش مدل‌های بزرگ زبانی استفاده می‌شوند. بدین صورت که بازۀ هر وزن را به صورت برابر یا نابرابر با سایر وزن‌ها به بازه‌های مختلف تقسیم می‌شوند. همچنین می‌توان تمام روش‌های گفته شده را باهم ترکیب کرد که به این کار Hybrid Quantization می‌گویند

1. Evaluation و Monitoring پیوستۀ مدل: بعد از عملیاتی شدن مدل، Evaluating و Monitoring پیوستۀ مدل دو مورد بسیار مهم هستند که باید به آن‌ها توجه شود. یکی از کارها راه‌اندازی سیستمی است که data drift در طول زمان را monitor کند. زیرا طبیعی است که در طول زمان با داده‌هایی با توزیع متفاوت روبرو شویم. همچنین tracking performance degradation در مواجهه با داده‌های واقعی اهمیت زیادی دارد. بعد از بررسی‌ پیوسته، در زمان برخورد با مشکل باید Automating model retraining انجام شود. در حقیقت فهمیدن زمانی که باید مدل retain شود خودش می‌تواند یک چالش بزرگ باشد. دو نگرش کلی برای این کار وجود دارد: 1. Fixed و 2. Dynamic که در fixed یک دوره مشخص برای retain کردن مشخص می‌شود (مثلا روزانه، هفتگی، ماهانه) و در dynamic این دورۀ فیکس وجود ندارد و بر اساس کیفیت مدل تصمیم هر زمان که احساس نیاز شد تصمیم گرفته می‌شود مدل retain بشود. روش fixed با آنکه ساده و مشخص است اما نقاط ضعفی دارد. ممکن است هزینۀ بیش‌تر از آن چیزی که واقعا باید بشود داشته باشیم و retain کردن دوباره و دوباره باعث شود inconsistency بین مدل‌ها داشته باشیم. در حالت داینامیک مثلا می‌توان بر اساس metric تصمیم گرفت. می‎‌توان با monitor کردن پیوستۀ Prediction Drift، Feature Drift و... به محض اینکه این driftها از حد مشخصی بیش‌تر شد، پروسۀ retaining انجام شود.
2. مدیریت حلقه‌های فیدبک و Edge Cases: حلقه‌های فیدبک برای بهبود عملکرد مدل بسیار اهمیت دارند که برای استفاده از داده‌های واقعی و خروجی آن‌ها برای بهبود مدل استفاده می‌شوند. مکانیزم‌هایی مانند A/B testing یا shadow development اجازۀ تست مدل‌های جدید با خطر ریسک کم را می‌دهند. مکانیزم A/B testing که در حقیقت برای مقایسۀ دو ورژن مختلف webpage استفاده می‌شود، برای مقایسۀ مدل‌ها در حین عملیاتی بودن نیز مفید است. با تغییراتی که به کمک داده‌های جدید روی مدل انجام می‌دهیم، می‌توان با تنظیم ترافیک بین دو مدل بررسی کرد که آیا تغییرات مفید بوده‌اند یا خیر و اگر بله، مدل جدید را جایگزین قبلی می‌کنیم.
3. امنیت و حریم خصوصی: امنیت و حریم خصوصی در هنگام تولید نهایی مدل به عنوان یک محصول اهمیت زیادی دارد. که یعنی باید داده‌های واقعی مشتری به صورت امن با حفظ حریم خصوصی پردازش شود. مثلا از طریق encryption and authentication این کار انجام شود و یا مقررات رگولاتوری GDPR و HIPAA رعایت شوند.

منابع:

<https://machinelearningmastery.com/optimizing-machine-learning-models-production-step-by-step-guide/>

<https://www.digitalocean.com/community/tutorials/model-quantization-large-language-models>

<https://medium.com/towards-data-science/embracing-automated-retraining-780ed49f9985>

<https://www.optimizely.com/optimization-glossary/ab-testing/>

**2.3. بخش دوم – پیاده‌سازی**

1**. در این بخش روش پیشنهادی خود را به صور عملی پیاده‌سازی می‌کنیم.**

**2. مقایسه با خروجی الگوریتم قبلی:**