

دانشگاه صنعتی شهرود

دانشکده مهندسی برق و کامپیوتر

شناسایی علائم راهنمایی رانندگی ترافیکی به
کمک شبکه عمیق **yolo**

پروژه کارشناسی مهندسی برق گرایش الکترونیک

مریم شیخ جعفری

استاد راهنما

دکتر علیرضا احمدی فرد

۱۴۰۳ شهریور

سَلَامٌ

تقدیم به:

پدر و مادر و خواهرم.

قدردانی

سپاس خداوندگار حکیم را که با لطف بی کران خود، آدمی را زیور عقل آراست.
در آغاز وظیفه خود می دانم از زحمات بی دریغ استاد راهنمای خود، جناب آقای دکتر احمدی فرد، صمیمانه
تشکر و قدردانی کنم که قطعاً بدون راهنمایی های ارزنده ایشان، این مجموعه به انجام نمی رسید.

مریم شیخ جعفری

۱۴۰۳ شهریور

چکیده

هدف از انجام این پروژه، طراحی و پیاده‌سازی سیستمی پیشرفته برای تشخیص علائم راهنمایی و رانندگی با استفاده از الگوریتم YOLO است. تشخیص دقیق و به موقع علائم راهنمایی و رانندگی، نقشی اساسی در توسعه سامانه‌های کمک راننده پیشرفته (ADAS) و خودروهای خودران ایفا می‌کند و می‌تواند به نحو مؤثری در کاهش تصادفات جاده‌ای و افزایش ایمنی سفرها سهمیم باشد. الگوریتم YOLO یکی از روش‌های نوین در حوزه شناسایی اشیاء است که به دلیل سرعت و دقت بالای خود شناخته شده است. در این الگوریتم، تصویر ورودی به چندین ناحیه تقسیم شده و هر ناحیه به طور همزمان برای شناسایی و طبقه‌بندی اشیاء مختلف مورد بررسی قرار می‌گیرد. این رویکرد، باعث افزایش کارایی و کاهش زمان پردازش می‌شود. در این پژوهش، ابتدا یک مجموعه‌داده جامع (Dataset) از تصاویر علائم راهنمایی و رانندگی گردآوری و پردازش گردیده است. سپس مدل YOLO با استفاده از این مجموعه‌داده آموزش داده شده است تا توانایی تشخیص و طبقه‌بندی علائم مختلف را در زمان واقعی کسب نماید.

واژگان کلیدی: Yolo, Dataset

فهرست مطالب

خ	فهرست تصاویر
ذ	فهرست جداول
ر	فهرست الگوریتم‌ها
ز	فهرست علائم اختصاری
۱	فصل ۱: مقدمه
۱	۱-۱ مقدمه
۱	۲-۱ تعریف مسئله
۲	۳-۱ اهمیت موضوع
۲	۴-۱ اهداف پژوهش
۲	۵-۱ ساختار گزارش پژوهش
۳	فصل ۲: آشنایی با تشخیص اشیا
۳	۱-۲ وظایف بینایی کامپیوترا
۵	۲-۲ نحوه کار تشخیص اشیا
۶	۱-۲-۲ ماتریس نتیجه
۸	فصل ۳: آشنایی با الگوریتم yolov5 و معماری yolov5
۸	۱-۳ مقدمه

فهرست مطالب

ج	
۸ معماری yolo ۲-۳
۹ Backbone ۱-۲-۳
۹ Neck ۲-۲-۳
۹ Head ۳-۲-۳
۱۰ مراحل کلی پردازش YOLO ۳-۳
۱۰ ۱-۳-۳ تغییر اندازه تصویر ورودی
۱۰ ۲-۳-۳ تقسیم تصویر به سلول ها(Grid-Cell-Division)
۱۱ ۳-۳-۳ پیش بینی جعبه های محدود کننده (Bounding Box Prediction)
۱۲ ۴-۳-۳ پیش بینی کلاس های اشیا (Class Prediction)
۱۲ ۵-۳-۳ اعمال Non-Maximum Suppression (NMS)
۱۳ ۴-۳ فرآیند های آموزش و اعتبار سنجی
۱۳ ۱-۴-۳ فرآیندهای آموزش YOLO (Training)
۱۵ ۲-۴-۳ فرآیندهای اعتبار سنجی yolo (Validation)
۱۶ yolov5 ۵-۳
۱۶ ۱-۵-۳ ویژگی های کلیدی YOLOv5
۱۷ ۲-۵-۳ معماری yolov5
۱۹ ۳-۵-۳ نسخه های مختلف yolov5

فصل ۴: مجموعه داده (Data set)

۲۰ مقدمه ۱-۴
۲۰ ۲-۴ فرآیند برچسب گذاری با استفاده از سایت Roboflow
۲۱ ۱-۲-۴ نحوه ساخت یک مجموعه داده
۲۵ ۳-۴ مجموعه داده traffic-signs
۲۵ ۱-۳-۴ تعداد داده ها

فصل ۵: آموزش و استفاده از مدل آموزش داده شده

۲۷ ۱-۵ راه اندازی
----	----------------------

فهرست مطالب

ح

۲۷	۱-۱-۵ دسترسی به فایل ها
۲۸	۲-۵ نصب پکیج ها و کتابخانه ها
۲۸	۱-۲-۵ نصب pip
۲۸	۲-۲-۵ نصب کتابخانه PyTorch
۲۸	۳-۲-۵ نصب Tensorboard و Tensorflow
۲۹	۳-۵ استفاده yolov5
۲۹	۱-۳-۵ انتقال فایل ها به google drive
۲۹	۲-۳-۵ دسترسی به فایل های yolov5
۲۹	۳-۳-۵ نصب کتابخانه های مورد نیاز yolov5
۳۰	۴-۵ فاز آموزش (training)
۳۰	۱-۴-۵ افزودن dataset
۳۰	۲-۴-۵ دستور training
۳۵	۵-۵ بررسی نمودارها
۴۸	۶-۵ استفاده از مدل آموزش داده شده

پیوست آ: *val.py* بررسی فایل

۵۰	آ-۱ کتابخانه ها
۵۰	آ-۲ بررسی بخش های مهم

پیوست ب: *train.py* بررسی فایل

۷۳	ب-۱ کتابخانه ها
۷۳	ب-۲ بررسی بخش های مهم

واژه‌نامه انگلیسی به فارسی

پیوست پ: مراجع

فهرست تصاویر

۴	۱-۲ آ- تشخیص اشیا ب- مکان یابی اشیا ج- طبقه بندی تصاویر
۵	۲-۲ جعبه محدود کننده
۷	۳-۲ مثال ماتریس نتیجه
۱۰	۱-۳ تقسیم بندی سلولی
۱۲	۲-۳ توضیح <i>iou</i>
۱۷	۳-۳ معماری yolov5
۱۹	۴-۳ مقایسه نسخه های مختلف yolov5
۱۹	۵-۳ مقایسه نسخه های مختلف yolov5
۲۱	۱-۴ <i>roboflow</i>
۲۱	۲-۴ <i>roboflow</i>
۲۲	۳-۴ <i>Augmentation</i>
۲۳	۴-۴ دیتا است <i>export</i>
۲۳	۵-۴ خروجی به صورت کد
۲۴	۶-۴ فایل دیتا است
۲۴	۷-۴ برچسب
۲۵	۸-۴ نمونه فایل <i>data.yaml</i>
۲۵	۹-۴ بخش بندی داده ها ۷۰٪ اموزش ۳۰٪ اعتبار سنجی ۱۰٪ تست
۳۲	۱-۵ آموزش
۳۵	۲-۵ ماتریس فراموشی

فهرست تصاویر

۳۶	(F1-Confidence) ۳-۵
۳۸	(Precision-Confidence) ۴-۵
۳۹	(Precision-Confidence) ۵-۵
۴۰	(Precision-Confidence) ۶-۵
۴۲	results ۷-۵
۴۵	labels ۸-۵
۴۷	labels-correlogram ۹-۵
۴۹	۱۰-۵ فرآیند تشخیص اشیا برای یک ویدیو

فهرست جداول

۲۶ ۱-۴ اسامی کلاس ها

فهرست الگوریتم‌ها

فهرست علائم اختصاری

a (m/s ²)	شتاب گرانش
F (N)	نیرو

فصل ۱

مقدمه

۱-۱ مقدمه

تشخیص اشیا^۱ یکی از مهم‌ترین و پرکاربردترین زمینه‌های هوش مصنوعی و یادگیری عمیق^۲ است که به شناسایی و طبقه‌بندی اشیا در تصاویر و ویدئوها می‌پردازد. این فناوری در حوزه‌های مختلفی از جمله خودروهای خودران کاربرد دارد. یکی از الگوریتم‌های برجسته و موثر در این زمینه، *yolo*^۳ است.

۱-۲ تعریف مسئله

یکی از عوامل کلیدی در بهبود اینمنی جاده‌ها، تشخیص دقیق و به موقع علائم راهنمایی و رانندگی است. تشخیص علائم راهنمایی و رانندگی توسط رانندگان در شرایط مختلف نوری و جوی می‌تواند چالش‌برانگیز باشد و مجرّب به نادیله گرفته شدن یا تفسیر اشتباه شود. در این راستا می‌توان از *بینایی کامپیوتری*^۴ کمک گرفت.

¹Object Detection

²Deep learning

³you only look once

⁴computer vision

۱-۳ اهمیت موضوع

با حل مشکل تفسیر اشتباه تابلو های راهنمایی رانندگان می توان کمک بسیاری در راستای کاهش تصادفات رانندگی انجام داد .

۱-۴ اهداف پروژه

هدف از انجام این پروژه، طراحی و پیاده سازی یک سیستم تشخیص علائم راهنمایی و رانندگی با استفاده از الگوریتم yolo ورژن ۵ است. این الگوریتم به دلیل رویکرد یکپارچه خود، قادر است با سرعت و دقت بالا اشیاء موجود در تصویر را شناسایی کند.

۱-۵ ساختار گزارش پروژه

این گزارش در ۵ فصل ارائه شده است :

● مقدمه

● آشنایی با تشخیص اشیا

● آشنایی با الگوریتم yolo و معماری yolov5

● مجموعه داده

● آموزش و استفاده از مدل آموزش داده شده

در ابتدا مقدمات آشنایی با تشخیص اشیا را فراهم شده است و سپس به معرفی الگوریتم yolo پرداخته شده است . پس از آشنایی با الگوریتم ، به دلیل استفاده ای نسخه ۵ ام آن مطالعه برای آشنایی با این نسخه نوشته شده است در ادامه در مورد نحوه ای ساخت یک مجموعه داده و همچنین مجموعه ای داده مورد استفاده ای این پروژه توضیحاتی بیان شده است که در نهایت در فصل آخر نحوه ای آموزش یک مدل و استفاده از آن عنوان شده است . همچنین این گزارش حاوی دو پیوست آ و ب می باشد که در آن ها بررسی فایل های train.py - val.py انجام شده است .

فصل ۲

آشنایی با تشخیص اشیا

۱-۲ وظایف بینایی کامپیوتری

وظایف بینایی کامپیوتری مجموعه‌ای از فعالیت‌ها و فرآیندهایی است که کامپیوتر برای تفسیر و درک تصاویر دیجیتال یا ویدیوهای انجام می‌دهد. برخی از وظایف معمول در بینایی کامپیوتر عبارتند از:

- *Object-Detection* تشخیص اشیا
- *Object-Localization* مکان یابی اشیا
- *Image-Classification* طبقه‌بندی تصاویر



شکل ۲-۱: آ- تشخیص اشیا ب- مکان یابی اشیا ج- طبقه بندی تصاویر

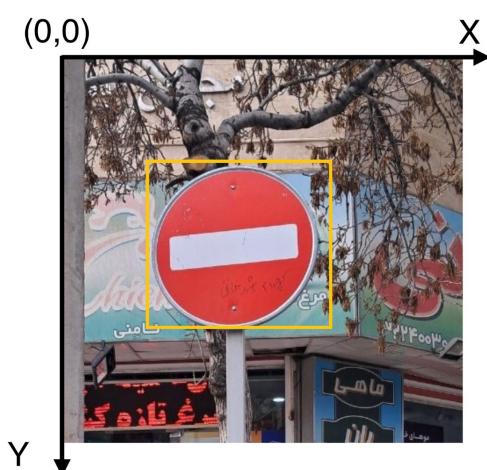
اگر هدف بررسی نوع تابلو باشد:

- تشخیص اشیا: خروجی پس از بررسی، تمامی تابلو های موجود و محل قرار. گیری آنها در تصویر را شناسایی می کند (گردش به راست ممنوع - مسیر یکطرفه).
- طبقه بندی تصاویر: خروجی پس از بررسی ، تنها مشخص می کند که تصویر دارای کدام تابلو است (تنها وجود گردش به راست ممنوع بررسی می شود).
- مکان یابی اشیا : خروجی پس از بررسی ، مشخص می کند که تصویر دارای کدام تابلو است و محل قرار گیری تابلو کجاست (تنها وجود گردش به راست ممنوع بررسی می شود).

با توجه به توضیحات می توان نتیجه گرفت برای تشخیص تابلو هایی راهنمایی رانندگی به دلیل وجود همزمان چند تابلو در تصویر بهتر است از تشخیص اشیا استفاده شود.

۲-۲ نحوه کار تشخیص اشیا

در تشخیص اشیا هر شی بطور یک کلاس تعریف شده و کلاس‌ها شماره‌گذاری می‌شوند همچنین همانطور که در بخش قبل اشاره شد، علاوه بر تشخیص نوع شی محل قرارگیری آن هم به وسیله یک **جعبه محدود کننده**^۱ مشخص می‌شود. یک نمونه از جعبه محدود کننده در تصویر ۲-۲ نشان داده شده.



شکل ۲-۲: جعبه محدود کننده

¹bounding box

۱-۲-۲ ماتریس نتیجه

نتیجه بررسی تشخیص اشیا در قالب یک ماتریس بیان می شود:

$$\begin{bmatrix} P \\ X_{min} \\ Y_{min} \\ X_{max} \\ Y_{max} \\ C_1 \\ C_2 \\ \vdots \\ \vdots \end{bmatrix}$$

توضیحات:

- p نشان دهنده وجود شی

در صورت وجود:

$$p = 1$$

در صورت عدم وجود:

$$p = 0$$

X_{max} و Y_{max} بیشترین مقدار x ، y -

برای مثال در تصویر ۲-۲ پایین گوشه‌ی سمت راست

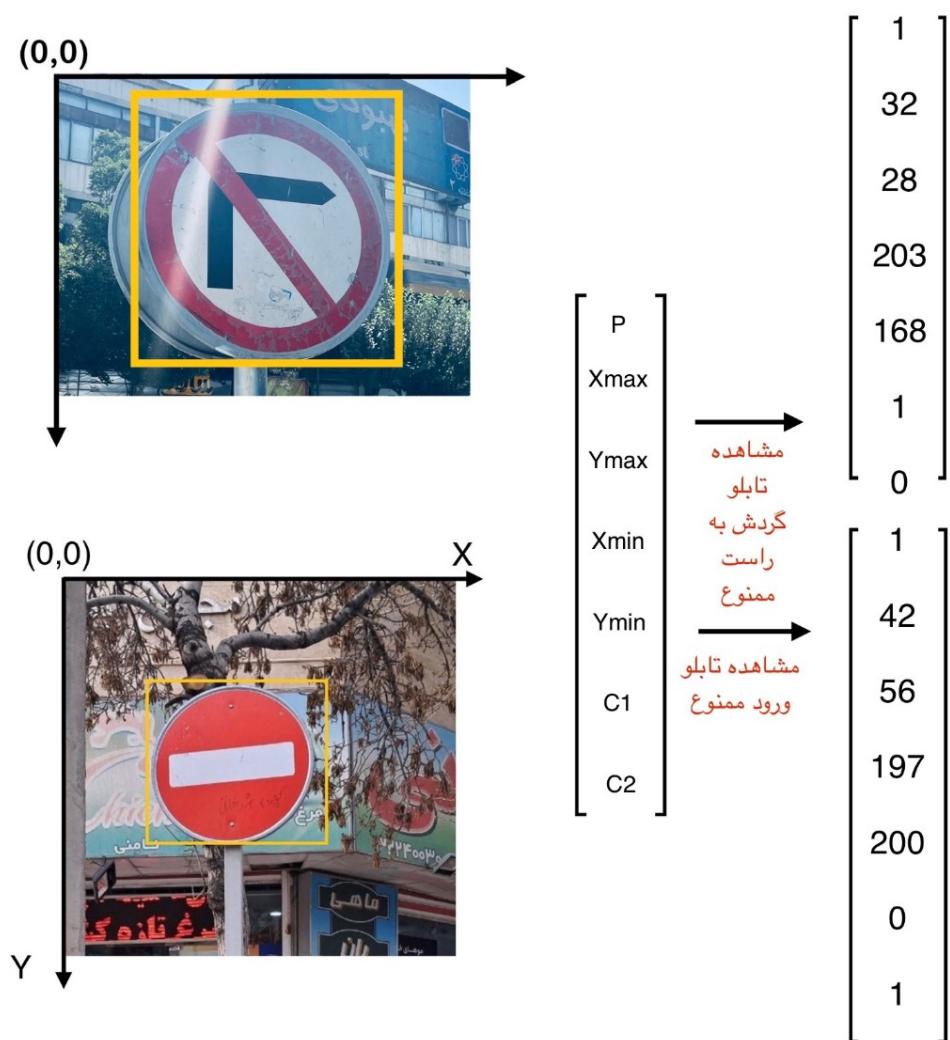
X_{min} و Y_{min} کمترین مقدار x ، y -

برای مثال در تصویر ۲-۲ بالا گوشه‌ی سمت چپ

C_1 و C_2 و ...

به تعداد کلاس های موجود متغیر C تعریف می شود ، در صورت وجود شی در تصویر متغیر C مربوط به آن برابر یک می شود.

در مثال زیر دو کلاس (گردش به راست ممنوع و ورود ممنوع) تعریف شده است که ماتریس نتیجه ای آن ها پس از بررسی به صورت زیر می شود :



شکل ۲-۳: مثال ماتریس نتیجه

در ادامه به آشنایی با الگوریتم *yolo* که یک الگوریتم تشخیص اشیا است می پردازیم .

فصل ۳

آشنایی با الگوریتم **yolo** و معماری **yolov5**

۱-۳ مقدمه

yolo مختصر شده عبارت **تو فقط یکبار نگاه می کنی^۱** یک الگوریتم تشخیص اشیا است که برخلاف روش های سنتی تشخیص اشیا، تنها در یک مرحله کل تصویر را با استفاده از شبکه های عصبی پیچشی (CNN) پردازش می کند. این موضوع باعث سرعت بالای این الگوریتم شده و آن را به یکی از پرطرفدار ترین ابزار های تشخیص اشیا تبدیل کرده است.

۲-۳ معماری **yolo**

معماری YOLO از سه بخش اصلی تشکیل شده است: **Head** و **Backbone**، **Neck** و هر یک از این بخش ها نقش کلیدی در فرآیند تشخیص اشیا ایفا می کنند

^۱you only look once

Backbone ۱-۲-۳

به عنوان بخش ابتدایی معماری YOLO ، وظیفه استخراج ویژگی های اولیه از تصویر ورودی را برعهده دارد. این بخش شامل یک شبکه عصبی کانولوشنی (CNN) است که لایه های کانولوشنی متعددی برای استخراج ویژگی ها از تصویر به کار می گیرد. در نسخه های مختلف YOLO ، از معماری های مختلفی به عنوان Backbone استفاده شده است. به طور مثال در yolo3 از Darknet استفاده شده است.

Neck ۲-۲-۳

بخشی از معماری YOLO است که وظیفه پردازش و بهبود ویژگی های استخراج شده توسط Backbone را دارد. Neck از ساختارهایی مانند ^۲ (FPN) و ^۳ (PAN) استفاده می کند. این شبکه ها اطلاعاتی را که در لایه های مختلف Backbone استخراج شده اند، ترکیب می کنند تا نمایشی چند مقیاسی از تصویر ایجاد شود. این فرآیند به مدل اجزه می دهد تا اشیای با اندازه های مختلف را با دقت بیشتری تشخیص دهد. به عبارت دیگر، Neck با ترکیب و تقویت ویژگی ها، امکان تشخیص اشیا در مقیاس های مختلف و بهبود دقت کلی مدل را فراهم می کند.

Head ۳-۲-۳

بخشنهایی معماری YOLO است که وظیفه تولید خروجی نهایی مدل را برعهده دارد. این بخش از شبکه عصبی کانولوشنی تشکیل شده که از ویژگی های ترکیب شده توسط Neck استفاده کرده و پیش بینی های نهایی را انجام می دهد. Head برای هر سلول از شبکه تقسیم بندی شده تصویر، چندین جعبه محدود کننده و مقادیر confidence (اطمینان از وجود شیء) به همراه کلاس های اشیا را پیش بینی می کند. پس از پیش بینی، از تکنیک ^۴ (NMS) برای فیلتر کردن جعبه های همپوشان و انتخاب بهترین جعبه ها استفاده می شود.

^۱ Network Pyramid Feature
^۲ Network Aggregation Path
^۳ Suppression Non-Maximum

۳-۳ مراحل کلی پردازش YOLO

۱-۳-۳ تغییر اندازه تصویر ورودی

تصاویر ورودی به اندازه‌های ثابت، معمولاً $416*416$ پیکسل تغییر اندازه داده می‌شوند تا با ورودی شبکه سازگار شوند.

۲-۳-۳ تقسیم تصویر به سلول‌ها (Grid-Cell-Division)

ابتدا، تصویر ورودی به یک شبکه منظم از سلول‌ها تقسیم می‌شود. این شبکه معمولاً دارای ابعاد $S*S$ است (مانند $7*7$). هر سلول مسئول پیش‌بینی اشیاء در ناحیه خاص خود را به عهده دارد. تا پردازش تصویر به‌طور موازی و سریع انجام شود.



شکل ۳-۱: تقسیم بندی سلولی

۳-۳-۳ پیش‌بینی جعبه‌های محدود کننده (Bounding Box Prediction)

مدل برای هر سلول در شبکه می‌تواند تعدادی جعبه محدود کننده را پیش‌بینی کند (مثلاً در YOLOv3، برای هر سلول ۳ جعبه محدود کننده را پیش‌بینی می‌کند). هر جعبه محدود کننده با چهار پارامتر تعريف می‌شود:

x و y : این مقادیر نشان‌دهنده مختصات مرکز جعبه محدود کننده نسبت به سلول مربوطه هستند. این مختصات به صورت نسبی و در محدوده $[0, 1]$ نمایش داده می‌شوند.

w و h : این مقادیر نشان‌دهنده عرض و ارتفاع جعبه محدود کننده هستند. عرض و ارتفاع نیز به صورت نسبی نسبت به ابعاد کل تصویر بیان می‌شوند.

همچنین، هر جعبه محدود کننده یک مقدار **مقدار اطمینان**^۵ دارد که نشان می‌دهد تا چه حد مدل مطمئن است که این جعبه حاوی یک شیء است. این مقدار از ۰ تا ۱ متغیر است و به صورت زیر تعريف می‌شود:

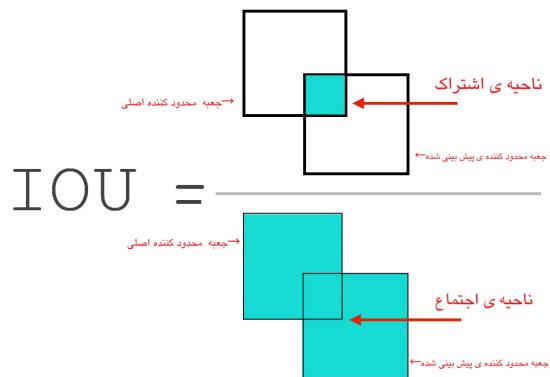
$$\text{Confidence} = P(\text{Object}) \times \text{IOU}$$

که در آن:

$P(\text{Object})$ احتمال وجود شیء در آن جعبه است.

IOU میزان همپوشانی جعبه پیش‌بینی شده با جعبه محدود کننده واقعی است.

⁵Confidence Score



شکل ۲-۳: توضیح iou

۴-۳-۳ پیش‌بینی کلاس‌های اشیا (Class Prediction)

علاوه بر مختصات جعبه محدود کننده و مقدار اطمینان، احتمال تعلق شیء درون جعبه به هر یک از کلاس‌های ممکن را نیز پیش‌بینی می‌شود. این پیش‌بینی به صورت یک بردار احتمال برای تمام کلاس‌ها انجام می‌شود. فرض کنید مدل قادر به تشخیص C کلاس مختلف باشد، در این صورت هر سلول C مقدار را به عنوان احتمال حضور شیء در هر کلاس خروجی می‌دهد. مقادیر احتمال کلاس‌ها همراه با مقدار اطمینان جعبه محدود کننده، برای تشخیص نهایی استفاده می‌شوند. نمره نهایی که برای هر جعبه مربوط به کلاس محاسبه می‌شود به صورت زیر است:

$$\text{Class Score} = P(\text{Object}) \times \text{IOU} \times P(\text{Class})$$

۵-۳-۳ اعمال Non-Maximum Suppression (NMS)

پس از اینکه جعبه‌های محدود کننده و کلاس‌ها پیش‌بینی شدند، ممکن است چندین جعبه برای یک شیء واحد پیش‌بینی شوند. برای رفع این مشکل، تکنیک (NMS) اعمال می‌شود. NMS جعبه‌هایی

فصل ۳. آشنایی با الگوریتم YOLO و معماری YOLOv5 آموزش و اعتبار سنجی

که بیشترین مقدار اطمینان را دارند، نگه داشته و بقیه را حذف می‌کند، به شرطی که همپوشانی (IOU) آن‌ها بیشتر از یک آستانه مشخص باشد.

۴-۳ فرایندهای آموزش و اعتبار سنجی

در واقع پردازش YOLO به طور کلی شامل دو مرحله اصلی است: آموزش^۶ و اعتبار سنجی^۷. هر یک از این مراحل شامل چندین فرایند کلیدی است که در ادامه به توضیح آن می‌پردازیم.

۱-۴-۳ فرآیندهای آموزش (Training)

۱- آماده سازی مجموعه داده ها^۸:

آموزش مدل شامل استفاده از مجموعه‌های از داده‌های برچسب دار است که مدل بر اساس آن‌ها یاد می‌گیرد تا روابط بین ورودی (تصاویر) و خروجی (جمعه‌های مجموعه و کلاس‌های اشیا) را شناسایی کند.

پیش‌پردازش داده‌ها: شامل تغییر اندازه تصاویر به ابعاد ثابت (مثلًاً ۴۱۶*۴۱۶)، نرمال‌سازی مقادیر پیکسل‌ها است.

۲- شبکه عصبی (Neural Network)

انتخاب معماری مناسب مانند yolov5m

۳- تنظیم‌های پارامترها

هایپرپارامترها مانند نرخ یادگیری^۹، اندازه دسته^{۱۰} و تعداد دوره‌های آموزش^{۱۱}

باید بدقت انتخاب شوند تا بهترین عملکرد مدل حاصل شود.

⁶Training

⁷Validation

⁸Data set

⁹learning rate

¹⁰batch size

¹¹epochs

فصل ۳. آشنایی با الگوریتم YOLO و معماری YOLOV5 ۴-۳. فرایندهای آموزش و اعتبار سنجی

نرخ یادگیری: مقدار تغییر وزن‌های شبکه در هر مرحله آموزش.

اندازه دسته: تعداد نمونه‌های داده که در هر مرحله آموزش به مدل ارائه می‌شود.

تعداد دوره‌های آموزش: تعداد دفعاتی که داده‌های آموزشی به مدل ارائه می‌شود.

۳- استفاده از توابع هزینه

yolo از [توابع هزینه](#)^{۱۲} برای بهینه‌سازی پیش‌بینی‌های خود استفاده می‌کند. تابع هزینه ترکیبی شامل خطای موقعیت جعبه‌های محدود کننده، خطای اندازه، خطای طبقه‌بندی، و خطای احتمال است.

- خطای موقعیت (Localization Loss): اختلاف بین موقعیت پیش‌بینی‌شده و موقعیت واقعی

جعبه‌های محدود کننده.

- خطای اندازه (Size Loss): اختلاف در اندازه و ابعاد جعبه‌های محدود کننده.

- خطای طبقه‌بندی (Classification Loss): اختلاف بین پیش‌بینی‌های کلاسی و برجسب‌های واقعی.

- خطای احتمال (Objectness Loss): احتمال وجود شیء در هر جعبه محدود کننده.

۴- فرایند یادگیری پس انتشار خطای:

شبکه با استفاده از پس انتشار خطای و بهینه‌سازی وزن‌ها، به یادگیری روابط بین ورودی و خروجی می‌پردازد.

تکرار آموزش :

مدل چندین بار بر روی مجموعه داده آموزش داده می‌شود تا به دقت مطلوب برسد

¹²Loss functions

فصل ۳. آشنایی با الگوریتم YOLO و معماری YOLOv5 ۴-۳. فرآیندهای آموزش و اعتبارسنجی

۲-۴-۳ فرآیندهای اعتبارسنجی (Validation) (yolo)

پس از هر دوره آموزش، مدل بر روی مجموعه داده‌های اعتبارسنجی ارزیابی می‌شود. مراحل این ارزیابی به صورت زیر است:

۱ - پیش‌بینی‌ها (Inference):

مدل آموزش دیده با استفاده از وزن‌های فعلی اش بر روی تصاویر مجموعه اعتبارسنجی اجرا می‌شود. برای هر تصویر، مدل پیش‌بینی‌هایی از اشیاء موجود در تصویر به همراه جعبه‌های محدود کننده و نمرات اعتماد^{۱۳} ارائه می‌دهد.

محاسبه‌ی معیارهای عملکرد:

توابع ضرر (Loss Functions):

در طول فرآیند اعتبارسنجی، مدل همان توابع ضرری که در آموزش استفاده می‌کند (مانند Box Loss، Classification Loss، Objectness Loss) را محاسبه می‌کند.

دقت (Precision) : نسبت پیش‌بینی‌های درست به کل پیش‌بینی‌ها.

بازخوانی (Recall) :

نسبت پیش‌بینی‌های درست به کل اشیاء موجود در تصاویر اعتبارسنجی.

: Mean Average Precision (mAP)

میانگین دقتهای محاسبه شده برای چندین آستانه‌ی مختلف از اعتماد.

این معیار برای ارزیابی کلی دقیق مدل استفاده می‌شود.

به روزرسانی پارامترها (در صورت لزوم):

براساس نتایج حاصل از اعتبارسنجی، پارامترهای مدل ممکن است تغییر یابند (برای مثال، تنظیمات آستانه‌ها یا سایر پارامترهای مرتبط با یادگیری). اگر مدل بیش از حد بر روی داده‌های آموزش تمرکز کرده باشد و دچار "Overfitting" شود، این مسئله در نتایج اعتبارسنجی نمایان خواهد شد

¹³confidence scores

yolov5 ۵-۳

در این بخش به آشنایی با yolov5 میپردازیم زیرا در اجرای این پروژه از این ورژن الگوریتم استفاده شده است.

1-۵-۳ ویژگی‌های کلیدی YOLOv5

پیاده‌سازی ساده:

YOLOv5 به طور کامل در PyTorch پیاده‌سازی شده است که باعث می‌شود پیاده‌سازی و استفاده از آن در پروژه‌های مختلف ساده‌تر باشد.

توسعه و بهینه‌سازی:

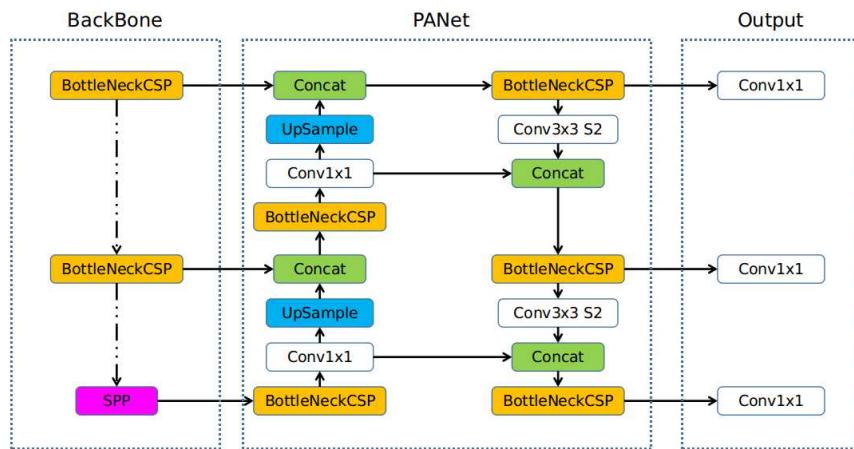
بهبودهایی در yolov5 شامل تکنیک‌های بهینه‌سازی و نرم‌السازی بهتر، استفاده از تکنیک‌های پیشرفته Augmentation، Mosaic Augmentation و همچنین تمرین‌های موثرتر مثل Auto Anchor برای بهبود دقت مدل انجام شده است.

سرعت و کارایی:

به دلیل بهینه‌سازی‌هایی که انجام شده، YOLOv5 یکی از سریع‌ترین مدل‌های تشخیص اشیاء است که می‌تواند روی سخت‌افزارهای متنوع اجرا شود.

۲-۵-۳ معماری yolov5

Overview of YOLOv5



شکل ۳-۳: معماری yolov5

تصویر بالا معماری yolov5 را نشان می‌دهد و به طور خلاصه می‌توان گفت که از سه بخش اصلی تشکیل شده است: *PANet*, *Backbone*, و *Output*. هر یک از این بخش‌ها وظایف خاصی در فرآیند تشخیص شیء دارند.

1-Backbone:

BottleNeckCSP:

این مأذول‌ها مسئول استخراج ویژگی‌ها از تصویر ورودی هستند. شبکه‌های عصبی کانولوشنی (CNN) به کار گرفته شده‌اند تا ویژگی‌های مهم تصویر را استخراج کنند.

YOLOv5 از مأذول‌های BottleNeckCSP استفاده می‌کند که قابلیت کاهش پیچیدگی محاسباتی را دارند و در عین حال ویژگی‌های حیاتی تصویر را حفظ می‌کنند.

SPP (Spatial Pyramid Pooling):

این مأژول که در انتهای بخش *Backbone* قرار دارد، به ترکیب و پردازش ویژگی‌ها با مقیاس‌های مختلف کمک می‌کند. این کار باعث می‌شود تا مدل بتواند ویژگی‌ها را در سطوح مختلف فضای خوبی ادغام کند.

2-PANet (Path Aggregation Network):

concat و Upsample:

در این بخش، از ویژگی‌های استخراج شده از *Backbone* استفاده می‌شود و با عملیات‌های ترکیب (*Concatenation*) و *Upsampling* ویژگی‌های مختلف با سطوح مختلف دقیق برای تقویت قابلیت تشخیص اشیاء آماده می‌شوند.

conv1x1 و Conv3x3 S2:

این لایه‌ها مسئول پردازش بیشتر ویژگی‌ها و ترکیب آن‌ها هستند. اندازه‌های مختلف کانولوشن به مدل کمک می‌کنند تا ویژگی‌های محلی (مانند لبه‌ها و بافت‌ها) را با ویژگی‌های کلی‌تر (مانند اشکال و ساختارهای بزرگتر) ترکیب کند.

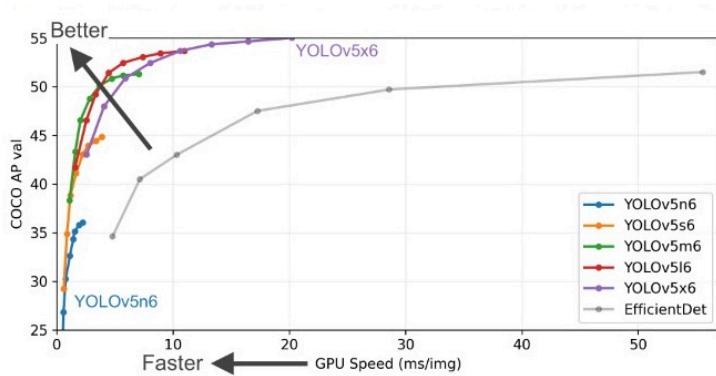
3-OutPut:

Conv1x1:

این لایه‌ها مسئول تولید خروجی نهایی هستند. در نهایت، مدل پیش‌بینی‌های خود را در قالب جعبه‌های محدود کننده، احتمال‌ها و برچسب‌های کلاس‌ها ارائه می‌دهد.

۳-۵-۳ نسخه های مختلف yolov5

yolov5 در نسخه های مختلفی ارائه شده است که عملکرد این نمونه ها یکسان است اما در ویژگی هایی مانند سرعت و دقت با یک دیگر متفاوت هستند. می توان با درنظر گرفتن الگوریتم ها در انجام یک پروژه نسخه مناسب را انتخاب کرد.



شکل ۳-۴: مقایسه نسخه های مختلف yolov5

Model	size (pixels)	mAP ^{val} 50-95	mAP ^{val} 50	Speed CPU b1 (ms)	Speed V100 b1 (ms)	Speed V100 b32 (ms)	params (M)	FLOPs @640 (B)
YOLOv5n	640	28.0	45.7	45	6.3	0.6	1.9	4.5
YOLOv5s	640	37.4	56.8	98	6.4	0.9	7.2	16.5
YOLOv5m	640	45.4	64.1	224	8.2	1.7	21.2	49.0
YOLOv5l	640	49.0	67.3	430	10.1	2.7	46.5	109.1
YOLOv5x	640	50.7	68.9	766	12.1	4.8	86.7	205.7
YOLOv5n6	1280	36.0	54.4	153	8.1	2.1	3.2	4.6
YOLOv5s6	1280	44.8	63.7	385	8.2	3.6	12.6	16.8
YOLOv5m6	1280	51.3	69.3	887	11.1	6.8	35.7	50.0
YOLOv5l6	1280	53.7	71.3	1784	15.8	10.5	76.8	111.4
YOLOv5x6	1280	55.0	72.7	3136	26.2	19.4	140.7	209.8

شکل ۳-۵: مقایسه نسخه های مختلف yolov5

برای انتخاب نسخه می مناسب می توان از تصاویر ۳-۴ و ۳-۵ کمک گرفت برای مثال در انجام این پروژه از yolov5m استفاده شده که از سرعت و دقت مناسبی برخوردار است.

فصل ۴

مجموعه داده (Data set)

۱-۴ مقدمه

همانطور که در فصل قبل توضیح داده شد در الگوریتم yolo برای فرآیندهای آموزش و اعتبار سنجی نیاز به یک مجموعه داده برچسب دار است در این فصل به توصیف مجموعه داده ها و برچسب گذاری آن ها می پردازیم.

۲-۴ فرآیند برچسب گذاری با استفاده از سایت Roboflow

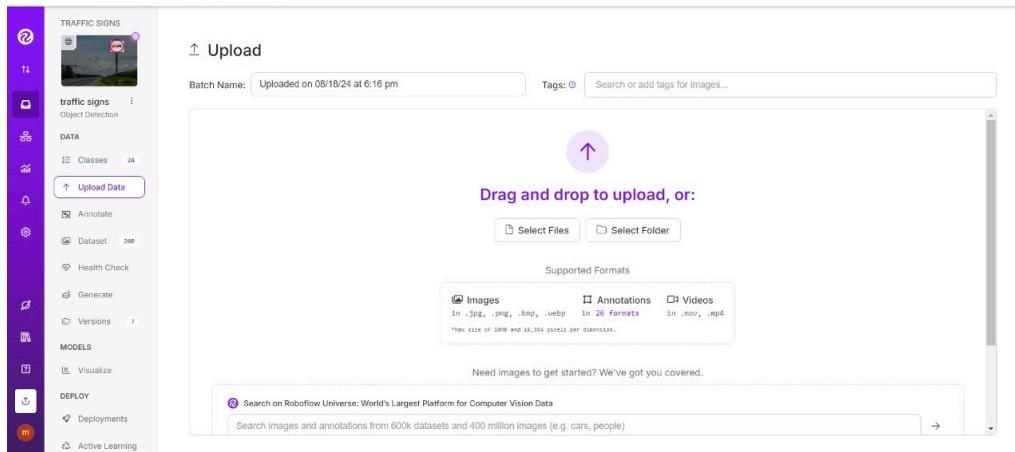
برای فرآیند برچسب گذاری اپلیکیشن ها و سایت های مختلفی وجود دارد. برای انجام این پروژه از سایت [roboflow](https://roboflow.com/)^۱ استفاده شده است. یکی از مزایا مهم فرآیند برچسب گذاری با استفاده از roboflow این است که میتوان به راحتی مجموعه داده را ارتقا داد.

<https://roboflow.com/>^۱

فصل ۴. مجموعه داده (DATA SET) ۲-۴ . فرآیند برچسب گذاری با استفاده از سایت ROBOFLOW

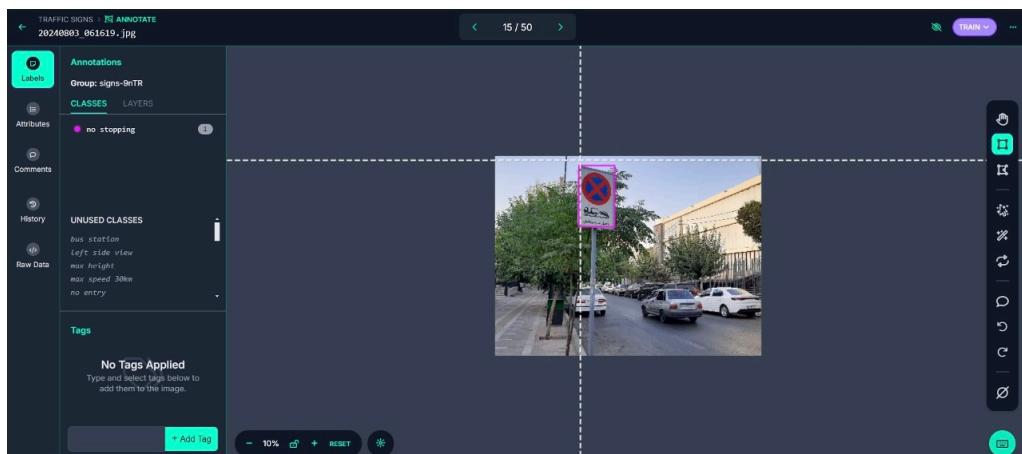
۱-۲-۴ نحوه ساخت یک مجموعه داده

پس از ساخت حساب کاربری در سایت *roboflow* یک پروژه جدید ایجاد میکنیم.



شکل ۱-۴

با انتخاب گزینه **Upload** میتوانیم عکس هایی که برای برچسب گذاری اماده کرده ایم را در سایت آپلود کنیم. آپلود دیتاها به صورت همزمان قابل انجام است و نیازی به افزودن آنها به صورت تکی نیست.



شکل ۲-۴

پس از آپلود تمامی عکس ها با انتخاب هر عکس می توانیم برچسب گذاری آن را شروع کنیم. برای مثال در تصویر ۲-۴ با یک جعبه محدود کننده شی مورد نظرمان که یک تابلو توقف ممنوع است

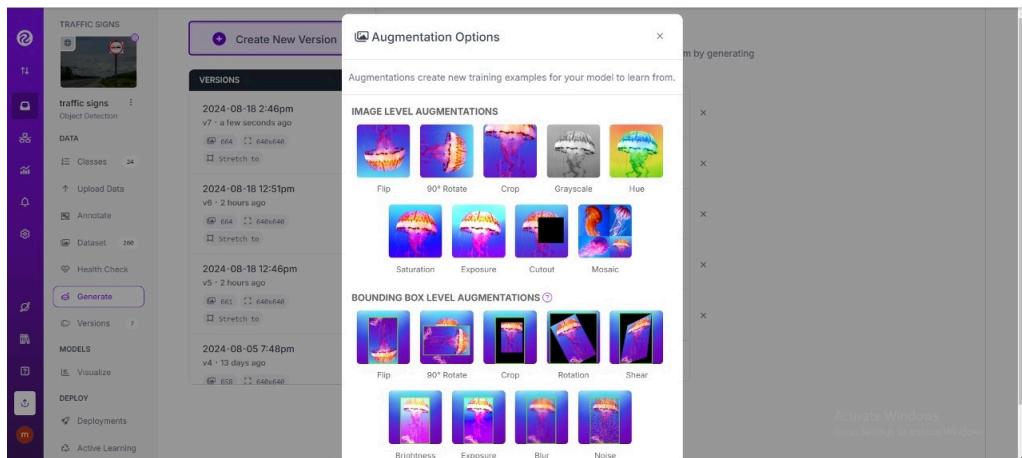
فصل ۴. مجموعه داده (DATA SET) ۲-۴ . فرآیند برچسب گذاری با استفاده از سایت ROBOFLOW

انتخاب کرده و برای آن یک کلاس تعریف میکنیم، اگر قبلاً کلاسی با این نام ساخته شده باشد نیازی به تعریف دوباره کلاس نیست تنها کافی است از بین لیست کلاس ها آن را انتخاب نماییم به همین ترتیب برای تمامی عکس ها این مرحله را تکرار میکنیم.

پس از اتمام این مرحله وارد با ورود به قسمت *generate* میتوانیم از امکانت زیر استفاده کنیم :

۱ - در قسمت *preprocessing* با انتخاب گزینه *resize* اندازه تمامی عکس ها را تغییر داده و یک اندازه ی کلی برای همه آنها در نظر بگیریم.

۲ - در قسمت *Augmentation* میتوانیم با اضافه کردن ویژگی هایی مانند افزایش نور یا افزودن نویز باعث افزایش تعداد داده ها شویم ، بالا بردن تعداد داده ها به بهبود عملکرد مدل کمک می کند.



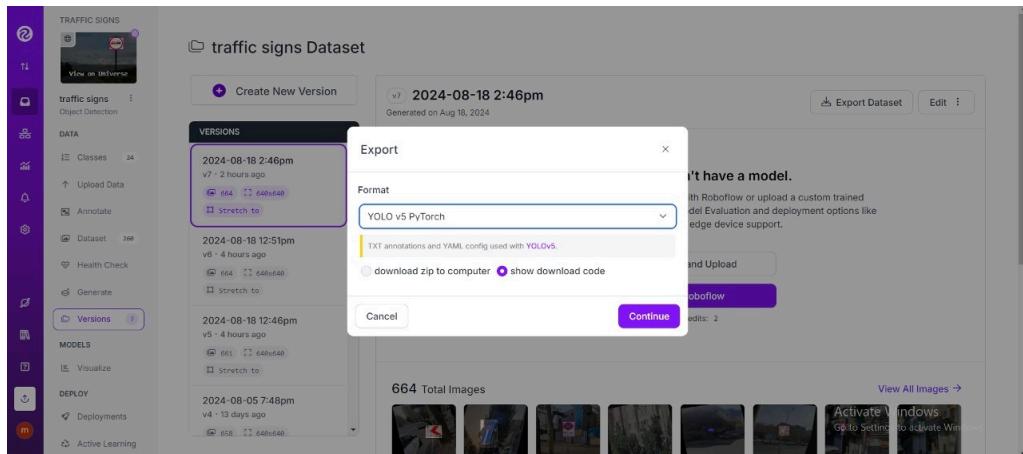
شکل ۴-۳: Augmentation

در تصویر ۴-۳ می توانیم برخی از موارد *Augmentation* مشاهده کنیم.

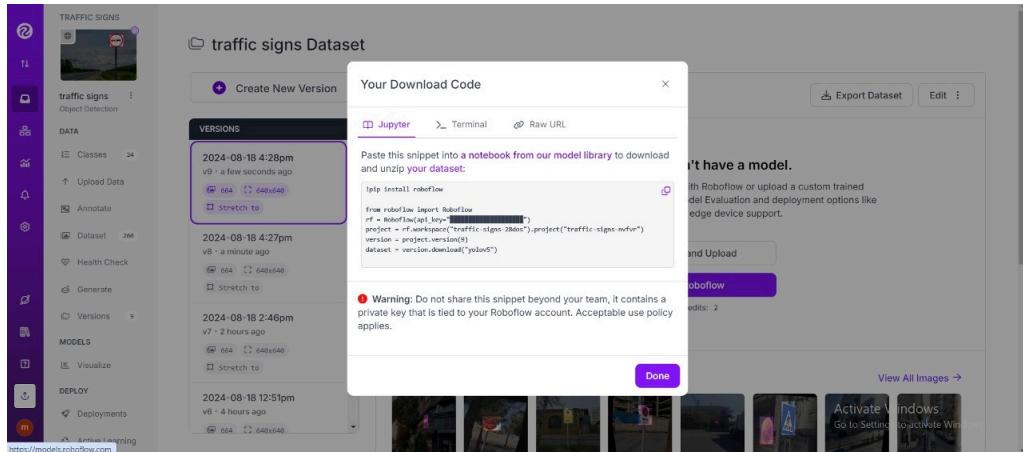
پس از انتخاب این گزینه ها باید انتخاب کنیم چند درصد از داده ها برای آموزش و چند درصد برای اعتبار سنجی استفاده شود . که بهترین حالت ۷۰٪ برای آموزش ۳۰٪ برای اعتبار سنجی و ۱۰٪ برای تست است .

حال در قسمت *versions* میتوانیم معماری مدل نظر خود را انتخاب کرده و با استفاده از گزینه *Export* مجموعه داده ای متناسب با آن معماری را دریافت کنیم . انجام این پروژه با استفاده از الگوریتم yolov5 بوده است پس گزینه *yolov5 pytorch* را انتخاب میکنیم .

فصل ۴. مجموعه داده (DATA SET) ۲-۴ . فرآیند برچسب گذاری با استفاده از سایت ROBOFLOW



شکل ۴-۴: دیتا ست export



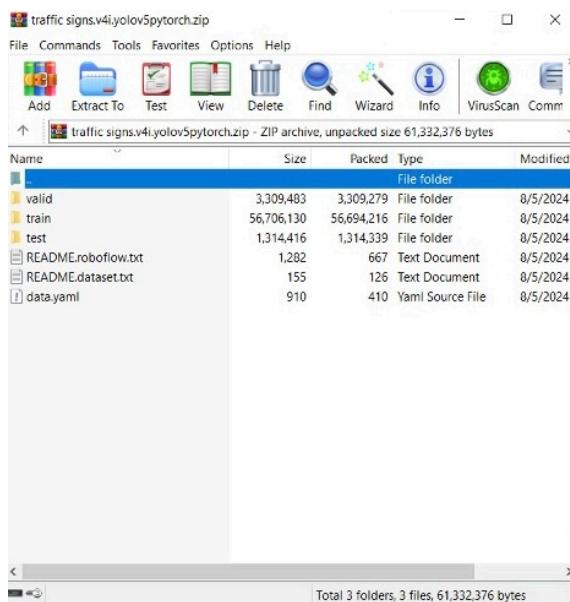
شکل ۴-۵: خروجی به صورت کد

خروجی دیتا ست ساخته شده توسط roboflow هم به صورت یک فایل زیپ و هم به صورت کد قابل دسترسی است.

فصل ۴. مجموعه داده (DATA SET) ۲-۴ . فرآیند برچسب گذاری با استفاده از سایت ROBOFLOW

بررسی فایل زیپ:

فایل زیپ حاوی سه فولدر *train* و *valid* و *test* است که هر پوشه بنا به تقسیم بندی انجام شده مقداری از داده ها را شامل می شود .



شکل ۴-۶: فایل دیتا ست

برای مثال در پوشه *train* ۷۰٪ عکس ها به همراه برچسب هایشان موجود است . برچسب هر عکس به صورت یک فایل *.txt* است که شامل شماره کلاس و مختصات شی نسب به جعبه محدود کننده است.



شکل ۷-۴: برچسب

فایل : *data.yaml*

فایل پیکربندی داده ها است . این فایل شامل اطلاعاتی مانند تعداد کلاس ها ، اسمی کلاس ها و مسیر فایل های (train , test , val) می باشد .

```
C:\Users\...\AppData\local\Temp\RoboFlow\1507636246> | dckyaml
1   train: ..\train\images
2   validation: ..\val\images
3   test: ..\test\images
4
5   nct: 33
6   names: ['bus station', 'left side view', 'max height', 'max speed', 'no entry', 'no left turn', 'no overtaking', 'no right turn', 'no stopping', 'no through road', 'red light', 'stop sign', 'yield sign']
7   roboflow:
8   workspace: traffic-signs-z8d8os
9   project: traffic-signs-nvfr
10  version: 12
11  license: CC BY 4.0
12  url: https://universe.roboflow.com/traffic-signs-z8d8os/traffic-signs-nvfr/dataset/12
```

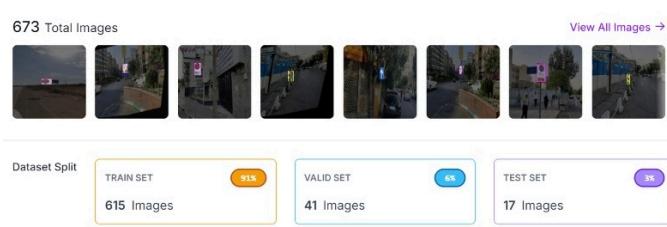
شکل ۴-۸: نمونه فایل *data.yaml*

۳-۴ مجموعه داده traffic-signs

۱-۳-۴ تعداد داده ها

برای انجام این پروژه مجموعه داده از تابلو های راهنمایی رانندگی شهری جمع آوری شده است . تعداد عکس های جمع آوری شده ۲۶۷ عدد است که پس از استفاده از گزینه های noise rotation و shear,blur,brightness به ۶۸۵ عدد افزایش یافته است .

همانطور که در بخش قبل گفته شد داده ها به صورت زیر بخش بندی شدند :



شکل ۴-۹: بخش بندی داده ها ۰.۱٪ انتبار سنجی ۳۰٪ اموزش ۷۰٪ تست

این تصاویر مربوط به ۲۱ کلاس هستند که به هر تابلو یک کلاس اختصاص داده شده است . اسامی کلاس ها در جدول ۴-۱ قابل مشاهده است .

class name	نام کلاس	شماره
bus station	ایستگاه اتوبوس	۱
no entry	ورود ممنوع	۲
no stopping	توقف ممنوع	۳
no right turn	گردنش به راست ممنوع	۴
no left turn	گردنش به چپ ممنوع	۵
stop	ایست	۶
no through road	بن بست	۷
max speed	حداکثر سرعت	۸
max height	حداکثر ارتفاع	۹
parking for disabled	پارکینگ معلولین	۱۰
parking lot	پارکینگ	۱۱
school crossing	محل عبور دانش آموزان	۱۲
left side view	حاشیه نما به چپ	۱۳
one way traffic	مسیر یک طرفه	۱۴
uneven road	دست انداز	۱۵
road bump ahead	برآمدگی	۱۶
p crossing ahead	گذرگاه عابر پیاده	۱۷
single left side view	حاشیه و جهت نمای چپ - تکی	۱۸
single right side view	حاشیه و جهت نمای راست - تکی	۱۹
round about	جهت عبور در میدان	۲۰
no overtaking	سبقت ممنوع	۲۱

جدول ۴-۱ : اسامی کلاس ها

در ادامه فرایند training با استفاده از این مجموعه داده ، توضیح داده می شود .

فصل ۵

آموزش و استفاده از مدل آموزش داده شده

۱-۵ راه اندازی

برای عملیات *training* نیازمند استفاده از *GPU* هستیم به همین منظور از *google colab*^۱ که *GPU* را به صورت رایگان در اختیارمان قرار می دهد استفاده شده است . موضوع مهم و قابل توجه این است که محیط *google colab* لینوکس است .

۱-۱-۵ دسترسی به فایل ها

برای آنکه به فایل های مورد نظرمان در *google colab* دسترسی پیدا کنیم میتوانیم فایل ها را ابتدا وارد کرده و سپس *google colab* را به *google drive* متصل کنیم . دستور زیر برای اتصال یا به اصطلاح *mount* کردن *google drive* به *google colab* است.

```
from google.colab import drive
drive.mount('/content/gdrive')
```

google colab^۱

۲-۵ نصب پکیج ها و کتابخانه ها

۱-۲-۵ نصب pip

pip سیستم نصب و مدیریت کتابخانه ها در پایتون است در اینجا با استفاده از این دستور آن را نصب و به روز رسانی میکنیم :

```
! python -m pip install --upgrade pip
```

۲-۲-۵ نصب کتابخانه PyTorch

PyTorch کتابخانه ای برای توسعه و پیاده سازی مدل های یادگیری عمیق به ویژه پردازش تصویر است و نصب آن به صورت زیر است :

```
! pip install torch
```

۳-۲-۵ نصب Tensorboard و Tensorflow

این امکان را می دهد تا مدل های پیچیده ای از جمله شبکه های عصبی عمیق را طراحی و پیاده سازی کنیم و همچنین پشتیبانی از پردازش های توزیع شده و بهینه سازی برای اجرا بر روی GPU را فراهم می کند . نصب آن به صورت زیر است :

```
!pip install tensorflow
```

TensorBoard به تجزیه و تحلیل روندهای آموزشی کمک کرده و امکان شناسایی مشکلات و بهینه سازی مدل ها را فراهم می کند و نصب آن به صورت زیر است :

```
!pip install tensorboard
```

۳-۵ استفاده yolov5

۱-۳-۵ انتقال فایل ها به google drive

این دستور یک کبی کامل از فایل های YOLOv5 در google drive ایجاد می کند:

```
! git clone https://github.com/ultralytics/yolov5
```

۲-۳-۵ دسترسی به فایل های yolov5

این دستور ما را به دایرکتوری yolov5 می برد، که در آن می توانیم به فایل ها و اسکریپت های مربوط ب دسترسی داشته باشیم:

```
%cd yolov5
```

۳-۳-۵ نصب کتابخانه های مورد نیاز yolov5

برای استفاده از yolov5 نیاز به نصب کتابخانه های مختلف با نسخه های مشخص داریم که این کار به صورت تک به تک زمان بر است، تمامی این کتابخانه ها در فایل requirements.txt وجود دارد با نصب آن تمامی پکیج های مورد نیاز را نصب می کنیم:

```
!pip install -r requirements.txt
```

۴-۵ فاز آموزش (training)

۱-۴-۵ افزودن dataset

همانطور که گفته شد مجموعه داده در سایت Roboflow هم به صورت فایل زیپ و هم به صورت Jupyter کد قابل دسترسی است . به دلیل آن که در محیط google cloab کار میکنیم میتوانیم از کد استفاده کنیم ، کد زیر توسط Roboflow ارائه شده است .

```
!pip install roboflow  
from roboflow import Roboflow  
  
rf rf = Roboflow(api-key="8B0OjllU8KYFybV3eVZe")  
  
project = rf.workspace("traffic-signs-28dos").project("traffic-signs-nvfvr")  
  
version = project.version(13)  
  
dataset = version.download("yolov5")
```

۲-۴-۵ دستور traning

برای train کردن از دستور زیر استفاده می شود :

```
!python train.py - -img 416 - -batch 16 - -epochs 30 - -data  
/content/yolov5/traffic-signs-4/data.yaml - -weights yolov5m.pt - -cach
```

توضیحات :

!python train.py:

این بخش از دستور نشان می دهد که فایل train.py باید با استفاده از Python اجرا شود.

فصل ۵. آموزش و استفاده از مدل آموزش داده شده

(TRAINING) ۴-۵. فاز آموزش

- -img 416:

این بخش اندازه تصویر ورودی را تعیین می‌کند. در اینجا، اندازه تصاویر 416×416 پیکسل تعیین شده است.

- -batch 16:

این بخش [اندازه‌ی دسته^۳](#) را تنظیم می‌کند. منظور از اندازه دسته تعداد داده‌هایی است که به صورت همزمان در هر مرحله پردازش می‌شوند. در اینجا، مدل در هر مرحله از آموزش، ۱۶ تصویر را به صورت همزمان پردازش می‌کند.

- -epochs 30:

این بخش [تعداد دوره‌های آموزش^۴](#) را تعیین می‌کند. در اینجا، مدل برای ۳۰ دوره آموزش داده می‌شود.

-data /content/yolov5/traffic-signs-4/data.yaml

این بخش به فایل پیکربندی داده‌ها (data.yaml) اشاره دارد که مسیر داده‌ها، کلاس‌ها و تنظیمات دیگر را مشخص می‌کند.

-weights yolov5m.pt

³Batch size

⁴Epoches

پس از اجرای این دستور خروجی زیر قابل مشاهده است:

```

train: New cache created: /content/yolov5/traffic-signs-4/train/labels.cache
train: Caching images (0.3GB ram): 100% 600/600 [00:02<00:00, 298.07it/s]
... val: Scanning /content/yolov5/traffic-signs-4/valid/labels... 41 images, 0 backgrounds, 0 corrupt: 100% 41/41 [00:00<00:00, 427.35it/s]
val: New cache created: /content/yolov5/traffic-signs-4/valid/labels.cache
val: Caching images (0.0GB ram): 100% 41/41 [00:00<00:00, 111.43it/s]

AutoAnchor: 5.84 anchors/target, 1.000 Best Possible Recall (BPR). Current anchors are a good fit to dataset ✓
Plotting labels to runs/train/exp/labels.jpg...
Image sizes 416 train, 416 val
Using 2 dataloader workers
Logging results to runs/train/exp
Starting training for 30 epochs...

Epoch GPU_mem box_loss obj_loss cls_loss Instances Size
0/29 2.86G 0.09941 0.01894 0.08874 8 416: 100% 38/38 [00:13<00:00, 2.84it/s]
Class Images Instances P R mAP50 mAP50-95: 100% 2/2 [00:02<00:00, 1.42s/it]
all 41 43 0.00325 0.319 0.00785 0.00268

Epoch GPU_mem box_loss obj_loss cls_loss Instances Size
1/29 3.46 0.07159 0.01988 0.07999 14 416: 100% 38/38 [00:07<00:00, 4.93it/s]
Class Images Instances P R mAP50 mAP50-95: 100% 2/2 [00:00<00:00, 5.31it/s]
all 41 43 0.0057 0.778 0.0908 0.033

Epoch GPU_mem box_loss obj_loss cls_loss Instances Size
2/29 3.4G 0.06636 0.01745 0.07581 6 416: 100% 38/38 [00:09<00:00, 4.02it/s]
Class Images Instances P R mAP50 mAP50-95: 100% 2/2 [00:00<00:00, 5.04it/s]
all 41 43 0.865 0.0417 0.138 0.0569

Epoch GPU_mem box_loss obj_loss cls_loss Instances Size
3/29 3.4G 0.06494 0.01551 0.07205 26 416: 18% 7/38 [00:01<00:06, 4.92it/s] private Windows

```

شکل ۱-۵: آموزش

Epoch:

این ستون نشان می‌دهد که مدل در کدام دوره از آموزش قرار دارد.

GPU-mem:

این ستون میزان حافظه *GPU* استفاده شده در طول آموزش را نشان می‌دهد. واحد آن گیگابایت (GB) است. مقدار حافظه‌ای که مدل برای پردازش نیاز دارد بسته به پیچیدگی مدل و اندازه داده‌ها متفاوت است.

box-loss:

این مقدار نشان‌دهنده میزان خطای مربوط به مختصات جعبه‌های محدود کننده است که مدل برای شناسایی اشیاء استفاده می‌کند. هرچه این مقدار کمتر باشد، مدل بهتر توانسته است جعبه‌های دقیق‌تری را پیش‌بینی کند.

obj-loss:

این ستون نشان دهنده میزان خطای مربوط به تشخیص وجود یا عدم وجود یک شیء در تصویر است.
این خطا مربوط به احتمال حضور یک شیء در هر جعبه است.

cls-loss:

این ستون نشان دهنده خطای طبقه بندی است که مدل در تشخیص کلاس درست شیء مرتکب می شود.
این خطا در صورتی افزایش می یابد که مدل نتواند کلاس صحیح اشیاء را تشخیص دهد.

Instances:

این ستون تعداد اشیائی را نشان می دهد که در طول هر دوره پردازش شده اند. این عدد نشان دهنده تعداد نمونه هایی است که مدل در آن دوره روی آن ها تمرین کرده است.

Size:

این ستون اندازه تصاویر ورودی (Input image size) به مدل را نشان می دهد. در این تصویر، سایز ورودی 416^*416 پیکسل است.

Class (P, R, mAP50, mAP50-95):

این بخش معیارهای ارزیابی مدل را برای عملکرد روی داده های آموزش و اعتبارسنجی نشان می دهد:

P (Precision):

دقت پیش بینی های مدل را نشان می دهد. به طور خلاصه، درصد مواردی که مدل به درستی تشخیص داده است که یک شیء وجود دارد.

R (Recall):

فراخوانی، نشان می دهد چند درصد از اشیاء واقعی موجود در تصویر توسط مدل شناسایی شده اند.

mAP50:

میانگین دقت در سطح (IoU). ۵۰٪ برای تمامی کلاس‌ها. این معیار نشان می‌دهد که مدل تا چه حد خوب اشیاء را شناسایی کرده است.

mAP50-95:

میانگین دقت در سطوح مختلف IoU از ۹۵٪ تا ۹۰٪. این معیار دقیق‌تر است و توانایی مدل در تشخیص دقیق‌تر اشیاء را نشان می‌دهد.

هرچه مقادیر mAP50 و mAP50-95 بالاتر باشند، به این معناست که مدل به خوبی آموزش دیده و عملکرد بهتری دارد. همچنین، کاهش مقدار loss‌ها نشانه‌ای از بهبود مدل در طول دوره‌های مختلف آموزش است.

پس از پایان فرآیند آموزش نتیجه آن در پوشه weights در دو فایل last.pt و best.pt ذخیره می‌شود. که برای استفاده از مدل از best.pt استفاده می‌کنیم.

`%load-ext tensorboard`

این دستور یک magic command است که افزونه (extension) tensorboard را بارگذاری می‌کند. این افزونه به شما امکان می‌دهد تا از TensorBoard به‌طور مستقیم در محیط google colab استفاده کنید.

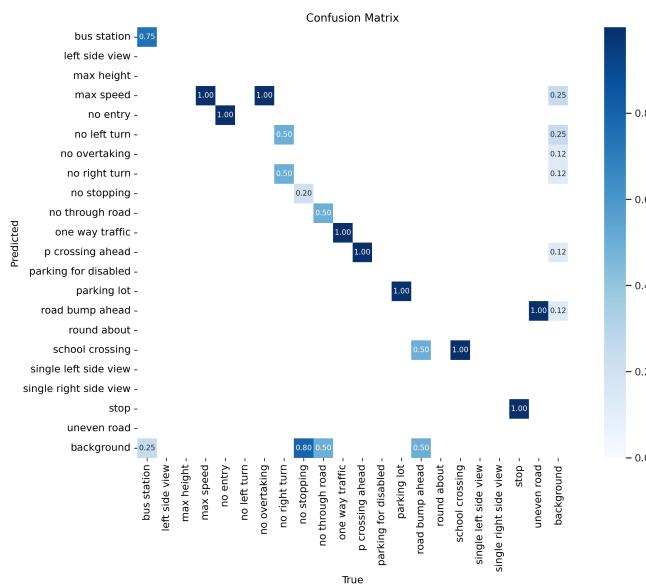
`%tensorboard --logdir runs`

این دستور مشخص می‌کند که TensorBoard باید دایرکتوری runs را که شامل لاغ‌هاست برای نمایش گراف‌ها و نمودارها بررسی کند. این لاغ‌ها معمولاً توسط TensorFlow یا PyTorch در طول آموزش مدل‌ها تولید می‌شوند.

نمودار‌های خروجی این قسمت در قسمت yolov5- runs - train- exp ذخیره می‌شود که در ادامه به بررسی آن‌ها می‌پردازیم.

۵-۵ بررسی نمودارها:

۱ - ماتریس سردرگمی (Confusion Matrix):



شکل ۲-۵: ماتریس فراموشی

: محور افقی (True)

برچسب‌های واقعی داده‌های تست هستند. این محور نشان می‌دهد که واقعاً چه کلاسی باید پیش‌بینی می‌شد.

: محور عمودی (Predicted)

برچسب‌های پیش‌بینی شده توسط مدل هستند. این محور نشان می‌دهد که مدل چه کلاسی را پیش‌بینی کرده است.

تحلیل نقاط موجود در ماتریس:

قطر اصلی (از بالا چپ به پایین راست):

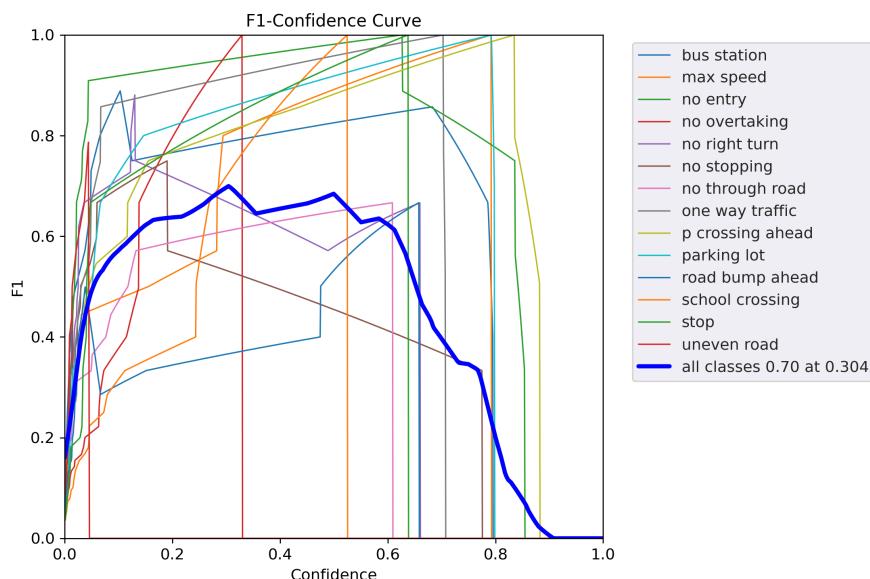
این خط نشان‌دهنده پیش‌بینی‌های صحیح است. یعنی هر نقطه روی این خط نشان می‌دهد که مدل کلاس واقعی را به درستی تشخیص داده است.

نقاط خارج از قطر اصلی:

نقاط خارج از این خط نشان دهنده پیش‌بینی‌های نادرست مدل هستند. این موارد نشان می‌دهند که مدل یک کلاس را به اشتباه به عنوان کلاس دیگری پیش‌بینی کرده است. هرچه این نقاط پرنگ‌تر باشند، یعنی تعداد بیشتری از نمونه‌ها به اشتباه طبقه‌بندی شده‌اند.

در این مدل نقاط پرنگ خارج از قطر اصلی نسبت به نقاط پرنگ روی قطر اصلی کمتر است که نشان دهنده‌ی عملکرد مطلوب مدل است.

۲ - منحنی (F1-Confidence):



شکل ۳-۵: (F1-Confidence)

ساختار منحنی F1-Confidence :

محور افقی (Confidence):

این محور نشان‌دهنده سطح اعتماد (confidence threshold) است. Confidence به احتمال مدل برای پیش‌بینی یک کلاس خاص اشاره دارد. هرچه این مقدار بالاتر باشد، مدل مطمئن‌تر است که پیش‌بینی‌اش صحیح است.

محور عمودی (F1):

این محور مقدار نمره F1 را نشان می‌دهد. نمره F1 میانگینی از دقت (Precision) و حساسیت (Recall) است و به ویژه زمانی مفید است که توزیع کلاس‌ها نامتوابن باشد. نمره F1 بالا نشان‌دهنده تعادلی بین دقت و فراخوانی است.

منحنی‌های رنگی:

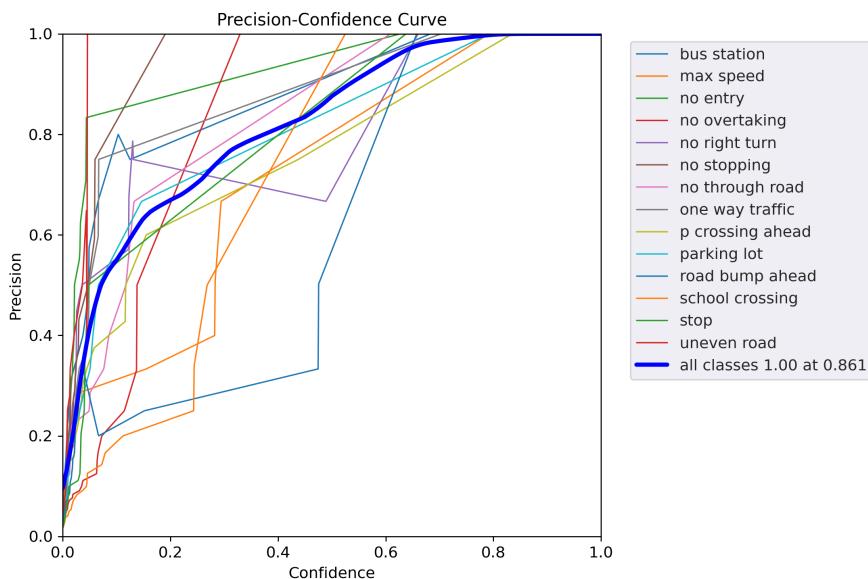
هر منحنی رنگی در نمودار نشان‌دهنده کلاس یکی از علائم ترافیکی است. این منحنی‌ها نشان می‌دهند که نمره F1 برای هر کلاس خاص با تغییر آستانه confidence چگونه تغییر می‌کند.

به عنوان مثال، ممکن است مشاهده کنید که برای برخی از کلاس‌ها نمره F1 با افزایش confidence به شدت کاهش می‌یابد. این نشان می‌دهد که مدل برای این کلاس‌ها ممکن است مطمئن نباشد یا نمونه‌های کافی برای آن‌ها وجود نداشته باشد.

منحنی ضخیم‌آبی (all classes) :

این منحنی نشان‌دهنده عملکرد کلی مدل در تمام کلاس‌ها است. در نقطه‌ای از این منحنی، نمره F1 به حد اکثر مقدار خود رسیده است (نقطه‌ای که با مقدار 0.70 مشخص شده است) که مربوط به سطح confidence است. این نقطه بهینه‌ای است که مدل بهترین تعادل را بین دقت و فراخوانی دارد.

۳- منحنی (Precision-Confidence)



شکل ۴-۵ : (Precision-Confidence)

محور افقی (Confidence):

این محور سطح confidence یا آستانه اطمینان مدل را نشان می‌دهد. هرچه confidence بالاتر باشد، مدل در پیش‌بینی خود مطمئن‌تر است.

محور عمودی (Precision):

این محور میزان دقیقت را نشان می‌دهد. دقیقت به این معناست که از پیش‌بینی‌های مثبت مدل، چه تعداد آن‌ها واقعاً درست هستند.

منحنی‌های رنگی:

هر منحنی رنگی نشان‌دهنده‌ی کلاس یکی از علائم ترافیکی است. این منحنی‌ها نشان می‌دهند که چگونه دقیقت مدل برای هر کلاس با تغییر مقدار confidence تغییر می‌کند.

برای برخی از کلاس‌ها، با افزایش confidence، دقیقت بهبود می‌یابد. این به این معناست که مدل با اطمینان بیشتر، پیش‌بینی‌های دقیق‌تری ارائه می‌دهد.

فصل ۵. آموزش و استفاده از مدل آموزش داده شده

۵-۵. بررسی نمودارها:

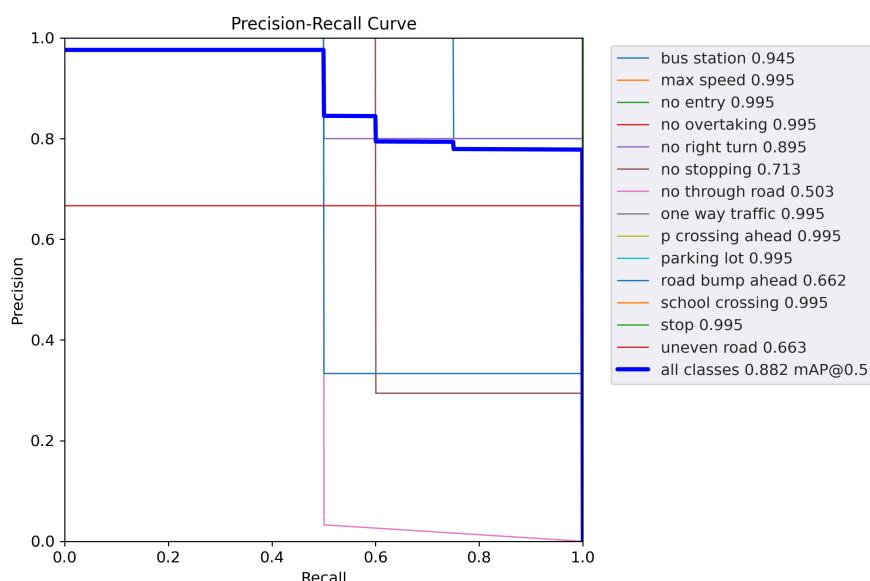
برخی از منحنی‌ها ممکن است نوساناتی داشته باشند که نشان‌دهنده عملکرد متغیر مدل در سطوح مختلف confidence برای آن کلاس‌ها است.

منحنی‌آبی (all classes):

این منحنی نمایانگر عملکرد مدل در کل کلاس‌ها است. نقطه‌ای که دقต به ۱۰۰٪ می‌رسد، جایی است که مدل به بالاترین دقیق خود دست یافته است. این نقطه در مقدار confidence برابر با ۰.۸۶۱ رخ داده است.

این نقطه نشان می‌دهد که در آستانه ۰.۸۶۱ confidence، مدل توانسته است تمام پیش‌بینی‌های خود را درست انجام دهد، یعنی دقیق ۱۰۰٪ داشته باشد.

۴ - منحنی (Precision-Recall)



شکل ۵-۵: (Precision-Confidence)

: محور افقی (Recall)

این محور مقدار فراخوانی را نشان می‌دهد که نسبت تعداد پیش‌بینی‌های درست مثبت به کل نمونه‌های مثبت واقعی است.

محور عمودی (Precision):

این محور مقدار دقیقت را نشان می‌دهد که نسبت تعداد پیش‌بینی‌های درست مشبت به کل پیش‌بینی‌های مشبت است.

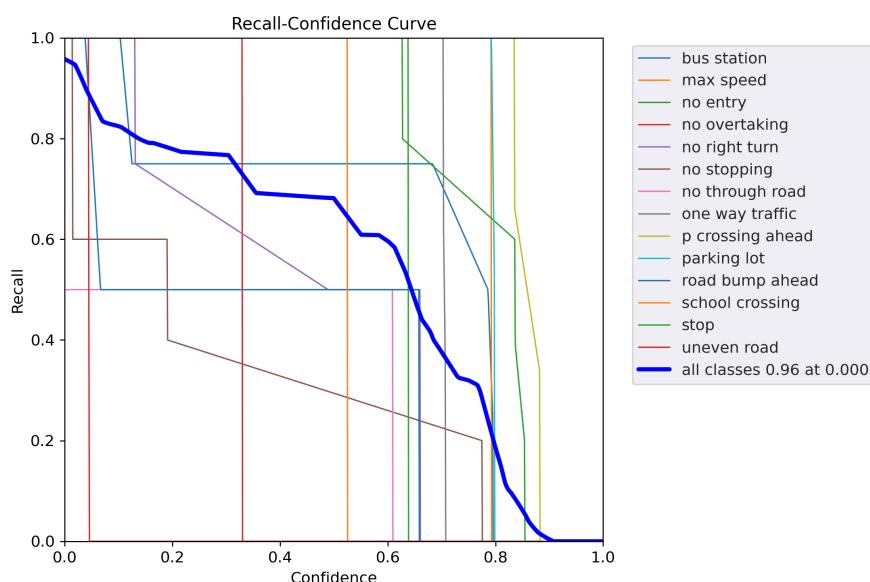
منحنی‌های رنگی:

هر منحنی رنگی نمایانگر عملکرد مدل برای کلاس یکی از علائم ترافیکی است. اعداد کنار نام هر کلاس (مثلًا "0.995") نشان دهنده مقدار دقیقت آن کلاس در بهترین حالت است.

منحنی ضخیم آبی رنگ (all classes):

این منحنی عملکرد کلی مدل را برای همه کلاس‌ها نشان می‌دهد. مقدار $0.882 \text{ mAP}@0.5$ در اینجا نشان دهنده میانگین دقیقت (mean Average Precision) مدل برای همه کلاس‌ها در آستانه 0.5 است. این معیار معمولاً برای مقایسه عملکرد کلی مدل‌ها استفاده می‌شود.

۵ - منحنی (Recall-Confidence Curve)



شکل ۶-۵: (Precision-Confidence)

محور افقی Recall :

درصدی از کل نمونه‌های مشبت (در اینجا، علائم درست شناسایی شده) که مدل به درستی شناسایی کرده است. محور عمودی (Y) میزان Recall را نشان می‌دهد که از ۰ تا ۱ متغیر است.

محور عمودی Confidence :

میزان اطمینانی که مدل به تصمیم خود دارد. این مقدار معمولاً از ۰ (کمترین اطمینان) تا ۱ (بیشترین اطمینان) متغیر است. محور افقی (X) میزان اطمینان را نشان می‌دهد.

تحلیل منحنی:

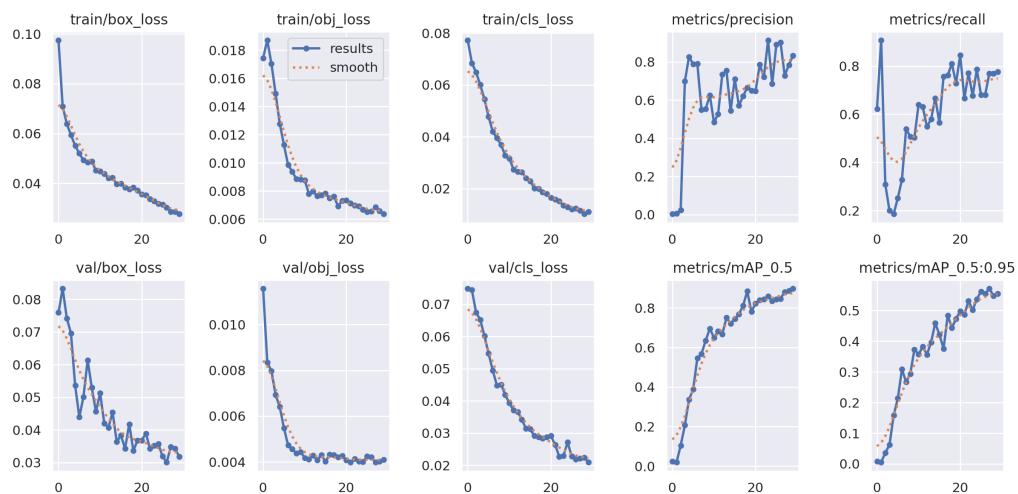
خطوط رنگی نازک:

هر کدام از این خطوط نشان‌دهنده عملکرد مدل در شناسایی یک کلاس خاص (مثلًا "stop") در سطوح مختلف اطمینان است. این خطوط نشان می‌دهند که مدل در چه سطح اطمینانی، چه میزان از حساسیت را برای هر کلاس مختلف به دست آورده است.

خط آبی ضخیم:

این خط میانگین عملکرد مدل در تمامی کلاس‌ها را نشان می‌دهد. همان‌طور که مشاهده می‌شود، با افزایش مقدار اطمینان (به سمت راست حرکت روی محور افقی)، مقدار بازخوانی کاهش می‌یابد.

۶- منحنی های نتایج:



شکل ۷-۵ :

نمودارهای ردیف اول (مربوط به داده‌های آموزشی):

1. train/box-loss:

- این نمودار نشان‌دهنده کاهش خطای مدل در پیش‌بینی موقعیت و اندازه

جعبه‌های محدود کننده در داده‌های آموزشی است. کاهش این مقدار نشان می‌دهد که مدل در پیش‌بینی جعبه‌های محدود کننده دقیق‌تر شده است.

2. train/obj-loss:

- این نمودار میزان خطای مدل در تشخیص وجود یا عدم وجود یک شیء در جعبه محدود کننده پیش‌بینی شده را نشان می‌دهد. کاهش این مقدار نشان‌دهنده بهبود دقت مدل در شناسایی اشیاء در داده‌های آموزشی است.

3. train/cls-loss:

- این نمودار خطای مدل در طبقه‌بندی صحیح اشیاء در داخل جعبه‌های محدود کننده را نشان می‌دهد. کاهش این مقدار به معنای بهبود دقت مدل در تشخیص نوع شیء در داده‌های آموزشی است.

4. metrics/precision:

- این نمودار دقت مدل (Precision) در داده‌های آموزشی را نشان می‌دهد. Precision نشان‌دهنده درصد اشیاء شناسایی شده توسط مدل است که به درستی شناسایی شده‌اند. افزایش این مقدار به معنای کاهش خطاهای مثبت کاذب است.

5. metrics/recall:

- این نمودار میزان بازخوانی مدل در داده‌های آموزشی را نشان می‌دهد. Recall نشان‌دهنده درصد کل اشیاء موجود در تصویر است که به درستی شناسایی شده‌اند. افزایش این مقدار به معنای کاهش خطاهای منفی کاذب است.

نمودارهای ردیف دوم (مریبوط به داده‌های اعتبارسنجی):

6. val/box-loss:

- مشابه با نمودار train/box-loss، اما مریبوط به داده‌های اعتبارسنجی است. کاهش این مقدار نشان می‌دهد که مدل در داده‌های ناآشننا (داده‌هایی که در فرآیند آموزش استفاده نشده‌اند) نیز به خوبی عمل می‌کند.

7. val/obj-loss:

- مشابه با نمودار train/obj-loss، اما مریبوط به داده‌های اعتبارسنجی است. کاهش این مقدار نشان‌دهنده عملکرد خوب مدل در شناسایی اشیاء در داده‌های اعتبارسنجی است.

8. val/cls-loss:

- مشابه با نمودار train/cls-loss اما مربوط به داده‌های اعتبارسنجی است. کاهش این مقدار نشان می‌دهد که مدل در داده‌های ناآشنا نیز به خوبی قادر به طبقه‌بندی اشیاء است.

9. metrics/mAP-0.5:

- این نمودار میانگین دقت mAP را با آستانه 0.5 نشان می‌دهد. mAP یک معیار کلی برای ارزیابی عملکرد مدل در تشخیص اشیاء است و افزایش این مقدار نشان دهنده بهبود کلی مدل است.

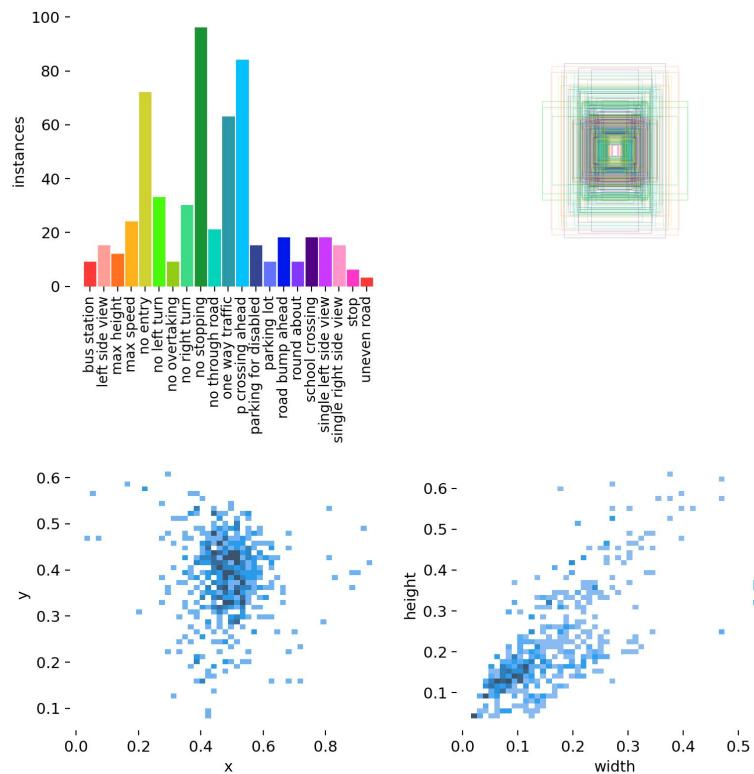
10. metrics/mAP-0.5:0.95:

- این نمودار میانگین دقت mAP را در بازه‌ای از آستانه‌های مختلف (0.95 - 0.5) نشان می‌دهد. این معیار دقیق‌تر از mAP-0.5 است و افزایش آن به معنای عملکرد بهتر مدل در تشخیص و طبقه‌بندی اشیاء در شرایط مختلف است.

تحلیل کلی:

- کاهش مقادیر Loss: نشان دهنده بهبود مدل در هر مرحله از آموزش و اعتبارسنجی است.
- افزایش Precision و Recall: به معنای کاهش خطاهای مثبت و منفی کاذب و افزایش دقت و بازخوانی مدل است.
- افزایش mAP: بیان‌گر بهبود کلی مدل در تشخیص و طبقه‌بندی اشیاء است.

۷- labels منحنی های :



شکل ۸-۵ : labels

۱. نمودار میله‌ای (Bar Chart) در بالا سمت چپ:

هدف: این نمودار توزیع تعداد نمونه‌های موجود در هر کلاس را نشان می‌دهد.

محور افقی:

نام کلاس‌های مختلف را نمایش می‌دهد. این کلاس‌ها شامل مواردی مانند "bus station" ، و غیره هستند.

محور عمودی: تعداد نمونه‌ها (instances) در هر کلاس را نشان می‌دهد.

نتیجه‌گیری: این نمودار کمک می‌کند تا عدم تعادل در کلاس‌های داده‌ها را شناسایی شود.

۲. نمودار جعبه‌های محدود کننده (Bounding Box Visualization) در بالا سمت راست:

فصل ۵. آموزش و استفاده از مدل آموزش داده شده

۵-۵. بررسی نمودارها:

هدف: این نمودار همه جعبه‌های محدود کننده در داده‌ها را روی همدیگر قرار می‌دهد تا الگوی کلی قرارگیری آنها در تصویر مشخص شود.

ویژگی‌ها: هر جعبه یک مستطیل است که نمایانگر یک جعبه محدود کننده از داده‌های می‌باشد.
نتیجه‌گیری: این نمودار نشان می‌دهد که جعبه‌های محدود کننده معمولاً در چه نواحی از تصویر قرار دارند.

۳. نمودار پراکنده‌گی(Scatter Plot)

هدف: این نمودار موقعیت‌های x و y مرکز جعبه‌های مرزی را نشان می‌دهد.

محور افقی: مختصات x (موقعیت افقی مرکز جعبه محدود کننده) را نشان می‌دهد.

محور عمودی: مختصات y (موقعیت عمودی مرکز جعبه محدود) را نشان می‌دهد.

نتیجه‌گیری: این نمودار توزیع جغرافیایی جعبه‌های محدود کننده را در تصاویر نشان می‌دهد.

۴. نمودار پراکنده‌گی(Scatter Plot)

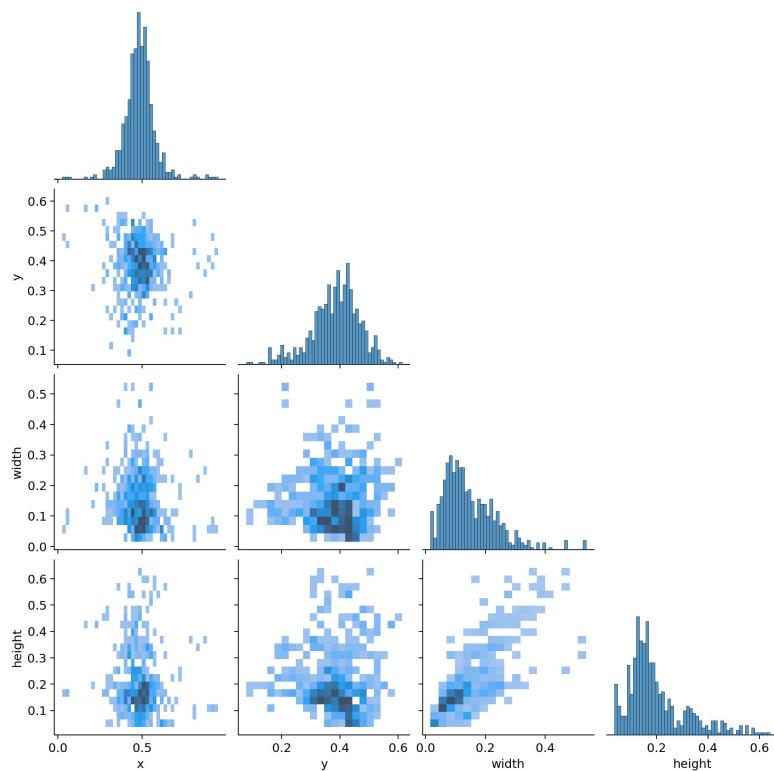
هدف: این نمودار عرض (width) و ارتفاع (height) جعبه‌های محدود کننده را نشان می‌دهد.

محور افقی: عرض جعبه محدود کننده را نمایش می‌دهد.

محور عمودی: ارتفاع جعبه محدود کننده را نمایش می‌دهد.

نتیجه‌گیری: این نمودار به بررسی رابطه بین عرض و ارتفاع جعبه‌های مرزی می‌پردازد.

labels-correlogram - A منحنی های :



شکل ۹-۵ labels-correlogram :

۱. ساختار کلی نمودار:

محورهای X و Y : این محورها چهار ویژگی مختلف را نشان می‌دهند:

X : مختصات افقی مرکز جعبه محدود کننده.

y : مختصات عمودی مرکز جعبه محدود کننده.

width: عرض جعبه محدود کننده.

height: ارتفاع جعبه محدود کننده.

زیر قطراصی: شامل نمودارهای پراکندگی (Scatter Plots) است که رابطه بین دو ویژگی مختلف را نشان می‌دهند.

فصل ۵. آموزش و استفاده از مدل آموزش داده شده

۵-۶. استفاده از مدل آموزش داده شده

روی قطر اصلی: شامل نمودارهای هیستوگرام است که توزیع هر ویژگی را به تنها یک نمایش می‌دهد.

۲. تجزیه و تحلیل نمودارهای پراکنده (Scatter Plots): این نمودارها روابط دو به دوی ویژگی‌ها را نشان می‌دهند. به عنوان مثال:

نمودار x در مقابل y : این نمودار نشان می‌دهد که چگونه نقاط داده‌های شما (یعنی جعبه‌های محدود کننده) در فضای تصویر توزیع شده‌اند. یک تراکم مرکزی مشاهده می‌شود که نشان‌دهنده این است که اکثر جعبه‌ها در نزدیکی مرکز تصویر قرار دارند.

نمودار $width$ در مقابل $height$: این نمودار ارتباط بین عرض و ارتفاع جعبه‌های محدود کننده را نشان می‌دهد. اگر رابطه قوی وجود داشته باشد (مثل یک خط مستقیم)، می‌تواند نشان‌دهنده این باشد که بیشتر جعبه‌های محدود کننده دارای نسبت ابعادی مشابهی هستند.

۳. تجزیه و تحلیل هیستوگرام‌ها: هیستوگرام‌های روی قطر نشان می‌دهند که مقادیر هر ویژگی به چه صورت توزیع شده‌اند:

هیستوگرام x : نشان می‌دهد که مختصات افقی مرکز جعبه‌ها بیشتر در وسط تصویر قرار دارند.
هیستوگرام y : توزیع مشابهی برای مختصات عمودی مرکز جعبه‌ها نشان می‌دهد، که باز هم تمکن جعبه‌ها در وسط تصویر است.

هیستوگرام‌های $width$ و $height$: این هیستوگرام‌ها نشان می‌دهند که عرض و ارتفاع جعبه‌ها معمولاً در محدوده‌های کوچکی قرار دارند، که نشان‌دهنده این است که اکثر جعبه‌ها دارای ابعاد کوچکی هستند.

۵-۶ استفاده از مدل آموزش داده شده

برای استفاده از مدل از کد زیر استفاده می‌کنیم:

```
!python /content/yolov5/detect.py -weights /content/yolov5/runs/train/exp/weights/best.pt  
-source /content/gdrive/MyDrive/pic/msg21535093-3576.jpg
```

فصل ۵. آموزش و استفاده از مدل آموزش داده شده

<content/yolov5/runs/train/exp/weights/best.pt>

این بخش به آدرس محل فایل best.pt را اشاره می کند. و باعث می شود مدل از آن استفاده کند.

</content/gdrive/MyDrive/pic/msg21535093-3576.jpg>

این بخش به آدرس عکس یا ویدیو ای که قصد تشخیص اشیا در آن را داریم اشاره می کند.

مدل پس از اتمام فرآیند تشخیص اشیا ، نتیجه را داخل پوشه exp ذخیره می نماید.

نکته : اگر تشخیص اشیا برای یک فیلم انجام شود ، فیلم به صورت فریم به فریم بررسی می شود و وجود هر شی در فریم گزارش می شود.

```

● v10eo 1/1 (27/134) /content/gdrive/MyDrive/pic/_2147483648_-210017.mp4: 640x384 1 no stopping, 10.1ms
video 1/1 (58/134) /content/gdrive/MyDrive/pic/_2147483648_-210017.mp4: 640x384 1 no stopping, 10.1ms
video 1/1 (59/134) /content/gdrive/MyDrive/pic/_2147483648_-210017.mp4: 640x384 1 no stopping, 10.2ms
video 1/1 (60/134) /content/gdrive/MyDrive/pic/_2147483648_-210017.mp4: 640x384 1 no stopping, 10.1ms
video 1/1 (61/134) /content/gdrive/MyDrive/pic/_2147483648_-210017.mp4: 640x384 1 no stopping, 10.0ms
video 1/1 (62/134) /content/gdrive/MyDrive/pic/_2147483648_-210017.mp4: 640x384 1 no stopping, 10.0ms
video 1/1 (63/134) /content/gdrive/MyDrive/pic/_2147483648_-210017.mp4: 640x384 1 no stopping, 9.9ms
video 1/1 (64/134) /content/gdrive/MyDrive/pic/_2147483648_-210017.mp4: 640x384 1 no stopping, 9.9ms
video 1/1 (65/134) /content/gdrive/MyDrive/pic/_2147483648_-210017.mp4: 640x384 1 no stopping, 9.8ms
video 1/1 (66/134) /content/gdrive/MyDrive/pic/_2147483648_-210017.mp4: 640x384 1 no stopping, 9.8ms
video 1/1 (67/134) /content/gdrive/MyDrive/pic/_2147483648_-210017.mp4: 640x384 1 no stopping, 9.8ms
video 1/1 (68/134) /content/gdrive/MyDrive/pic/_2147483648_-210017.mp4: 640x384 1 no stopping, 9.8ms
video 1/1 (69/134) /content/gdrive/MyDrive/pic/_2147483648_-210017.mp4: 640x384 1 no stopping, 9.8ms
video 1/1 (70/134) /content/gdrive/MyDrive/pic/_2147483648_-210017.mp4: 640x384 1 no stopping, 9.8ms
video 1/1 (71/134) /content/gdrive/MyDrive/pic/_2147483648_-210017.mp4: 640x384 1 no stopping, 9.9ms
video 1/1 (72/134) /content/gdrive/MyDrive/pic/_2147483648_-210017.mp4: 640x384 1 no stopping, 9.8ms
video 1/1 (73/134) /content/gdrive/MyDrive/pic/_2147483648_-210017.mp4: 640x384 1 no stopping, 11.0ms
video 1/1 (74/134) /content/gdrive/MyDrive/pic/_2147483648_-210017.mp4: 640x384 1 no stopping, 9.8ms
video 1/1 (75/134) /content/gdrive/MyDrive/pic/_2147483648_-210017.mp4: 640x384 1 no stopping, 9.8ms
video 1/1 (76/134) /content/gdrive/MyDrive/pic/_2147483648_-210017.mp4: 640x384 1 no stopping, 9.8ms
video 1/1 (77/134) /content/gdrive/MyDrive/pic/_2147483648_-210017.mp4: 640x384 1 no stopping, 10.9ms
video 1/1 (78/134) /content/gdrive/MyDrive/pic/_2147483648_-210017.mp4: 640x384 1 no stopping, 9.8ms
video 1/1 (79/134) /content/gdrive/MyDrive/pic/_2147483648_-210017.mp4: 640x384 1 no stopping, 9.8ms
video 1/1 (80/134) /content/gdrive/MyDrive/pic/_2147483648_-210017.mp4: 640x384 1 no stopping, 9.8ms
video 1/1 (81/134) /content/gdrive/MyDrive/pic/_2147483648_-210017.mp4: 640x384 1 no stopping, 9.9ms
video 1/1 (82/134) /content/gdrive/MyDrive/pic/_2147483648_-210017.mp4: 640x384 1 no stopping, 9.8ms
video 1/1 (83/134) /content/gdrive/MyDrive/pic/_2147483648_-210017.mp4: 640x384 1 no stopping, 9.8ms
video 1/1 (84/134) /content/gdrive/MyDrive/pic/_2147483648_-210017.mp4: 640x384 1 no stopping, 9.8ms
video 1/1 (85/134) /content/gdrive/MyDrive/pic/_2147483648_-210017.mp4: 640x384 1 no stopping, 9.8ms
video 1/1 (86/134) /content/gdrive/MyDrive/pic/_2147483648_-210017.mp4: 640x384 1 no stopping, 9.8ms
video 1/1 (87/134) /content/gdrive/MyDrive/pic/_2147483648_-210017.mp4: 640x384 1 no stopping, 9.8ms
video 1/1 (88/134) /content/gdrive/MyDrive/pic/_2147483648_-210017.mp4: 640x384 1 no stopping, 9.8ms
video 1/1 (89/134) /content/gdrive/MyDrive/pic/_2147483648_-210017.mp4: 640x384 1 no stopping, 9.8ms
video 1/1 (90/134) /content/gdrive/MyDrive/pic/_2147483648_-210017.mp4: 640x384 1 no stopping, 12.0ms
video 1/1 (91/134) /content/gdrive/MyDrive/pic/_2147483648_-210017.mp4: 640x384 1 no stopping, 13.2ms

```

شکل ۱۰-۵ : فرآیند تشخیص اشیا برای یک ویدیو

آپوست

بررسی فایل val.py

آ-۱ کتابخانه ها

```
import argparse
import json
import os
import subprocess
import sys
from pathlib import Path
import numpy as np
import torch
from tqdm import tqdm
```

argparse:

این کتابخانه به شما اجازه می دهد آرگومان های خط فرمان را مدیریت و پارس کنید. این امکان را فراهم می کند تا به راحتی ورودی های مورد نیاز برنامه خود را از خط فرمان دریافت کنید و گزینه های مختلفی را برای کاربر فراهم آورید.

json:

این کتابخانه برای کار با داده های JSON استفاده می شود. می توانید از json برای تبدیل داده های JSON به اشیای پایتون و بالعکس استفاده کنید. این کتابخانه معمولاً برای ذخیره سازی و تبادل داده ها با فرمت JSON بسیار مفید است.

os:

کتابخانه ای برای انجام عملیات سیستم عاملی مانند مدیریت فایل ها و دایرکتوری ها، دسترسی به متغیر های محیطی و اجرای دستورات سیستم os ابزاری قدرتمند برای تعامل با سیستم عامل در داخل اسکریپت های پایتون است.

subprocess:

این کتابخانه برای اجرا و مدیریت پروسه های سیستم عاملی استفاده می شود. با استفاده از subprocess می توانید دستورات خارجی را اجرا کرده و خروجی آنها را در داخل پایتون مدیریت کنید.

sys:

کتابخانه ای که امکان دسترسی به پارامترها و توابع سطح پایین مرتبط با مفسر پایتون و سیستم عامل را فراهم می کند. sys اغلب برای مدیریت آرگومان های خط فرمان و مسیر ماژول ها استفاده می شود.

pathlib:

کتابخانه‌ای مدرن برای کار با مسیرهای فایل و دایرکتوری‌ها. Path کلاس اصلی این کتابخانه است که کار با مسیرها را ساده و شهودی می‌کند.

numpy (np):

یکی از کتابخانه‌های پایه‌ای برای محاسبات علمی و داده‌پردازی در پایتون. numpy آرایه‌های چند بعدی و مجموعه‌ای از توابع ریاضی برای کار با این آرایه‌ها را فراهم می‌کند. np مخفف رایج numpy است که به طور گسترده در جامعه پایتون استفاده می‌شود.

torch:

این کتابخانه بخش اصلی PyTorch، یکی از فریمورک‌های محبوب یادگیری عمیق است. torch برای تعریف و آموزش مدل‌های شبکه‌های عصبی استفاده می‌شود و قابلیت‌هایی برای انجام محاسبات GPU را فراهم می‌کند.

tqdm:

کتابخانه‌ای برای ایجاد نوار پیشرفت (progress bar) در پایتون. tqdm به شما اجازه می‌دهد تا به سادگی وضعیت پیشرفت یک حلقه یا فرآیند طولانی را در کنسول یا نوت‌بوک‌های Jupyter نمایش دهید.

آ-۲ بخش های مهم *val.py*

Initialize/load model and set device:

training = model is not None

if training: **called by train.py**

`device, pt, jit, engine = next(model.parameters()).device, True, False, False` **get model**

`device, PyTorch model`

`half = device.type != "cpu" half precision only supported on CUDA`

`model.half() if half else model.float() else: called directly`

`device = select-device(device, batch-size=batch-size)`

این بخش برای تنظیم مدل و دستگاه محاسباتی (CPU یا GPU) استفاده می شود به طور خلاصه:

تشخیص حالت اجرا:

اگر `model` مقداردهی شده باشد، یعنی کد در حالت آموزش (`training=True`) است و توسط `train.py` فراخوانی شده است. اگر `model` مقداردهی نشده باشد، یعنی کد به صورت مستقیم برای ارزیابی فراخوانی شده است. تنظیمات در حالت آموزش:

دستگاه (`device`) که مدل روی آن اجرا می شود از پارامترهای مدل استخراج می شود. اگر دستگاه GPU باشد، مدل می تواند به صورت half-precision اجرا شود (که محاسبات سریع تر و حافظه کمتری نیاز دارد). مدل با دقت مناسب (نصف یا کامل) تنظیم می شود. تنظیمات در حالت ارزیابی:

اگر کد مستقیم اجرا شود، دستگاه مناسب با استفاده از تابع `select-device` انتخاب می شود.

Directories:

`save-dir = increment-path(Path(project) / name, exist-ok=exist-ok)` **increment run**

`(save-dir / "labels" if save-txt else save-dir).mkdir(parents=True, exist-ok=True)` **make dir**

این بخش وظیفه ایجاد دایرکتوری برای ذخیره نتایج ارزیابی را دارد. به صورت خلاصه:

تعیین مسیر ذخیره سازی:

مسیرنهایی (`save-dir`) بر اساس پارامترهای `project` و `name` ایجاد می شود. اگر پوشه ای با این نام وجود داشته باشد، نام جدید با افزودن یک شماره افزایش می یابد. ایجاد دایرکتوری ها:

پوشه های لازم برای ذخیره نتایج ساخته می شوند. اگر save-txt فعال باشد، یک زیر پوشه labels نیز ایجاد می شود.

Load model:

```
model = DetectMultiBackend(weights, device=device, dnn=dnn, data=data, fp16=half)
```

```
stride, pt, jit, engine = model.stride, model.pt, model.jit, model.engine
```

imgsz = check-img-size(imgsz, s=stride) **check image size**

half = model.fp16 **FP16 supported on limited backends** with CUDA if engine:

```
batch-size = model.batch-size
```

```
else:
```

```
device = model.device
```

```
if not (pt or jit):
```

batch-size = 1 **export.py models default to batch-size 1**

LOGGER.info(f'Forcing –batch-size 1 square inference (1,3,imgsz,imgsz) for non-PyTorch
models”)

این بخش به بارگذاری و تنظیم مدل YOLOv5 مربوط می شود به طور خلاصه:

بارگذاری مدل: مدل با پارامترهای ورودی بارگذاری می شود.

استخراج ویژگی ها:

ویژگی های مدل مانند اندازه گام (stride)، نوع مدل (pt، jit، engine) استخراج می شود.

بررسی و تنظیم اندازه تصویر:

اندازه تصویر با توجه به ویژگی های مدل بررسی و تنظیم می شود.

تنظیم اندازه دسته:

اندازه دسته (batch-size) بر اساس نوع مدل تنظیم می شود؛ برای مدل های غیر PyTorch به ۱ تنظیم می شود. به طور خلاصه، این کد مدل را بازگذاری کرده و پارامترهای آن را تنظیم می کند.

[Configure:](#)

```
model.eval()
```

```
cuda = device.type != "cpu"
```

```
is_coco = isinstance(data.get("val"), str) and
```

```
data["val"].endswith(f"cocoos.sepval2017.txt") COCO dataset
```

```
nc = 1 if single_cls else int(data["nc"]) number of classes
```

```
iouv = torch.linspace(0.5, 0.95, 10, device=device) iou vector for mAP@0.5:0.95
```

```
niou = iouv.numel()
```

این بخش به پیکربندی مدل برای ارزیابی مربوط می شود. به طور خلاصه:

تنظیم مدل برای ارزیابی:

مدل به حالت ارزیابی (eval) تنظیم می شود که برای ارزیابی عملکرد استفاده می شود.

بررسی نوع دستگاه:

بررسی می شود که آیا مدل بر روی GPU (CUDA) اجرا می شود یا بر روی CPU.

تشخیص داده های COCO :

بررسی می شود که آیا داده های ارزیابی از نوع COCO هستند یا خیر، با توجه به مسیر فایل داده ها.

تعیین تعداد کلاس ها:

تعداد کلاس ها بر اساس پارامتر `single-cls` یا اطلاعات داده ها (`data["nc"]`) تعیین می شود.

ایجاد بردار IoU :

برداری از مقادیر *IoU* برای محاسبه میانگین دقت (mAP) در مقادیر مختلف *IoU* (از ۰.۵ تا ۰.۹۵) ایجاد می شود.

به طور خلاصه، این کد مدل را برای ارزیابی پیکربندی می کند و پارامترهای مورد نیاز برای محاسبه دقت و تحلیل داده ها را تنظیم می کند.

Dataloader:

if not training:

if pt and not single-cls: **check –weights are trained on –data**

`ncm = model.model.nc`

`assert ncm == nc, (f'weights (ncm classes) trained on different –data than what you passed
(nc ”`

`f'classes). Pass correct combination of –weights and –data that are trained together.”)`

`model.warmup(imgsz=(1 if pt else batch-size, 3, imgsz, imgsz)) warmup`

`pad, rect = (0.0, False) if task == ”speed” else (0.5, pt) square inference for benchmarks`

`task = task if task in (“train”, ”val”, ”test”) else ”val” path to train/val/test images`

`dataloader = create-dataloader(`

`data[task],`

`imgsz,`

`batch-size,`

`stride,`

`single-cls,`

```
pad=pad,  
rect=rect,  
workers=workers,  
prefix=colorstr(f'task: '), )[0]  
  
seen = 0 confusion-matrix = ConfusionMatrix(nc=nc)  
  
names = model.names if hasattr(model, "names") else  
  
model.module.names get class names  
  
if isinstance(names, (list, tuple)): old format names = dict(enumerate(names))  
  
class-map = coco80-to-coco91-class() if is-coco else list(range(1000))  
  
s = (%.22s" + "%.11s" * 6) % ("Class", "Images", "Instances", "P", "R", "mAP50",  
"mAP50-95")  
  
tp, fp, p, r, f1, mp, mr, map50, ap50, map = 0. 0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0  
  
dt = Profile(device=device), Profile(device=device),  
  
Profile(device=device) profiling times loss = torch.zeros(3, device=device)  
  
jdict, stats, ap, ap-class = [], [], [], []  
  
callbacks.run("on-val-start")  
  
pbar = tqdm(dataloader, desc=s, bar-format=TQDM-BAR-FORMAT) progress bar  
  
for batch-i, (im, targets, paths, shapes) in enumerate(pbar):  
  
callbacks.run("on-val-batch-start")  
  
with dt[0]:  
  
if cuda:  
  
im = im.to(device, non-blocking=True)
```

```
targets = targets.to(device)
```

```
im = im.half() if half else im.float()
```

```
im /= 255
```

`nb, -, height, width = im.shape` **batch size, channels, height, width**

این بخش به تنظیم و استفاده از داده‌لودر برای بارگذاری داده‌ها مربوط می‌شود. به طور خلاصه:

پیکربندی داده‌لودر:

اگر مدل در حال آموزش نیست: بررسی سازگاری وزن‌ها: اگر مدل PyTorch باشد و تعداد کلاس‌ها در وزن‌های مدل با تعداد کلاس‌های داده ورودی مطابقت نداشته باشد، خطایجاد می‌شود.
گرم‌کردن مدل: مدل برای اندازه تصویر معین گرم می‌شود تا عملکرد بهینه داشته باشد.

تنظیمات `Padding` و نوع `Inference`:

براساس نوع [وظیفه](#)^۱، تنظیمات مربوط به `padding` و شکل `inference` (مستطیلی یا مربعی) تعیین می‌شود.

ایجاد داده‌لودر:

داده‌لودر برای بارگذاری داده‌های ارزیابی با تنظیمات مناسب ایجاد می‌شود.

آماده‌سازی برای ارزیابی:

تنظیم ماتریس سردرگمی: برای محاسبه عملکرد مدل. دریافت نام کلاس‌ها: نام‌های کلاس‌ها از مدل استخراج و به دیکشنری تبدیل می‌شود. تنظیم نقشه کلاس‌ها: برای مجموعه داده COCO یا لیست پیش‌فرض کلاس‌ها. تنظیمات اولیه: متغیرهای مورد نیاز برای جمع‌آوری نتایج و پروفایلینگ زمان‌ها مقداردهی اولیه می‌شود.

ایجاد نوار پیشرفت: برای نمایش وضعیت پردازش `batch`‌ها.

¹task

پردازش *batch* ها:

پیش پردازش داده ها: برای هر *batch*، داده ها به دستگاه مناسب منتقل شده و پیش پردازش هایی مانند تبدیل نوع داده و نرمال سازی انجام می شود. به طور خلاصه، این کد داده لودر را تنظیم کرده و داده ها را برای ارزیابی مدل آماده می کند و فرآیند ارزیابی را با نظارت بر پیشرفت آغاز می کند.

Inference:

with *dt[1]*:

preds, train-out = model(im) if *compute-loss* else (*model(im)**augment=augment*), *None*)

به طور خلاصه، این کد مرحله استنباط مدل را انجام می دهد و خروجی های پیش بینی را با توجه به اینکه آیا محاسبه زیان لازم است یا نه، تولید می کند.

Loss:

if *compute-loss*:

loss += compute-loss(train-out, targets)[1] **box, obj, cls**

این کد ضرر هارا محاسبه می کند به طور خلاصه :

بررسی نیاز به محاسبه ضرر:

اگر *compute-loss* فعال باشد (یعنی محاسبه زیان لازم باشد)، ضررها محاسبه می شوند.

افزودن زیان:

ضرر های محاسبه شده (*train-out*) به مجموع ضرر های قبلی (*loss*) اضافه می شود. ضرر ها شامل ضرر های جعبه (*box*)، شیء (*obj*)، و کلاس (*cls*) هستند.

NMS:

```
targets[:, 2:] *= torch.tensor((width, height, width, height), device=device) to pixels
```

```
lb = [targets[targets[:, 0] == i, 1:] for i in range(nb)] if save-hybrid else [] for autolabelling
```

with $dt[2]$:

```
preds = non-max-suppression(
```

```
preds, conf-thres, iou-thres, labels=lb, multi-label=True, agnostic=single-cls,
```

```
max-det=max-det )
```

این بخش به مرحله انجام NMS و پردازش اهداف مربوط می‌شود. به طور خلاصه:

تبدیل مختصات به پیکسل:

مختصات اهداف که به صورت نسبی (در نسبت‌های عرض و ارتفاع) هستند، به پیکسل تبدیل می‌شوند با ضرب کردن آن‌ها در ابعاد تصویر.

آماده‌سازی برچسب‌ها:

اگر *save-hybrid* فعال نباشد، برای هر تصویر لیستی از برچسب‌ها (*lb*) ایجاد می‌شود. این لیست شامل برچسب‌های مربوط به هر کلاس است که برای اتوماتیک‌سازی برچسب‌گذاری استفاده می‌شود.

: انجام NMS

با استفاده از تابع *no-max-suppression*، پیش‌بینی‌ها (preds) پس از بررسی آستانه اطمینان و آستانه *IoU* پردازش می‌شوند.

تنظیمات اضافی شامل برچسب‌ها، پشتیبانی از چند برچسب، و تشخیص غیروابسته به کلاس‌ها و بیشترین تعداد نتایج انجام می‌شود.

Metrics:

for si, pred in enumerate(preds):

```
labels = targets[targets[:, 0] == si, 1:]
```

nl, npr = labels.shape[0], pred.shape[0] **number of labels, predictions**

```
path, shape = Path(paths[si]), shapes[si][0]
```

```
correct = torch.zeros(npr, niou, dtype=torch.bool, device=device)
```

```
seen += 1
```

```
if npr == 0:
```

```
if nl:
```

```
stats.append((correct, *torch.zeros((2, 0), device=device), labels[:, 0]))
```

```
if plots:
```

```
confusion-matrix.process-batch(detections=None, labels=labels[:, 0])
```

```
continue
```

این بخش مربوط به محاسبه و بهروزرسانی متریک‌های عملکرد مدل پس از اعمال *NMS* است. به

طور خلاصه:

تنظیمات اولیه:

برای هر تصویر در دسته (batch) :

برچسب‌های مرتبط با آن تصویر و پیش‌بینی‌های مدل استخراج می‌شوند.

تعداد برچسب‌ها (nl) و تعداد پیش‌بینی‌ها (npr) مشخص می‌شود. مسیر تصویر (path) و شکل تصویر (shape) نیز استخراج می‌شود. متغیر correct برای نگهداری وضعیت صحیح بودن پیش‌بینی‌ها نسبت به آستانه‌های IoU تنظیم می‌شود. شمارشگر seen برای تعداد تصاویر پردازش شده افزایش می‌یابد.

بررسی شرایط پیش‌بینی‌ها:

اگر پیش‌بینی‌ای (npr) وجود نداشته باشد:

اگر برجسبی (nl) وجود داشته باشد:

متغیرهای stats با اطلاعات مربوط به برجسب‌ها و وضعیت پیش‌بینی‌های نادرست، به روز می‌شود. اگر گزینه رسم نمودار (plots) فعال باشد، ماتریس سردرگمی (confusion-matrix) با توجه به برجسب‌ها به روز می‌شود. اگر هیچ پیش‌بینی‌ای وجود نداشته باشد، حلقه برای پردازش تصویر بعدی ادامه می‌یابد.

Predictions:

if single-cls:

```
pred[:, 5] = 0
```

```
predn = pred.clone()
```

```
scale-boxes(im[si].shape[1:], predn[:, :4], shape, shapes[si][1]) native-space pred
```

این بخش مربوط به پردازش و آماده‌سازی پیش‌بینی‌های مدل برای ارزیابی است. به طور خلاصه:

تنظیم کلاس برای مدل‌های تک‌کلاسی:

اگر مدل به صورت تک‌کلاسی تنظیم شده باشد، مقدار کلاس تمامی پیش‌بینی‌ها به ۰ تنظیم می‌شود. این بدان معناست که همه پیش‌بینی‌ها به عنوان یک کلاس در نظر گرفته می‌شوند.

ایجاد نسخه‌ای از پیش‌بینی‌ها:

نسخه‌ای از پیش‌بینی‌ها با استفاده از تابع ایجاد می‌شود تا تغییرات بر روی یک نسخه مستقل از پیش‌بینی‌ها اعمال شود.

مقیاس‌دهی جعبه‌های پیش‌بینی شده به فضای اصلی تصویر:

با استفاده از تابع `scale-boxes`، مختصات جعبه‌های پیش‌بینی شده از ابعاد پردازش تصویر به ابعاد فضای اصلی تصویر مقیاس‌دهی می‌شوند. این کار با توجه به ابعاد تصویر اصلی و نسبت تغییر اندازه انجام می‌شود.

به طور خلاصه، این کد پیش‌بینی‌ها را برای مدل‌های تک‌کلاسی تنظیم کرده، مختصات جعبه‌های پیش‌بینی شده را به فضای اصلی تصویر مقیاس‌دهی می‌کند تا برای ارزیابی آماده شوند.

Evaluate:

if nl:

```
tbox = xywh2xyxy(labels[:, 1:5]) target boxes

scale_boxes(im[si].shape[1:], tbox, shape, shapes[si][1]) native-space labels

labelsn = torch.cat((labels[:, 0:1], tbox), 1) native-space labels

correct = process_batch(predn, labelsn, iouv) if plots:
    confusion_matrix.process_batch(predn, labelsn)

stats.append((correct, pred[:, 4], pred[:, 5], labels[:, 0])) (correct, conf, pcls, tcls)
```

این بخش مربوط به ارزیابی مدل است. این کد پیش‌بینی‌های مدل را با برچسب‌های واقعی مقایسه کرده و اطلاعات مربوط به دقت را جمع‌آوری می‌کند به طور خلاصه:

تبدیل و مقیاس‌دهی جعبه‌های هدف:

اگر برچسب‌هایی وجود داشته باشد (nl) :

جعبه‌های هدف از قالب *xywh* به *xyxy* تبدیل می‌شوند. سپس این جعبه‌ها با استفاده از *scale_boxes* به فضای اصلی تصویر مقیاس‌دهی می‌شوند تا با مختصات اصلی تصویر هماهنگ باشند. برچسب‌های نهایی (labelsn) شامل کلاس و جعبه‌های مقیاس‌دهی شده تشکیل می‌شوند.

ارزیابی دقت پیش‌بینی‌ها: تابع *process_batch* برای مقایسه جعبه‌های پیش‌بینی شده (predn) با برچسب‌های نهایی (labelsn) استفاده می‌شود. نتیجه، یک آرایه بولی (correct) است که مشخص می‌کند کدام پیش‌بینی‌ها صحیح هستند. در صورت فعال بودن گزینه رسم نمودارها (plots)، ماتریس سردرگمی با استفاده از *process_batch* به روزرسانی می‌شود.

ذخیره آمار:

نتایج ارزیابی شامل وضعیت پیش‌بینی‌های صحیح (*correct*)، اطمینان پیش‌بینی‌ها (*pred[:, 4]*)، کلاس پیش‌بینی شده (*pred[:, 5]*)، و کلاس واقعی (*labels[:, 0]*) در *stats* ذخیره می‌شود.

Save/log:

if save-txt:

```
(save-dir / "labels").mkdir(parents=True, exist_ok=True)
save-one-txt(predn, save-conf, shape, file=save-dir / "labels" / f"path.stem.txt")
```

if save-json:

```
save-one-json(predn, jdict, path, class-map) append to COCO-JSON dictionary
callbacks.run("on-val-image-end", pred, predn, path, names, im[si])
```

این بخش مربوط به ذخیره‌سازی و لاغرگذاری نتایج ارزیابی است. به طور خلاصه:

ذخیره نتایج به صورت متنی:

اگر گزینه ذخیره فایل‌های متنی فعال باشد (*save-txt*)، دایرکتوری برای ذخیره برچسب‌ها ایجاد می‌شود و پیش‌بینی‌ها به یک فایل متنی ذخیره می‌شوند.

ذخیره نتایج به فرمت JSON :

اگر گزینه ذخیره JSON فعال باشد (*save-json*)، پیش‌بینی‌ها به یک دیکشنری JSON برای فرمت COCO اضافه می‌شوند.

فرآخوانی توابع کالبک:

در پایان پردازش هر تصویر، یک کالبک (*on-val-image-end*) اجرا می‌شود که می‌تواند برای انجام عملیات اضافی در انتهای ارزیابی هر تصویر استفاده شود. به طور کلی، این بخش نتایج ارزیابی را به فرمت‌های مختلف ذخیره کرده و عملیات لاغرگذاری را انجام می‌دهد.

Plot images:

if plots and batch-i < 3:

```
plot-images(im, targets, paths, save-dir / f"val-batchbatch-i-labels.jpg", names) labels
```

```
plot-images(im, output-to-target(preds), paths, save-dir / f"val-batchbatch-i-pred.jpg",
names)
```

```
callbacks.run("on-val-batch-end", batch-i, im, targets, paths, preds)
```

این بخش به رسم تصاویر و اجرای کالبک‌ها در طول فرآیند ارزیابی می‌پردازد. به طور خلاصه:

رسم تصاویر:

اگر گزینه رسم نمودارها فعال باشد (plots) و این دسته (batch) یکی از سه دسته اول باشد، تصاویر همراه با برچسب‌ها و پیش‌بینی‌های مدل رسم و ذخیره می‌شوند: plot-images برای رسم برچسب‌های واقعی تصاویر استفاده می‌شود. همچنین برای رسم پیش‌بینی‌های مدل به کار می‌رود.

اجرای توابع کالبک:

در پایان هر دسته (batch)، کالبک می‌تواند برای انجام on-val-batch-end اجرا می‌شود. این کالبک می‌تواند برای انجام کارهای اضافی مثل لاغندازی، محاسبات خاص یا مدیریت منابع در پایان پردازش هر دسته استفاده شود.

Compute metrics:

```
stats = [torch.cat(x, 0).cpu().numpy() for x in zip(*stats)] to numpy
```

if len(stats) and stats[0].any():

```
tp, fp, p, r, f1, ap, ap-class = ap-per-class(*stats, plot=plots, save-dir=save-dir,
names=names)
```

ap50, ap = ap[:, 0], ap.mean(1) **AP@0.5, AP@0.5:0.95**

```
mp, mr, map50, map = p.mean(), r.mean(), ap50.mean(), ap.mean()
```

```
nt = np.bincount(stats[3].astype(int), minlength=nc) number of targets per class
```

این بخش محاسبات نهایی مربوط به ارزیابی مدل را انجام می‌دهد. به طور خلاصه:

تبدیل آمار به numpy :

نتایج آماری جمع‌آوری شده (مانند تعداد پیش‌بینی‌های صحیح و نادرست) از فرمت PyTorch به numpy تبدیل می‌شوند.

محاسبه معیارهای ارزیابی:

اگر داده‌های آماری موجود باشند و حداقل یک مورد درست وجود داشته باشد، معیارهای مختلف مانند دقت (Precision)، بازخوانی (Recall)، و میانگین دقت (AP) برای هر کلاس محاسبه می‌شوند.

AP برای آستانه‌های IOU مختلف محاسبه می‌شود، از جمله AP@0.5:0.95 و AP@0.5 میانگین دقت (mAP) و میانگین بازخوانی (mR) برای همه کلاس‌ها محاسبه می‌شود.

محاسبه تعداد اهداف:

تعداد اهداف (یا برچسب‌های واقعی) برای هر کلاس با استفاده از np.bincount شمارش می‌شود. به طور کلی، این بخش معیارهای ارزیابی نهایی را محاسبه کرده و خلاصه‌ای از عملکرد مدل بر اساس این معیارها ارائه می‌دهد.

Print results:

```
pf = "%22s" + "%11i" * 2 + "%.11g" * 4 print format
```

```
LOGGER.info(pf % ("all", seen, nt.sum(), mp, mr, map50, map))
```

```
if nt.sum() == 0:
```

```
LOGGER.warning(f'WARNING no labels found in task set, can not compute metrics  
without labels")
```

این بخش وظیفه چاپ نتایج ارزیابی مدل را بر عهده دارد. به طور خلاصه:

فرمت چاپ:

قالبی برای چاپ نتایج با استفاده از رشته قالب `pf` تعریف می شود که شامل تعداد تصاویر دیده شده، تعداد کل اهداف (برچسبها) و میانگین دقت و بازخوانی مدل است. چاپ نتایج:

نتایج کلی شامل تعداد تصاویر پردازش شده، تعداد اهداف، میانگین دقت (mp)، میانگین بازخوانی (mr) و mAP@0.5:0.95 با استفاده از `LOGGER.info` چاپ می شود. هشدار در صورت عدم وجود برچسب:

اگر تعداد برچسبها صفر باشد، یک پیام هشدار با استفاده از `LOGGER.warning` چاپ می شود که بیان می کند هیچ برچسبی یافت نشده و در نتیجه نمی توان معیارها را محاسبه کرد.

Print results per class:

```
if(verbose or (nc < 50 and not training)) and nc > 1 and len(stats):
```

```
for i, c in enumerate(ap-class):
```

```
LOGGER.info(pf % (names[c], seen, nt[c], p[i], r[i], ap50[i], ap[i]))
```

این بخش نتایج ارزیابی مدل را برای هر کلاس به صورت جداگانه چاپ می کند. به طور خلاصه:

شرایط چاپ نتایج برای هر کلاس:

نتایج به صورت کلاسی چاپ می شوند اگر:

حالت `verbose` فعال باشد (`verbose=True`)

یا تعداد کلاس ها کمتر از ۵ باشد و مدل در حال آموزش نباشد (`not training`). همچنین باید بیش از یک کلاس و داده های آماری موجود باشند (`len(stats)`) چاپ نتایج برای هر کلاس:

برای هر کلاس (که در `ap-class` موجود است)، اطلاعات مربوط به آن کلاس چاپ می شود. این شامل نام کلاس، تعداد تصاویر پردازش شده، تعداد اهداف برای آن کلاس، دقت (precision)، بازخوانی و AP@0.5:0.95 است.

استفاده از : LOGGER

نتایج با استفاده از LOGGER.info و با قالب تعریف شده در pf چاپ می شوند.

Print speeds:

$t = \text{tuple}(x.t / \text{seen} * 1e3 \text{ for } x \text{ in } dt)$ speeds per image

if not training:

shape = (batch-size, 3, imgsz, imgsz)

LOGGER.info(f'Speed: %.1fms pre-process, %.1fms inference, %.1fms NMS per image at
shape shape' % t)

این بخش زمان های مربوط به پردازش تصویر را محاسبه و چاپ می کند. به طور خلاصه:

محاسبه سرعت ها:

زمان های مختلف برای پردازش هر تصویر (پیش پردازش، استنتاج مدل، و عملیات NMS) از داده های زمانی dt استخراج شده و به میلی ثانیه برای هر تصویر تبدیل می شوند (t) .

چاپ سرعت ها:

اگر مدل در حالت آموزش نباشد، این سرعت ها به همراه ابعاد ورودی تصویر (shape) چاپ می شوند.
زمان ها به ترتیب برای پیش پردازش، استنتاج، و NMS با استفاده از LOGGER.info نمایش داده می شوند.

Plots:

if plots:

confusion-matrix.plot(save-dir=save-dir, names=list(names.values()))

callbacks.run("on-val-end", nt, tp, fp, p, r, f1, ap, ap50, ap-class, confusion-matrix)

این بخش وظیفه ترسیم نمودارها و اجرای اقدامات پایانی پس از ارزیابی را بر عهده دارد. به طور خلاصه:

ترسیم نمودارها:

اگر گزینه plots فعال باشد، نمودار ماتریس سردرگمی (Confusion Matrix) با استفاده از داده های موجود رسم می شود و در مسیر save-dir ذخیره می شود. این ماتریس عملکرد مدل در طبقه بندی کلاس ها را نشان می دهد.

اجرای کالبک ها:

پس از ترسیم نمودار، یک کالبک با نام on-val-end فراخوانی می شود. این کالبک اطلاعات نهایی مانند تعداد برچسب ها (nt) ، تعداد پیش بینی های صحیح (tp) ، تعداد پیش بینی های نادرست (fp) ، دقت (p) ، بازخوانی (r) ، مقدار AP ، مقدار $1F$ ، مقدار $AP@0.5$ ، کلاس های AP و ماتریس سردرگمی را دریافت می کند.

Save JSON:

```

if save-json and len(jdict):
    w = Path(weights[0]) if isinstance(weights, list) else weights.stem if weights is not None
    else """weights
    anno-json = str(Path("../datasets/coco/annotations/instances-val2017.json"))
    annotations
    if not os.path.exists(anno-json):
        anno-json = os.path.join(data["path"], "annotations", "instances-val2017.json")
    pred-json = str(save-dir / f"w-predictions.json") predictions
    LOGGER.info(f" nEvaluating pycocotools mAP... saving pred-json...")
    with open(pred-json, "w") as f:
        json.dump(jdict, f)

```

try:

<https://github.com/cocodataset/cocoapi/blob/master/PythonAPI/pycocoEvalDemo.ipynb>

```
check-requirements("pycocotools>=2.0.6")
```

```
from pycocotools.coco import COCO
```

```
from pycocotools.cocoeval import COCOeval
```

```
anno = COCO(anno-json) init annotations api
```

```
pred = anno.loadRes(pred-json) init predictions api
```

```
eval = COCOeval(anno, pred, "bbox")
```

```
if is-coco:
```

```
eval.params.imgIds = [int(Path(x).stem) for x in dataloader.dataset.im-files] image IDs to
evaluate
```

```
eval.evaluate()
```

```
eval.accumulate()
```

```
eval.summarize()
```

```
map, map50 = eval.stats[:2] update results (mAP@0.5:0.95, mAP@0.5)
```

```
except Exception as e:
```

```
LOGGER.info(f'pycocotools unable to run: {e}')
```

این بخش وظیفه ذخیره‌سازی نتایج پیش‌بینی مدل به صورت فایل JSON و ارزیابی عملکرد آن با استفاده از ابزارهای pycocotools را بر عهده دارد. به طور خلاصه:

ذخیره‌سازی : JSON

اگر save-json فعال باشد و داده‌های پیش‌بینی (jdict) موجود باشند، نتایج در یک فایل JSON ذخیره می‌شوند. نام این فایل بر اساس نام وزن‌های مدل (weights) تعیین می‌شود. آدرس فایل JSON

حاوی اطلاعات برچسب ها (annotations) از مسیر پیش فرض یا مسیر موجود در داده های ورودی تنظیم می شود.

ارزیابی با pycocotools :

اگر ابزار pycocotools در محیط نصب باشد، از آن برای ارزیابی نتایج پیش بینی شده استفاده می شود. ابتدا COCO برای برچسب های داده شده و loadRes برای بارگذاری پیش بینی ها مقداردهی می شود. سپس COCOeval برای ارزیابی جعبه های پیش بینی شده (bounding boxes) استفاده می شود. اگر داده ها مربوط به مجموعه COCO باشند، ارزیابی فقط برای های *ID* مشخص شده تصاویر انجام می شود. نتایج ارزیابی شامل مقادیر *mAP* و *mAP@0.5* جمع آوری و خلاصه سازی می شود.

مدیریت خطای:

در صورت وقوع خطای پیامی برای اطلاع از ناتوانی در اجرای pycocotools نمایش داده می شود.

Return results:

`model.float() for training`

`if not training:`

```
s = f' nlen(list(save-dir.glob('labels/*.txt'))) labels saved to save-dir / 'labels'" if save-txt
```

```
else """
```

```
LOGGER.info(f'Results saved to colorstr('bold', save-dir)s")
```

```
maps = np.zeros(nc) + map
```

```
for i, c in enumerate(ap-class):
```

```
maps[c] = ap[i]
```

```
return (mp, mr, map50, map, *(loss.cpu() / len(dataloader)).tolist()), maps, t
```

این بخش مربوط به پایان فرآیند ارزیابی و بازگشت نتایج در اسکریپت *val.py* است. به طور خلاصه:

بازگشت مدل به حالت `:float()`:

اگر مدل در حالت half precision باشد، برای اطمینان از اینکه می‌تواند دوباره آموزش ببیند، به حالت `float()` تغییر داده می‌شود. ثبت نتایج:

اگر اسکریپت در حالت آموزش نباشد، تعداد برجسب‌های ذخیره شده را در پوشه `labels` چاپ می‌کند.
همچنین، مسیر پوشه‌ای که نتایج در آن ذخیره شده است، نمایش داده می‌شود.

محاسبه مقادیر *mAP* برای همه کلاس‌ها:

یک آرایه از مقادیر *mAP* برای تمامی کلاس‌ها ایجاد می‌شود. ابتدا تمام مقادیر برابر با *map* تنظیم می‌شوند. سپس برای هر کلاس که در ارزیابی استفاده شده است (ap-class)، مقدار دقیق *AP* محاسبه شده جایگزین می‌شود. بازگرداندن نتایج:

در نهایت، مقادیر مختلف از جمله میانگین دقت، *(mp)* میانگین بازخوانی، *(mr)*، *mAP@0.5*، *mAP@0.5:0.95* می‌شوند.

پیوست ب

بررسی فایل *train.py*

در این پیوست با بخش های مهم فایل *train.py* آشنا می شویم.

ب-۱ کتابخانه ها

```
import argparse
import math
import os
import random
import subprocess
import sys
import time
from copy import deepcopy
from datetime import datetime, timedelta
from pathlib import Path
```

ب-۱. کتابخانه ها

argparse:

این کتابخانه به شما اجازه می دهد تا به راحتی آرگومان های خط فرمان را مدیریت و پارس کنید. با استفاده از argparse، می توانید به سادگی مشخص کنید که اسکریپت شما چه ورودی هایی باید دریافت کند و چگونه باید آنها را پردازش کند.

math:

کتابخانه ای برای انجام محاسبات ریاضی پایه و پیشرفته. شامل توابعی مانند sqrt (جذر)، cos، sin، tan، تابع لگاریتمی و نمایی و همچنین ثابت های ریاضی مانند pi.

os:

این کتابخانه برای انجام عملیات سیستم عاملی استفاده می شود. با استفاده از os می توانید به فایل ها و دایرکتوری ها دسترسی داشته باشید، مسیر های فایل ها را مدیریت کنید، محیط های کاری را تغییر دهید و به متغیر های محیطی دسترسی داشته باشید.

random:

این کتابخانه برای تولید اعداد تصادفی و انتخاب های تصادفی استفاده می شود. می توانید از آن برای تولید اعداد تصادفی، انتخاب یک آیتم تصادفی از یک لیست، یا شافل کردن لیست ها استفاده کنید.

subprocess:

این کتابخانه برای اجرا و مدیریت پروسه های سیستم عاملی در داخل اسکریپت پایتون استفاده می شود. با استفاده از subprocess می توانید دستورات سیستم را اجرا کنید، خروجی آنها را دریافت کنید، و یا یک پروسه جدید را در پس زمینه اجرا کنید.

sys:

کتابخانه‌ای که دسترسی به برخی از پارامترها و توابع سطح پایین مرتبط با سیستم‌عامل را فراهم می‌کند. sys برای مدیریت آرگومان‌های خط فرمان، خروج از برنامه، مدیریت مسیر ماژول‌ها و موارد دیگر استفاده می‌شود.

time:

این کتابخانه توابعی برای مدیریت و اندازه‌گیری زمان ارائه می‌دهد. می‌توانید از time برای اندازه‌گیری مدت زمان اجرای یک عملیات، تاخیر ایجاد کردن، یا دریافت زمان فعلی استفاده کنید.

copy (deepcopy):

کتابخانه copy برای کپی‌برداری از اشیا استفاده می‌شود. deepcopy یک کپی کامل از یک شیء (حتی اشیای تو در تو) ایجاد می‌کند که مستقل از نسخه اصلی است.

datetime:

این کتابخانه برای کار با تاریخ و زمان استفاده می‌شود. datetime شامل توابعی برای ایجاد، تغییر، و محاسبات مربوط به تاریخ‌ها و زمان‌ها است. همچنین می‌توان از آن برای کار با بازه‌های زمانی (timedelta) استفاده کرد.

pathlib:

یک کتابخانه مدرن برای مدیریت مسیرها و فایل‌ها است. این کتابخانه کار با مسیرهای فایل را بسیار ساده‌تر و شهودی‌تر از کتابخانه قدیمی os.path کلاس اصلی این کتابخانه است که می‌توان با آن مسیرهای فایل‌ها و دایرکتوری‌ها را مدیریت کرد.

ب-۲ بررسی بخش های مهم train.py

Directories:

```
w = save-dir / "weights"
```

```
(w.parent if evolve else w).mkdir(parents=True, exist_ok=True)
```

```
last, best = w / "last.pt", w / "best.pt"
```

این کد یک پوشه برای ذخیره وزن‌ها (weights) ایجاد می‌کند و دو فایل last.pt و best.pt را مشخص می‌کند که برای ذخیره آخرین وزن‌ها و بهترین وزن‌های مدل در طول فرآیند آموزش استفاده می‌شود.

Hyperparameters:

```
if isinstance(hyp, str):
```

```
with open(hyp, errors="ignore") as f:
```

```
hyp = yaml.safe_load(f)
```

```
LOGGER.info(colorstr("hyperparameters: ") + ", ".join(f'{k}={v}' for k, v in hyp.items()))
```

```
opt.hyp = hyp.copy()
```

این کد های پارامترها را از یک فایل YAML باگذاری می‌کند (در صورتی که hyp یک مسیر فایل باشد)، آنها را در لگ نمایش می‌دهد و سپس آنها را برای استفاده‌های بعدی و ذخیره در نقاط بررسی (checkpoints) ذخیره می‌کند. این یک مرحله کلیدی در تنظیم و مدیریت فرآیند آموزش مدل است.

Save run settings:

if not evolve:

```
yaml-save(save-dir / "hyp.yaml", hyp)
te yaml-save(save-dir / "opt.yaml", vars(opt))xt
```

این کد وقتی که در حالت **تکامل**^۱ مدل نیستیم، های پارامترها (*hyp*) و گرینهای پیکربندی (*opt*) را در فایل‌های *opt.yaml* و *hyp.yaml* ذخیره می‌کند. این فایل‌ها برای مستندسازی و بازیابی تنظیمات استفاده می‌شوند و معمولاً برای باز تولید نتایج یا ادامه آموزش مدل در مراحل بعدی کاربرد دارند.

Loggers:

```
data-dict = None
if RANK in -1, 0:
    include-loggers = list(LOGGERS)
    if getattr(opt, "ndjson-console", False):
        include-loggers.append("ndjson-console")
    if getattr(opt, "ndjson-file", False):
        include-loggers.append("ndjson-file")
    loggers = Loggers(
        save-dir=save-dir,
        weights=weights,
        opt=opt,
        hyp=hyp,
        logger=LOGGER,
        include=tuple(include-loggers),)
```

^۱evolve

این بخش از کد مربوط به تنظیم و مقداردهی اولیه ابزارهای لاغرگیری (loggers) در فرآیند آموزش مدل است. لاغرگیرها برای ثبت وقایع، نتایج، هایپرپارامترها، متريک‌ها و دیگر اطلاعات حیاتی در طول آموزش مدل استفاده می‌شوند. اين کد ابزارهای لاغرگیری مختلفی را براساس تنظیمات موجود در opt و LOGGERS راهاندازی می‌کند. اگر فرآیند فعلی به عنوان فرآیند اصلی یا غیرتوزیعی شناخته شود (با توجه به مقدار RANK)، لاغرها مربوطه فعال می‌شوند. این لاغرها ممکن است شامل لاغرها برای کنسول، فایل‌ها، و حتی فرمتهای خاص مانند NDJSON باشند.

Register actions:

for k in methods(loggers):

```
callbacks.register-action(k, callback=getattr(loggers, k))
```

این کد، متدهای موجود در شیء loggers را به عنوان "اعمال" در سیستم "بازخوانی" ثبت می‌کند. به این ترتیب، هر متدهای مربوط به عنوان یک بازخوانی برای یک عمل خاص ثبت می‌شود. این کار باعث می‌شود که متدهای مربوط به لاغرها در زمان مناسب، مانند پایان هر دوره آموزشی یا ثبت نتایج، به صورت خودکار اجرا شوند، بدون اینکه نیاز به فراخوانی دستی آن‌ها باشد.

Process custom dataset artifact link :

```
data-dict = loggers.remote-dataset
```

```
if resume:
```

```
weights, epochs, hyp, batch-size = opt.weights, opt.epochs, opt.hyp, opt.batch-size
```

این کد برای پردازش داده‌های مربوط به مجموعه داده‌های ذخیره شده از منابع خارجی استفاده می‌شود. اگر آموزش باید از نقطه‌ای قبلی ادامه یابد، مقادیر مربوط به وزن‌ها، تعداد epoch‌ها، هایپرپارامترها و اندازه دسته‌بندی از تنظیمات قبلی بارگذاری می‌شود تا فرآیند آموزش از همان نقطه قبلی ادامه یابد.

Config :

```

plots = not evolve and not opt.noplots

cuda = device.type != "cpu"

init-seeds(opt.seed + 1 + RANK, deterministic=True)

with torch-distributed-zero-first(LOCAL-RANK):

    data-dict = data-dict or check-dataset(data)

    train-path, val-path = data-dict["train"], data-dict["val"]

    nc = 1 if single-cls else int(data-dict["nc"])

    names = 0: "item" if single-cls and len(data-dict["names"]) != 1 else data-dict["names"]

    is-coco = isinstance(val-path, str) and val-path.endswith("coco/val2017.txt")

```

این کد شامل چندین مرحله کلیدی برای آماده سازی داده ها و تنظیمات آموزش مدل است. ابتدا تصمیم می گیرد که آیا نمودارها باید ایجاد شوند یا خیر، سپس دستگاه پردازش را تنظیم می کند و بندر تصادفی Random Seed را برای تکرار پذیری تنظیم می کند. در ادامه، با استفاده از محیط توزیع شده، داده ها بررسی و بارگذاری می شود و مسیر های داده و تعداد کلاس ها تنظیم می شود. همچنین، نام های کلاس ها بر اساس شرایط خاص تنظیم می شود و در نهایت، بررسی می شود که آیا مجموعه داده استفاده شده از نوع COCO است یا خیر. این مراحل به منظور اطمینان از آماده بودن تمام جنبه های آموزش مدل و صحت داده ها انجام می شود.

Model:

```

check-suffix(weights, ".pt")

pretrained = weights.endswith(".pt")

if pretrained:

```

with torch-distributed-zero-first(LOCAL-RANK):

```
weights = attempt-download(weights)

ckpt = torch.load(weights, map-location="cpu")

model = Model(cfg or ckpt["model"].yaml, ch=3, nc=nc,
anchors=hyp.get("anchors")).to(device)

exclude = ["anchor"] if (cfg or hyp.get("anchors")) and not resume else []

csd = ckpt["model"].float().state-dict()

csd = intersect-dicts(csd, model.state-dict(), exclude=exclude)

model.load-state-dict(csd, strict=False)

LOGGER.info(f"Transferred len(csd)/len(model.state-dict()) items from weights")
```

else:

```
model = Model(cfg, ch=3, nc=nc, anchors=hyp.get("anchors")).to(device)

amp = check-amp(model)
```

این بخش از کد مربوط به بررسی و باگذاری مدل است. کد بررسی می‌کند که آیا مدل باید از وزن‌های پیش‌آموزش دیده (pretrained) استفاده کند یا خیر، و سپس مدل را باگذاری یا ایجاد می‌کند. همچنین، برخی از تنظیمات مانند استفاده از دقت مخلوط (AMP) را نیز پیکربندی می‌کند.

Freeze:

```
freeze = [f'model.{x}.' for x in (freeze if len(freeze) > 1 else range(freeze[0]))]
```

for k, v in model.named-parameters():

v.requires-grad = True

if any(x in k for x in freeze):

```
LOGGER.info(f'freezing k")
```

```
v.requires_grad = False
```

این بخش از کد مسئول تنظیم لایه‌های مدل برای فرآیند آموزش است، به ویژه در مورد "فریز کردن" برخی از لایه‌ها. ابتدا لیست پیشوندهای لایه‌هایی که باید فریز شوند، ایجاد می‌شود. سپس برای هر پارامتر مدل، بررسی می‌شود که آیا نام آن پارامتر شامل پیشوندهای فریز شده است یا خیر. اگر پارامتر باید فریز شود، گرادیان آن غیرفعال می‌شود و از آموزش آن جلوگیری می‌شود. این کار به مدل اجازه می‌دهد تا تنها روی بخش‌های خاصی از آن که نیاز به آموزش دارند تمرکز کند و از تغییرات غیرضروری در بخش‌های دیگر جلوگیری نماید. اگر بخواهیم تنها لایه‌های خاصی با نام‌های خاص را فریز کنیم، می‌توانیم لیستی از نام‌های دقیق لایه‌ها را به کد اضافه کنیم یا اگر بخواهیم تنها لایه‌های موجود در عمق خاصی از مدل را فریز کنیم، می‌توانیم عمق لایه‌ها را بررسی و از آن‌ها استفاده کنیم.

Image size:

```
gs = max(int(model.stride.max()), 32)
```

```
imgsz = check-img-size(opt.imgsz, gs, floor=gs * 2)
```

این بخش از کد اطمینان می‌یابد که اندازه تصویر ورودی با نیازهای مدل مطابقت دارد. ابتدا اندازه شبکه تعیین می‌شود که از بزرگ‌ترین اندازه‌های گام مدل و حداقل مقدار ۳۲ استفاده می‌کند. سپس اندازه تصویر ورودی بررسی می‌شود تا مطمئن شود که با اندازه شبکه سازگار است و از مقادیر نادرست جلوگیری می‌شود.

Optimizer :

```
nbs = 64
```

```
accumulate = max(round(nbs / batch-size), 1)
```

```
hyp[”weight-decay”] *= batch-size * accumulate / nbs
```

```
optimizer = smart-optimizer(model, opt.optimizer, hyp[“lr0”], hyp[“momentum”],
                            hyp[“weight-decay”])
```

این بخش از کد به تنظیمات مربوط به بهینه‌ساز برای آموزش مدل یادگیری عمیق اختصاص دارد. ابتدا اندازه دسته نامی تعیین می‌شود و سپس تعداد تجمعات گرادیان محاسبه می‌شود تا از حافظه بهینه استفاده شود. مقدار وزن کاهشی بر اساس اندازه دسته واقعی و تعداد تجمعات مقیاس دهی می‌شود. در نهایت، با استفاده ازتابع *smart-optimizer* بهینه‌ساز با پارامترهای مناسب ایجاد می‌شود تا فرآیند آموزش مدل بهینه‌تر و مؤثرتر انجام شود.

Scheduler :

```
if opt.cos-lr:
    lf = one-cycle(1, hyp[‘lrf’], epochs)
    cosine 1->hyp[‘lrf’]
else:
    def lf(x):
        return (1 - x / epochs) * (1.0 - hyp[“lrf”]) + hyp[“lrf”]
scheduler = lr-scheduler.LambdaLR(optimizer, lr-lambda=lf)
```

این بخش از کد به تنظیم زمان‌بند نرخ یادگیری برای آموزش مدل مربوط می‌شود. اگر *گزینه-*cos** یادگیری باشد، نرخ یادگیری با استفاده از یک تابع کازین تغییر می‌کند، که نرخ یادگیری را از مقدار اولیه به مقدار نهایی در طول دوران آموزش کاهش می‌دهد. در غیر این صورت، نرخ یادگیری به صورت خطی کاهش می‌یابد. در نهایت، یک زمان‌بند نرخ یادگیری با استفاده از تابع *LambdaLR* ایجاد می‌شود که نرخ یادگیری را به صورت دینامیک و با توجه به تابع *If* تنظیم می‌کند.

EMA :

```
ema = ModelEMA(model) if RANK in -1, 0 else None
```

این بخش از کد برای پیاده‌سازی میانگین نمایی متغیر (EMA) در مدل یادگیری عمیق استفاده می‌شود. EMA به حفظ یک نسخه صاف‌تر و پایدارتر از وزن‌های مدل در طول آموزش کمک می‌کند. این نسخه‌ی آرام‌تر از مدل، اغلب برای ارزیابی یا ذخیره‌سازی نهایی استفاده می‌شود تا از نوسانات لحظه‌ای جلوگیری شود. شرط RANK تضمین می‌کند که EMA فقط در فرآیند اصلی یا زمانی که مدل به صورت توزیع‌نشده اجرا می‌شود، ایجاد شود.

Resume :

```
best-fitness, start-epoch = 0.0, 0
```

```
if pretrained:
```

```
if resume:
```

```
best-fitness, start-epoch, epochs = smart-resume(ckpt, optimizer, ema, weights, epochs,
resume)
```

```
del ckpt, csd
```

این بخش از کد مسئول از سرگیری آموزش مدل از یک نقطه‌ی ذخیره‌شده قبلی است. اگر مدل با وزن‌های از پیش آموزش دیده شده بارگذاری شده و گزینه‌ی resume فعال باشد، وضعیت قبلی آموزش (از جمله بهترین عملکرد، شماره دوره، و وزن‌ها) بارگذاری می‌شود و آموزش از همان نقطه ادامه می‌یابد. این کار باعث می‌شود که نیازی به شروع مجلد آموزش از ابتدا نباشد.

DP mode :

```
if cuda and RANK == -1 and torch.cuda.device_count() > 1:
```

```
LOGGER.warning(
```

”WARNING DP not recommended, use torch.distributed.run for best DDP Multi-GPU results. n” ”See Multi-GPU Tutorial at

<https://docs.ultralytics.com/yolov5/tutorials/multi-gpu-training> to get started.”

)

```
odel = torch.nn.DataParallel(model)
```

این کد حالتی را بررسی می‌کند که در آن از چندین GPU استفاده می‌شود اما DDP فعال نیست. در این حالت، به کاربر هشدار داده می‌شود که استفاده از (DP) توصیه نمی‌شود و به جای آن بهتر است از DDP برای نتایج بهینه استفاده کند. اگر کاربر همچنان از DP استفاده کند، مدل به صورت موازی در چندین GPU اجرا خواهد شد، هرچند این روش ممکن است کندتر و کمتر بهینه باشد.

SyncBatchNorm :

```
if opt.sync-bn and cuda and RANK != -1:
```

```
model = torch.nn.SyncBatchNorm.convert_sync_batchnorm(model).to(device)
```

```
LOGGER.info("Using SyncBatchNorm()")
```

این کد زمانی که آموزش مدل به صورت توزیع شده روی چندین GPU انجام می‌شود و گزینه sync-bn فعال است، از SyncBatchNorm استفاده می‌کند. این تکنیک، Batch-Normalization را در بین GPU ها همگام سازی می‌کند تا نتایج پایدارتری حاصل شود.

Trainloader :

```
train-loader, dataset = create-dataloader(
```

```
train-path,
```

```
imgsz,
```

```

batch-size // WORLD-SIZE,
gs,
single-cls,
hyp=hyp,
augment=True,
cache=None if opt.cache == "val" else opt.cache,
rect=opt.rect,
rank=LOCAL-RANK,
workers=workers,
image-weights=opt.image-weights,
quad=opt.quad,
prefix=colorstr("train: "),
shuffle=True,
seed=opt.seed,
)
labels = np.concatenate(dataset.labels, 0)
mlc = int(labels[:, 0].max())
assert mlc < nc, f"Label class mlc exceeds nc={nc} in data. Possible class labels are 0-nc - 1"

```

این کد با استفاده از تابع `create-dataloader` یک (DataLoader) برای آموزش مدل ایجاد می‌کند. این داده‌بار تصاویر و برچسب‌های مربوطه را به دسته‌های مختلف تقسیم می‌کند و به مدل تحویل می‌دهد. تنظیمات مختلفی مانند اندازه تصویر، تقویت داده‌ها و غیره در ایجاد داده‌بار لحاظ می‌شود.

همچنین، برچسب های داده ها بررسی می شوند تا اطمینان حاصل شود که تعداد کلاس های واقعی با تعداد کلاس های تعریف شده مطابقت دارد.

Process 0 :

```
if RANK in -1, 0:
    val-loader = create-dataloader(
        val-path,
        imgsz,
        batch-size // WORLD-SIZE * 2,
        gs,
        single-cls,
        hyp=hyp,
        cache=None if noval else opt.cache,
        rect=True,
        rank=-1,
        workers=workers * 2,
        pad=0.5,
        prefix=colorstr("val: "),
    )[0]
    if not resume:
        if not opt.noautoanchor:
            check-anchors(dataset, model=model, thr=hyp["anchor-t"], imgsz=imgsz)
```

AutoAnchor

```
model.half().float()
callbacks.run("on-pretrain-routine-end", labels, names)
```

این بخش از کد داده‌بار (DataLoader) را برای اعتبارسنجی ایجاد می‌کند و در صورت نیاز، Anchor های مدل را بهینه‌سازی می‌کند. همچنین، اگر مدل از یک نقطه شروع مجلد نشده باشد، دقت محاسبات Anchor ها کاهش داده می‌شود. در نهایت، یک کالبک برای پایان تنظیمات پیش‌آموزش فراخوانی می‌شود. این تنظیمات برای اطمینان از عملکرد بهینه مدل در طول آموزش و اعتبارسنجی انجام می‌شود.

DDP mode :

```
if cuda and RANK != -1:
    model = smart-DDP(model)
```

این کد بررسی می‌کند که اگر CUDA فعال باشد و مدل به صورت توزیع شده اجرا شود، مدل به صورت موازی روی چندین GPU اجرا می‌شود. تابع smart-DDP مدل را در قالب Distributed Data Parallel قرار می‌دهد تا از مزایای موازی‌سازی در های GPU بهره‌برداری شود. این کار باعث افزایش کارایی و کاهش زمان آموزش مدل می‌شود.

Model attributes :

```
nl = de-parallel(model).model[-1].nl
hyp["box"] *= 3 / nl
hyp["cls"] *= nc / 80 * 3 / nl
hyp["obj"] *= (imgsz / 640) ** 2 * 3 / nl
```

```
hyp[“label-smoothing”] = opt.label-smoothing
```

```
model.nc = nc
```

```
model.hyp = hyp
```

```
model.class-weights = labels-to-class-weights(dataset.labels, nc).to(device) * nc
```

```
model.names = names
```

این تنظیمات باعث می شوند که مدل بتواند به درستی با داده های آموزشی و معماری خود هماهنگ شود. به عبارت دیگر، این کد به مدل کمک می کند تا با توجه به تعداد کلاس ها، لایه ها و اندازه تصویر، هایپر امترهای مناسب را دریافت کند و آمادگی لازم برای شروع آموزش را داشته باشد.

* Start training :

```
t0 = time.time() nb = len(train-loader) nw = max(round(hyp[“warmup-epochs”] * nb), 100)
```

```
iterations, max(3 epochs, 100 iterations) * nw = min(nw, (epochs - start-epoch) / 2 * nb) *
```

```
limit warmup to < 1/2 of training
```

```
last-opt-step = -1
```

```
maps = np.zeros(nc) * mAP per class
```

```
results = (0, 0, 0, 0, 0, 0, 0) * P, R, mAP @.5, mAP @.5-.95, val-loss(box, obj, cls)
```

```
scheduler.last-epoch = start-epoch - 1
```

```
scaler = torch.cuda.amp.GradScaler(enabled=amp)
```

```
stopper, stop = EarlyStopping(patience=opt.patience), False
```

```
compute-loss = ComputeLoss(model)
```

```
callbacks.run(“on-train-start”)
```

```
LOGGER.info(
```

```
f'Image sizes imgsz train, imgsz val n'
f'Using train-loader.num-workers * WORLD-SIZE dataloader workers n'
f'Logging results to colorstr('bold', save-dir) n"
f'Starting training for epochs epochs...'
```

این بخش از کد مسئول آماده سازی و شروع فرآیند آموزش مدل است. ابتدا زمان شروع آموزش ثبت می شود تا مدت زمان کل فرآیند قابل اندازه گیری باشد. تعداد مینی بچه ها در داده های آموزشی محاسبه شده و تعداد تکرارهای گرم کردن برای جلوگیری از نوسانات شدید در ابتدای آموزش تنظیم می شود.

سپس، برخی متغیرهای اولیه شامل maps برای ذخیره میانگین دقت (mAP) کلاس ها و results برای نگهداری نتایج معیارهای مختلف مثل دقت، بازخوانی و mAP مقداردهی می شوند. scheduler نیز از دوره قبلی شروع می شود تا تغییرات ناگهانی در تنظیمات جلوگیری شود.

در ادامه، برای محاسبات دقیق تر و سریع تر در صورت فعل بودن محاسبات دقت ترکیبی (AMP)، از استفاده می شود. همچنین، مکانیزم توقف زودهنگام Early stopping برای متوقف کردن آموزش در صورت عدم بهبود در عملکرد مدل تنظیم می گردد. سپس، تابعی برای محاسبه خطا (Loss) تعریف می شود تا در طول آموزش مورد استفاده قرار گیرد.

با اجرای کالبک های مرتبط با شروع آموزش و گزارش اطلاعات مربوط به اندازه تصویر، تعداد کارکنان پردازش داده و محل ذخیره نتایج، فرآیند آموزش رسماً آغاز می شود. این کد اطمینان می دهد که همه چیز قبل از شروع آموزش به درستی تنظیم و آماده شده است تا مدل بتواند به صورت بهینه و تحت نظارت دقیق آموزش ببیند.

epoch :

for epoch in range(start-epoch, epochs):

 callbacks.run("on-train-epoch-start")

 model.train()

این کد مسئول شروع هر دوره (epoch) از آموزش مدل است. در هر دوره، ابتدا توابع مربوط به شروع دوره اجرا می شود و سپس مدل به حالت آموزش تغییر می یابد تا ویژگی هایی مانند dropout فعال شوند. این فرآیند برای هر دوره از آموزش به صورت مکرر تکرار می شود.

Update image weights (optional, single-GPU only) :

```
if opt.image-weights:
```

```
    iw = labelsto-image-weights(dataset.labels, nc=nc, class-weights=cw)
```

```
    dataset.indices = random.choices(range(dataset.n), weights=iw, k=dataset.n)
```

این کد وزن های تصاویر را بر اساس عملکرد مدل در شناسایی کلاس ها به روز می کند. اگر گزینه image-weights فعال باشد، این کد وزن های جدیدی برای تصاویر محاسبه می کند تا تصاویری که مدل در شناسایی آن ها ضعیف تر عمل کرده، وزن بیشتری داشته باشند. سپس این وزن ها برای انتخاب تصادفی تصاویر با احتمال های مشخص استفاده می شود تا مدل تمرکز بیشتری بر روی تصاویر دشوار تر داشته باشد.

Warmup :

```
if ni <= nw:
```

```
    xi = [0, nw]
```

```
    accumulate = max(1, np.interp(ni, xi, [1, nbs / batch-size]).round())
```

```
for j, x in enumerate(optimizer.param-groups):
```

```
    x[”lr”] = np.interp(ni, xi, [hyp[”warmup-bias-lr”]] if j == 0 else
```

```
0.0, x[”initial-lr”] * lf(epoch)])
```

```
if ”momentum” in x:
```

```
x[“momentum”] = np.interp(ni, xi, [hyp[“warmup-momentum”],  
hyp[“momentum”]])
```

این بخش از کد مربوط به مرحله **گرم کردن^۲** در فرآیند آموزش مدل است.

در این مرحله:

تغییرات تدریجی: نرخ یادگیری و مومنتوم (momentum) به تدریج در طول مراحل اولیه آموزش تنظیم می‌شوند تا مدل بتواند به آرامی به فرآیند یادگیری وارد شود و از نوسانات بزرگ در ضررها (losses) جلوگیری شود.

محاسبات بینابینی: مقادیر نرخ یادگیری و مومنتوم برای هر گروه از پارامترهای بهینه‌ساز به صورت خطی بین مقادیر اولیه و هدف موردنظر محاسبه می‌شوند.

تنظیم انباشت‌سازی: مقدار انباشت‌سازی ضررها نیز به صورت خطی تغییر می‌کند، که به مدل اجازه می‌دهد تا به تدریج به شرایط بهینه نزدیک شود.

این مرحله باعث می‌شود که مدل در ابتدا با تغییرات کوچک‌تر شروع به یادگیری کند و سپس به مرور با افزایش نرخ یادگیری، آموزش خود را بهینه‌تر کند.

Multi-scale :

```
if opt.multi-scale:
```

```
sz = random.randrange(int(imgsz * 0.5), int(imgsz * 1.5) + gs) // gs * gs size
```

```
sf = sz / max(imgs.shape[2:]) scale factor
```

```
if sf != 1:
```

```
ns = [math.ceil(x * sf / gs) * gs for x in imgs.shape[2:]] new
```

shape (stretched to gs-multiple)

```
imgs = nn.functional.interpolate(imgs, size=ns, mode="bilinear", align_corners=False)
```

²warmup

این بخش از کد مربوط به آموزش با مقیاس چندگانه^۳ است و به شرح زیر عمل می کند:

بررسی فعال بودن مقیاس چندگانه:

ابتدا چک می کند که آیا این گزینه فعال شده است یا خیر.

تعیین اندازه جدید تصویر:

یک اندازه جدید به صورت تصادفی در محدوده ۰٪ تا ۱۵۰٪ اندازه اصلی تصویر ورودی انتخاب می شود. این اندازه به گونه ای تنظیم می شود که با اندازه گام های شبکه^۴ سازگار باشد.

مقیاس بندی تصویر:

در صورتی که اندازه جدید با اندازه قبلی متفاوت باشد، تصویر ورودی با استفاده از روش bilinear interpolation به اندازه جدید تغییر مقیاس داده می شود.

این روش کمک می کند تا مدل به تغییرات مقیاس در تصاویر ورودی حساسیت کمتری داشته باشد و بتواند در شرایط مختلف بهتر عمل کند.

Forward :

with torch.cuda.amp.autocast(amp):

pred = model(imgs) **forward**

loss, loss-items = compute-loss(pred, targets.to(device)) **loss**

scaled by batch-size

if RANK != -1:

loss *= WORLD-SIZE **gradient averaged between devices in DDP mode**

if opt.quad:

loss *= 4.0

³multi-scale training

⁴grid size

این بخش از کد مربوط به انجام عملیات پیش روی **عملیات پیش روی^۵** و به شرح زیر عمل می‌کند:

فعال سازی *AMP* (محاسبات دقیق و سریع):

محاسبات با دقت مختلط انجام می‌شود. این ویژگی باعث افزایش سرعت پردازش و بهینه‌سازی مصرف حافظه *GPU* می‌شود.

اجرای مدل بر روی داده‌ها:

داده‌های ورودی (imgs) از طریق مدل عبور می‌کنند و پیش‌بینی‌ها (pred) تولید می‌شوند. این همان مرحله‌ی پیش روی است که در آن مدل، خروجی‌های خود را بر اساس ورودی‌ها تولید می‌کند.

محاسبه‌ی زیان:

ضرر و مقادیر ضرر با مقایسه‌ی پیش‌بینی‌های مدل و مقادیر هدف محاسبه می‌شود. این زیان معیاری برای اندازه‌گیری عملکرد مدل در هر مرحله است.

مقیاس‌بندی زیان در حالت توزیع شده:

اگر از حالت توزیع شده DDP استفاده شود و RANK برابر یا بیشتر از ۰ باشد، ضرر محاسبه شده برای همه دستگاه‌ها با استفاده از تعداد کل دستگاه‌ها (WORLD-SIZE) مقیاس‌بندی می‌شود.

مقیاس‌بندی زیان در حالت چهارگانه:

در صورتی که گزینه‌ی opt.quad فعال باشد، زیان نهایی چهار برابر می‌شود.

⁵Forward Pass

Backward :

```
scaler.scale(loss).backward()
```

این کد در فرآیند آموزش است که به مدل می‌آموزد چگونه به روزرسانی شود و با هر دوره (epoch) به سمت بهبود عملکرد حرکت کند.

Optimize :

```
if ni - last-opt-step >= accumulate:
```

```
    scaler.unscale_(optimizer) unscale gradients
```

```
    torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=10.0) clip gradients
```

```
    scaler.step(optimizer) optimizer.step
```

```
    scaler.update()
```

```
    optimizer.zero_grad()
```

```
if ema:
```

```
    ema.update(model)
```

```
    last-opt-step = ni
```

این کد مرحله بهینه‌سازی در فرآیند آموزش را انجام می‌دهد که در آن پارامترهای مدل براساس گرادیان‌ها به روزرسانی می‌شوند، گرادیان‌ها برای جلوگیری از ناپایداری محدود می‌شوند، و مقیاس AMP برای مراحل بعدی تنظیم می‌شود.

log :

if RANK in -1, 0:

mloss = (mloss * i + loss-items) / (i + 1) **update mean losses**

mem = f'torch.cuda.memory-reserved() / 1E9 if torch.cuda.is-available() else 0:.3gG"

pbar.set-description(

("%11s" * 2 + "%11.4g" * 5)

% (f'epoch/epochs - 1", mem, *mloss, targets.shape[0], imgs.shape[-1])))

callbacks.run("on-train-batch-end", model, ni, imgs, targets, paths, list(mloss))

if callbacks.stop-training:

return

این کد بخش گزارش دهی و ثبت وضعیت در طی فرآیند آموزش مدل را مدیریت می کند. به طور خلاصه:

به روزرسانی میانگین خطاهای:

خطاهای محاسبه شده در هر مرحله از آموزش با میانگین گیری به روزرسانی می شوند.

محاسبه مصرف حافظه:

میزان حافظه گرافیکی مصرف شده برای آموزش محاسبه و ثبت می شود.

نمایش وضعیت آموزش:

نوار پیشرفت (progressbar) با اطلاعات مربوط به دوره آموزشی، میانگین خطاهای، تعداد نمونه ها، و اندازه تصاویر به روز می شود.

اجرای عملیات های اضافی: با استفاده از کالبک ها، عملیات هایی مانند ذخیره مدل یا بررسی توقف زودهنگام در انتهای هر دسته (batch) انجام می شود.

بررسی توقف: اگر شرایط توقف زودهنگام فراهم شود، آموزش متوقف می شود. این بخش از کد به طور کلی وظیفه نمایش وضعیت و مدیریت عملیات های مرتبط با هر مرحله از آموزش را بر عهده دارد.

mAP:

if RANK in -1, 0:

 callbacks.run("on-train-epoch-end", epoch=epoch)

 ema.update-attr(model, include=["yaml", "nc", "hyp", "names", "stride", "class-weights"])

 final - epoch = (epoch + 1 == epochs) or stopper.possible-stop if not noval or final - epoch:

Calculate mAP

 results, maps, - = validate.run(

 data-dict,

 batch-size=batch-size // WORLD-SIZE * 2,

 imgsz=imgsz,

 half=amp,

 model=ema.ema,

 single-cls=single-cls,

 dataloader=val-loader,

 save-dir=save-dir,

 plots=False,

 callbacks=callbacks,

 compute-loss=compute-loss,

)

این بخش از کد عملکرد و ارزیابی مدل را در پایان هر دوره بررسی می‌کند. به طور خلاصه:

اجرای کالبک‌ها:

در پایان هر دوره، کالبک‌های مربوطه اجرا می‌شوند تا هر عملیات خاصی که نیاز است در این مرحله انجام شود، صورت گیرد.

به روزرسانی ویژگی‌های مدل EMA:

مدل EMA که نسخه پایدارتر از مدل اصلی است، با ویژگی‌های کلیدی مدل اصلی به روزرسانی می‌شود.

بررسی توقف زودهنگام:

اگر دوره پایانی آموزش یا شرایط توقف زودهنگام برقرار باشد، آموزش متوقف می‌شود.

محاسبه mAP:

اگر نیاز به اعتبارسنجدی یا دوره پایانی باشد، مدل اعتبارسنجدی می‌شود تا مقدار mAP (میانگین دقت) محاسبه و ارزیابی شود. این محاسبه با استفاده از داده‌های اعتبارسنجدی انجام می‌شود و نتایج در پایان ذخیره می‌شوند. این بخش کد تصمیم‌گیری می‌کند که در پایان هر دوره، مدل به روزرسانی و ارزیابی شده و در صورت نیاز، آموزش متوقف شود.

Update best mAP:

```
fi = fitness(n p .array(results).reshape(1, - 1))
```

```
combination of [P, R, mAP @.5, mAP @.5-.95]
```

```
stop = stopper(epoch=epoch, fitness=fi) early stop check
```

```
if fi > best-fitness:
```

```
best-fitness = fi
```

```
log-vals = list(mloss) + list(results) + lr
```

```
callbacks.run("on-fit-end", log-vals, epoch, best-fitness, fi)
```

این کد در پایان هر دوره آموزشی وظیفه ارزیابی عملکرد مدل، بهروزرسانی بهترین عملکرد، و بررسی شرایط توقف زودهنگام را دارد. به طور خلاصه:

محاسبه Fitness:

یک معیار ترکیبی از شاخص های عملکرد مدل مانند دقت و mAP محاسبه می شود.

بررسی توقف زودهنگام:

بررسی می شود که آیا مدل به حد مطلوب رسیده است یا نه. در صورت لزوم، آموزش متوقف می شود.

بهروزرسانی بهترین عملکرد:

اگر معیار جدید بهتر از مقدار قبلی باشد، بهترین عملکرد بهروزرسانی می شود.

ثبت نتایج: نتایج و معیارهای عملکرد برای استفاده های بعدی ذخیره می شوند. این کد به بهبود مدل کمک می کند و از ادامه آموزش بی فایده جلوگیری می کند.

Update best mAP:

```
torch.save(ckpt, last)
```

```
if best-fitness == fi:
```

```
    torch.save(ckpt, best)
```

```
if opt.save-period > 0 and epoch % opt.save-period == 0:
```

```
    torch.save(ckpt, w / f'epoch{epoch}.pt")
```

```
del ckpt
```

```
callbacks.run("on-model-save", last, epoch, final-epoch, best-fitness, fi)
```

این بخش از کد وظیفه ذخیره مدل را به عهده دارد و به طور خلاصه:

ذخیره مدل فعلی:

مدل پس از هر دوره آموزشی به عنوان آخرین مدل (last) ذخیره می شود.

ذخیره بهترین مدل:

اگر مدل فعلی بهترین عملکرد را داشته باشد، به عنوان بهترین مدل (best) ذخیره می شود.

ذخیره دوره ای:

اگر گزینه ذخیره دوره ای فعال باشد (opt.save-period) و دوره فعلی با دوره ذخیره تعیین شده هماهنگ باشد، مدل ذخیره می شود.

پاکسازی حافظه:

پس از ذخیره سازی، حافظه از داده های مدل پاک می شود.

اجرای کالبک:

پس از ذخیره، یک کالبک برای اطلاع از این رویداد فراخوانی می شود. این فرآیند تضمین می کند که همیشه آخرین و بهترین نسخه مدل در دسترس باشد و در صورت نیاز دوره های خاصی از مدل ذخیره شوند.

EarlyStopping:

```
if RANK != -1: if DDP training
```

```
broadcast-list = [stop if RANK == 0 else None]
```

```
dist.broadcast-object-list(broadcast-list, 0) broadcast 'stop' to all ranks
```

```
if RANK != 0:
```

```
stop = broadcast-list[0]
```

```
if stop:
```

```
break must break all DDP ranks
```

این بخش از کد مربوط به مکانیزم توقف زودهنگام در شرایط آموزش توزیع شده (DDP) است. این کد به طور خلاصه:

بررسی آموزش توزیع شده:

اگر مدل با استفاده از DDP آموزش داده می شود (یعنی $1 \neq ! - RANK$)، لازم است وضعیت توقف زودهنگام بین تمام پردازش‌ها همگام‌سازی شود.

ارسال وضعیت توقف:

اگر پردازش فعلی رنک ۰ باشد (رنکی که معمولاً به عنوان رنک اصلی در نظر گرفته می شود)، وضعیت stop را به سایر رنک‌ها ارسال می‌کند. در غیر این صورت، پردازش‌های دیگر وضعیت stop را از رنک ۰ دریافت می‌کنند.

توقف زودهنگام:

اگر وضعیت stop فعال شود، حلقه آموزش برای همه پردازش‌ها متوقف می شود.
این مکانیزم اطمینان می دهد که در صورت فعال شدن توقف زودهنگام، تمام پردازش‌ها همزمان متوقف شوند و هیچ کدام به آموزش ادامه ندهند.
و در نهایت عملیات آموزش تمام می شود.

واژه‌نامه انگلیسی به فارسی

bounding box.....	جعبه محدود کننده
you only look once.....	تو فقط یک بار نگاه می کنی
Training.....	آموزش
Validation.....	اعتبار سنجی
Data set.....	مجموعه داده
learning rate.....	نرخ یادگیری
batch size.....	اندازه دسته
evolve.....	تکامل
task.....	وظیفه
warmup.....	گرم کردن
multi scale training.....	آموزش با مقیاس چندگانه
grid size.....	اندازه گام ها

پیوست ب

مراجع

[1] YOLO: You Only Look Once-Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi-2016

[2] Optimal Speed and Accuracy of Object Detection-Alexey Bochkovskiy, Chien-Yao Wang, Hong-Yuan Mark Liao-2020