

 <p>دانشگاه صنعتی خواجه نصیرالدین طوسی دانشکده مهندسی برق - کروه مهندسی کنترل</p>	<p>به نام خدا</p> <p>دانشگاه تهران - دانشگاه صنعتی خواجه نصیرالدین طوسی تهران</p> <p>دانشکده مهندسی برق و کامپیوتر</p>	
<h2>تشخیص و شمارش اشیاء</h2>		

محمد جواد احمدی	نام و نام خانوادگی
۴۰۱۰۰۰۸۶	شماره دانشجویی

فهرست مطالب

۳	پاسخ پرسش دوم
۳	۱.۱ پاسخ قسمت ۱ - توضیحات معماری
۴	۱.۱.۱ شبکه پایه برای استخراج ویژگی ها
۵	۲.۱.۱ شبکه پیشنهاد ناحیه (RPN)
۵	۳.۱.۱ نحوه پیش بینی جعبه محصورگر توسط رگرسور
۶	۴.۱.۱ نحوه آموزش (RPN)
۸	۵.۱.۱ تایع اتلاف چندفعالیتی
۸	۶.۱.۱ معیارهای ارزیابی
۱۰	۲.۱ پاسخ قسمت ۲ - آموزش شبکه پیش آموزش دیده
۲۵	۳.۱ پاسخ قسمت ۳ - ارزیابی

فهرست تصاویر

۴	نمایی کلی از شبکه Faster-RCNN	۱
۵	پیاده‌سازی کانولوشنی معماری RPN	۲
۶	شرح پیش‌بینی شیفت دلتا از جعبه‌های لنگر و مختصات جعبه محصورگر	۳
۷	جعبه‌ای لنگر در مرکز هر پنجره حرکت‌کننده	۴
۸	شبکه پیشنهاد ناحیه	۵
۹	توضیح پارامترها	۶
۱۰	مقایسه‌های خانواده RCNN‌ها	۷
۱۱	شاخص IoU	۸
۱۱	منحنی دقیق-یادآوری	۹
۳۱	نتیجه ارزیابی مدل روی داده‌های تصادفی از مجموعه ارزیابی	۱۰
۳۱	نتیجه ارزیابی مدل روی داده‌های تصادفی از مجموعه ارزیابی	۱۱
۳۲	نتیجه ارزیابی مدل روی داده‌های تصادفی از مجموعه ارزیابی	۱۲
۳۲	نتیجه ارزیابی مدل روی داده‌های تصادفی از مجموعه ارزیابی	۱۳
۳۲	نتیجه ارزیابی مدل روی داده‌های تصادفی از مجموعه ارزیابی	۱۴
۳۲	نتیجه ارزیابی مدل روی داده‌های تصادفی از مجموعه ارزیابی	۱۵
۳۳	نتیجه ارزیابی مدل روی داده‌های تصادفی از مجموعه ارزیابی	۱۶
۳۳	نتیجه ارزیابی مدل روی داده‌های تصادفی از مجموعه ارزیابی	۱۷
۳۳	نتیجه ارزیابی مدل روی داده‌های تصادفی از مجموعه ارزیابی	۱۸
۳۳	نتیجه ارزیابی مدل روی داده‌های تصادفی از مجموعه ارزیابی	۱۹
۳۳	نتیجه ارزیابی مدل روی داده‌های تصادفی از مجموعه ارزیابی	۲۰
۳۴	نتیجه ارزیابی مدل روی داده‌های تصادفی از مجموعه ارزیابی	۲۱
۳۴	یک نمونه بزرگ از نتیجه	۲۲

پرسش ۲. تشخیص و شمارش اشیاء

۱ پاسخ پرسش دوم

توضیح پژوهه کدهای تشخیص و شمارش اشیاء

کدهای مربوط به این قسمت، علاوه بر پژوهه محلی کدها در [این لینک گوگل کولب](#) آورده شده است. مدل‌های ذخیره شده نیز از طریق [این لینک](#) در دسترس هستند.

۱.۱ پاسخ قسمت ۱ - توضیحات معماري

این شبکه یک شبکه دو مرحله‌ای است که در سال ۲۰۱۵ معرفی شده است. در این شبکه هم مشابه سایر روش‌های تشخیص و اشیاء از شبکه‌ای کانولوشنی برای استخراج ویژگی (در قالب تنسورهای سه‌بعدی) استفاده می‌کند ([شکل ۱](#)). نگاشت ویژگی استخراج شده نشان‌داده شده در [شکل ۱](#) با توجه به ابعادی که دارد، یک بردار ۲۰۴۸ تایی است که هر درایه آن یک ماتریس ۳۲×۳۲ است که شامل یک سری نقشه ویژگی تاریک و روشن است که تمرکز و اطلاعات استخراج شده از قسمت‌های مختلف را نشان می‌دهد. بنابراین، یک سری اطلاعات در این نگاشتهای ویژگی وجود دارد. در این شبکه به جای استفاده از الگوریتم جستجوی گزینشی روش نگاشت ویژگی برای شناسایی پیشنهادهایی نواحی، یک شبکه برای پیش‌بینی پیشنهاد نواحی به عنوان بخشی از فرآیند آموزش معرفی شده است که شبکه پیشنهاد ناحیه یا RPN نام دارد. بنابراین، شبکه Faster-RCNN اطلاعات و نگاشتهای ویژگی استخراج شده را به یک ماژول که دارای لایه‌های کانولوشنی مختلف است (RPN) می‌دهد تا یک سری جعبه محصورگر با تعداد بالا پیشنهاد شود. هر جعبه یک بردار به طول چهار است. بنابراین، وظیفه قسمت RPN پیشنهاد نواحی است. این جعبه‌های محصورگر استفاده می‌کنیم تا وارد فاز و مرحله دوم شویم.^۱ در این مرحله برای این که بتوانیم به صورت همزمان از این جعبه‌های محصورگر و نگاشتهای ویژگی استفاده کنیم از یک ماژول به نام ROI Poll استفاده می‌کنیم. ROI Poll یعنی Pooling زدن روی یک ناحیه خاص و مورد علاقه. بنابراین ما نمی‌آییم که از کل نگاشت ویژگی Pooling بگیریم؛ بلکه، تک‌تک جعبه‌های محصورگر تولید شده در مرحله اول را نگاشت می‌کنیم تا روی ناحیه متناظر روی نگاشت ویژگی بیافتد. سپس، فرآیند Pooling را انجام می‌دهیم. به صورت خلاصه اگر n جعبه محصورگر در مرحله اول استخراج شود، در خروجی این مرحله n نگاشت ویژگی خواهیم داشت که سایز آن به عدد درنظرگرفته شده در قسمت Pooling بستگی دارد. در ادامه تک‌تک این نگاشتهای ویژگی را مسطح می‌کنیم و سپس از دو لایه تمام‌متصل عبور می‌دهیم. در انتها یک لایه تمام‌متصل برای طبقه‌بندی داریم و یک لایه تمام‌متصل هم برای پیشنهاد جعبه محصورگر دقیق. برای هر یک از این‌ها هم تابع اتلاف متناظر با هدف در نظر گرفته می‌شود.

بنابراین معماري این شبکه را می‌توان با دو شبکه اصلی شرح داد:

- شبکه پیشنهاد ناحیه (RPN): جستجوی گزینشی با شبکه کانولوشنی جایگزین شده که برای پیشنهاد نواحی مدنظر (RoI) از آخرین نگاشتهای استخراج گر ویژگی برای بررسی در نظر گرفته شود. RPN دو خروجی دارد: «نمرة شیئت (شئ یا عدم شئ)» و محل جعبه. این بخش در واقع نواحی بالقوه حاوی اشیاء مدنظر و محل تقریبی آن‌ها را شناسایی می‌کند.

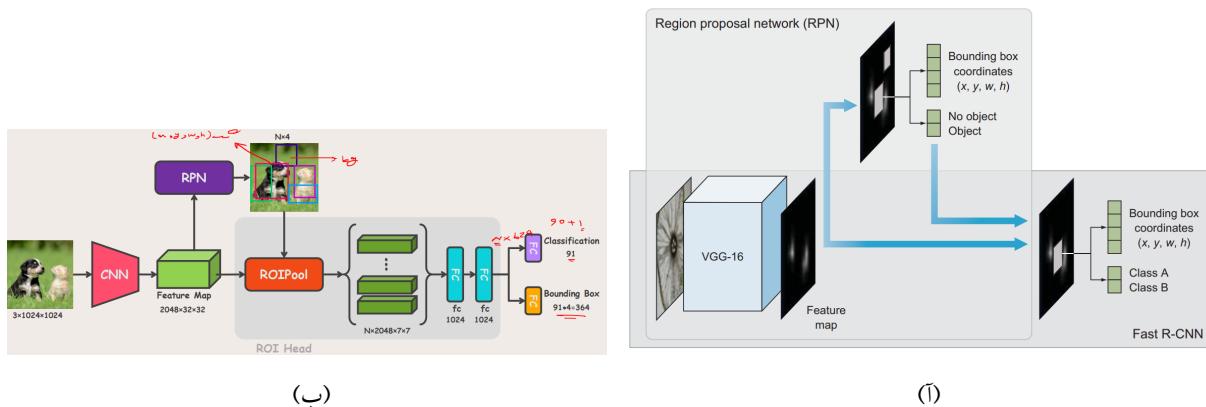
^۱ به همین دلیل این شبکه از نوع دو مرحله‌ای است.

• RCNN سریع: از مولفه‌های معمولی RCNN سریع تشکیل شده است. از جمله شبکه پایه برای استخراج ویژگی که یک مدل کانولوشنی پیش‌آموزش دیده معمولی برای استخراج ویژگی از تصویر ورودی است، و لایه رأی‌گیری ROI که برای استخراج نواحی مدنظر اندازه ثابت استفاده می‌شود. این بخش در واقع اشیاء را دسته‌بندی کرده و با استفاده از جعبه‌های محصورگر محل آن‌ها را تدقیق می‌کند. این بخش یک قسمت به عنوان لایه خروجی هم دارد که حاوی دو لایه تمام‌امتصل است که یکی طبقه‌گر سافت‌مکس برای خروجی دادن احتمال طبقه و دیگری شبکه کانولوشنی رگرسیون جعبه محصورگر برای پیش‌بینی جعبه‌های محصورگر است.

همان‌طور که در **شکل ۱** آمده است، تصویر ورودی به شبکه ارائه شده و ویژگی‌های آن از طریق یک شبکه کانولوشنی پیش‌آموزش دیده استخراج می‌گردد. این ویژگی‌ها به صورت موازی به دو مولفه مختلف معماری مدل ارسال می‌شود:

- به RPN برای تعیین این که کجا در تصویر شیئی بالقوه می‌تواند وجود داشته باشد. در این نقطه مشخص نیست شی چیست و فقط این که بالقوه یک شی در یک ناحیه خاص وجود دارد.
- رأی‌گیری ROI برای استخراج پنجگرهای اندازه ثابت ویژگی‌ها.

در ادامه خروجی به دو لایه تمام‌امتصل ارسال می‌شود. یکی برای طبقه‌گر شی و یکی هم برای پیش‌بینی‌های مختصات جعبه محصورگر به منظور جانمایی‌های نهایی. این معماری به خط مسیر ابتدا تا انتهای آشکارسازی شی قابل آموزش و کامل دست پیدا کرده که همه مولفه‌های مورد نیاز شامل موارد زیر در داخل شبکه انجام می‌شود: استخراج‌گر ویژگی شبکه پایه، پیشنهاد ناحیه، رأی‌گیری ROI، طبقه‌بندی شی و رگرسور جعبه محصورگر.



شکل ۱: نمایی کلی از شبکه Faster-RCNN

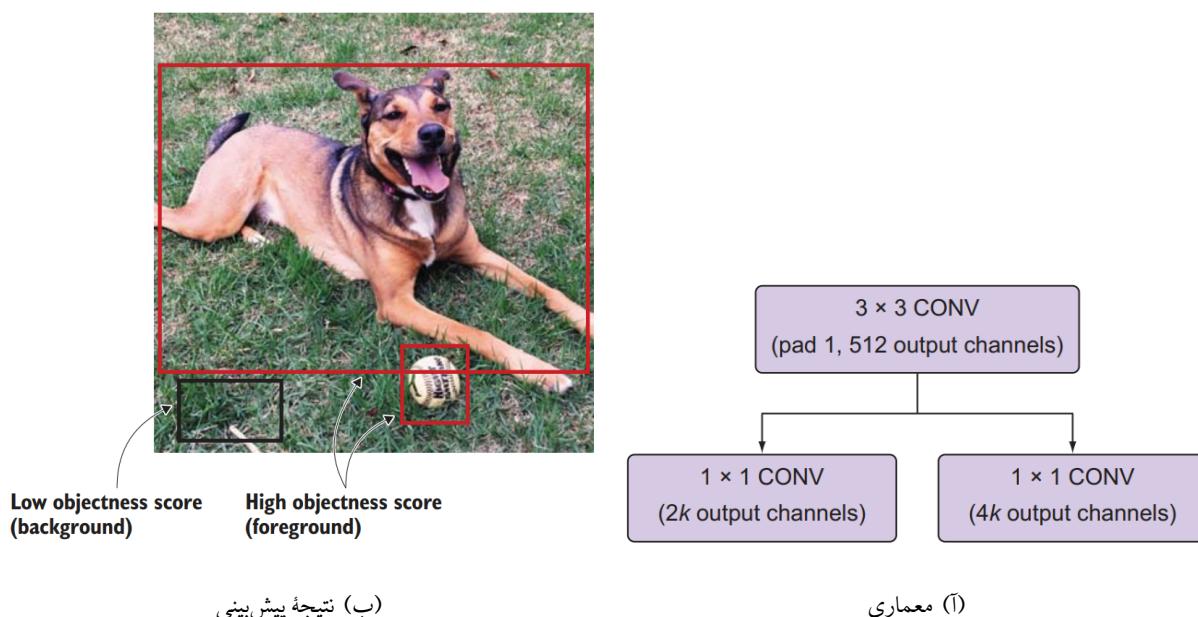
۱.۱.۱ شبکه پایه برای استخراج ویژگی‌ها

مشابه مدل RCNN، اولین گام، استفاده از CNN پیش‌آموزش دیده و قطعه کردن بخش طبقه‌بندی آن است. شبکه پایه برای استخراج ویژگی‌ها از تصویر ورودی استفاده می‌شود. در این مولفه هر معماری CNN مورد اقبال براساس مسئله‌ای که قرار است حل شود، قابل استفاده است. مقاله اصلی این مدل از شبکه‌های ZF و VGG پیش‌آموزش دیده روی ImageNet استفاده کرده ولی از آن موقع تعداد زیادی شبکه مختلف با تعداد اوزان متغیر وجود داشته است. امروزه معماری‌های ResNet عمده‌تاً جایگزین VGG به عنوان شبکه پایه برای استخراج ویژگی شده‌اند. مزیت بدیهی ResNet نسبت به VGG این است که تعداد لایه‌های به

مراتب بیشتری دارد (عمیق‌تر) که این ظرفیت یادگیری ویژگی‌های پیچیده‌تری را فراهم می‌کند. هم‌چنین این معماری با استفاده از اتصالات باقی‌مانده و نرم‌افزاری دسته آموزش مدل عمیق را ساده کرده است.

۲.۱.۱ شبکه پیشنهاد ناحیه (RPN)

این شبکه بر مبنای آخرین نگاشت ویژگی شبکه عصبی کانولوشنی پیش‌آموزش دیده مکان‌هایی را شناسایی می‌کند که می‌توانند به صورت بالقوه حاوی اشیاء مدنظر باشند. این شبکه به شبکه توجه نیز مشهور است؛ چراکه، توجه شبکه را به نواحی جالب در تصویر حلب می‌کند. معماری این شبکه از دو لایه تشکیل شده است. یکی لایه تماماً متصل سه‌درسه با ۵۱۲ کanal و دیگری دو لایه کانولوشنی یک‌دیگر موازی که یکی برای پیش‌بینی طبقه وجود با عدم وجود شئ در یک ناحیه و دیگری برای رگرسیون یا پیش‌بینی جعبه محصورگر به کار می‌رود. لایه کانولوشنی با اندازه سه‌درسه به آخرین نگاشت ویژگی شبکه پایه اعمال می‌شود که پنجره سه‌درسه روی نگاشت ویژگی حرکت داده می‌شود. سپس خروجی به دو لایه کانولوشنی با اندازه یک‌دیگر طبقه‌بند و رگرسور جعبه محصورگر ارسال می‌شود. باید توجه داشت که طبقه‌بند و رگرسور RPN سعی در پیش‌بینی طبقه شئ و جعبه محصورگر آن ندارند. این کار بعد از RPN انجام می‌شود. در این بخش در واقع از یک طبقه‌بند دوتایی برای پیش‌بینی نمرة شیئت و ارسال ناحیه برای بررسی بیش‌تر استفاده می‌شود.

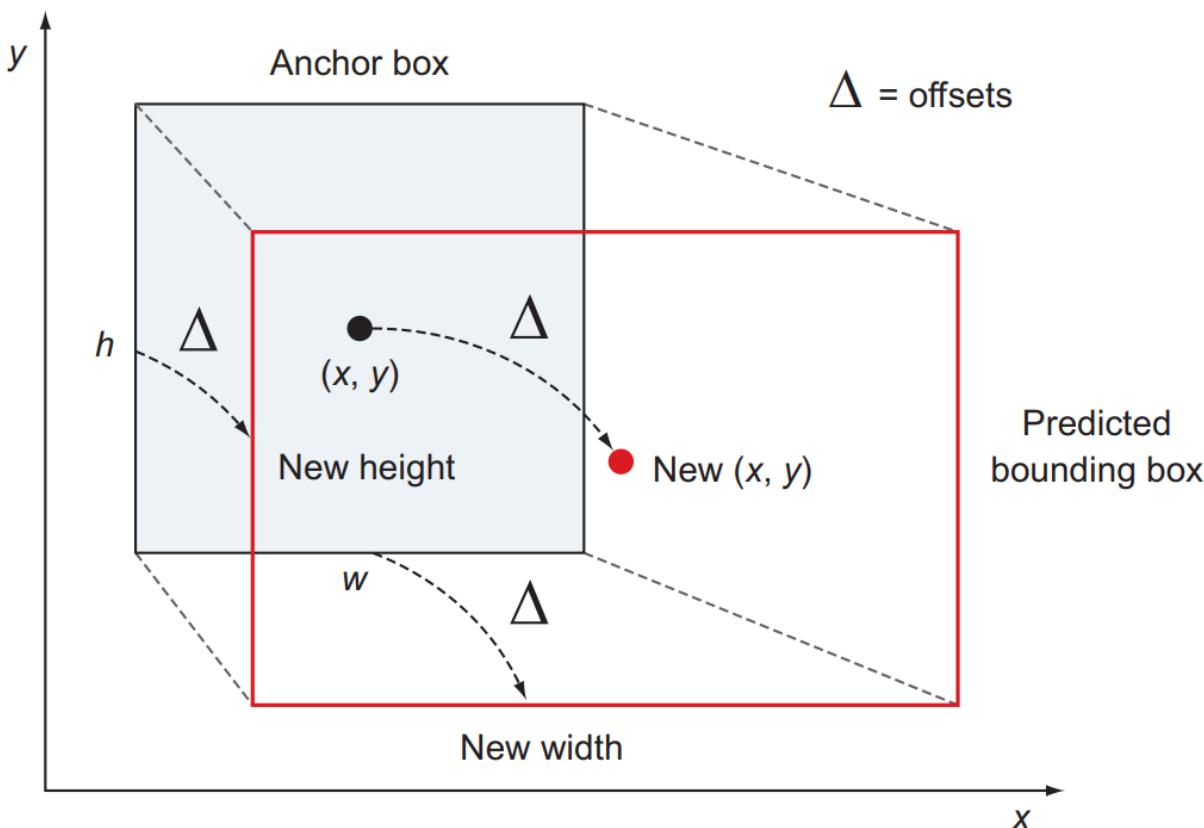


شکل ۲: پیاده‌سازی و نمایش نتیجه کانولوشنی معماری RPN که در آن k تعداد لنگر است.

۳.۱.۱ نحوه پیش‌بینی جعبه محصورگر توسط رگرسور

جعبه محصورگر که یک شئ را محصور می‌کند با چندتایی (x, y, w, h) شناسایی می‌شود. x و y مختصات تصویر بوده که مرکز جعبه را توصیف می‌کند و w و h ارتفاع و عرض جعبه است. تعریف مختصات (x, y) برای مرکز جعبه می‌تواند چالش‌آفرین باشد. چون برخی قواعد به منظور اطمینان از این که شبکه مقادیر را داخل مرزهای تصویر پیش‌بینی می‌کند، لازم است. به جای

آن می‌توان از جعبه‌های مرجع موسوم به لنگر در تصویر ایجاد کرد و لایه رگرسیون را مجبور کرد آفست‌هایی از این جعبه‌ها موسوم به دلتاها ($\Delta_x, \Delta_y, \Delta_w, \Delta_h$) را پیش‌بینی کند تا جعبه‌های لنگر بهتر شئ را گنجانده و پیشنهادهای نهایی به دست آید (شکل ۳).

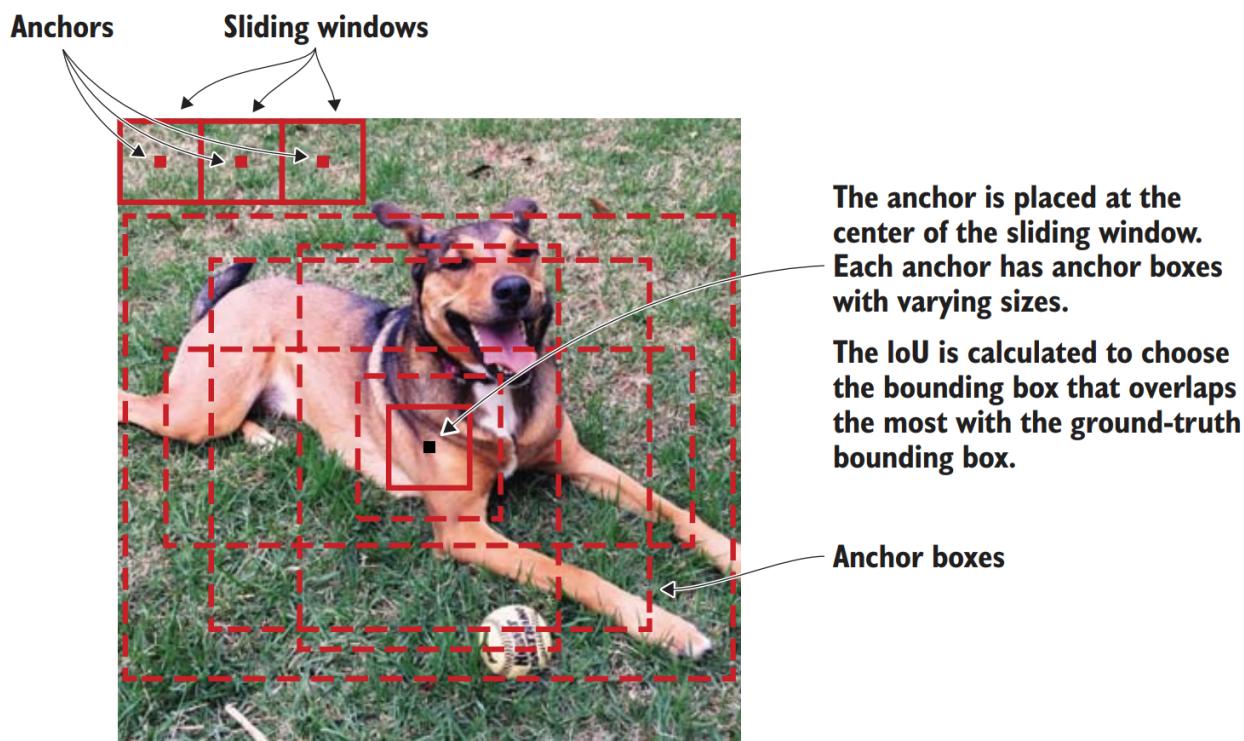


شکل ۳: شرح پیش‌بینی شیفت دلتا از جعبه‌های لنگر و مختصات جعبه مخصوص‌گر.

با روی کرد پنجره متوجه، RPN تعداد k ناحیه برای هر محل در نگاشت ویژگی تولید کرده که به صورت جعبه‌های موسوم به لنگر نمایش داده می‌شوند. لنگرهای در وسط پنجره متوجه متناظر خویش مرکزیت داده شده و از لحاظ مقیاس و نسبت ظاهر (ابعاد) برای پوشش دادن به تنوعی از اشیاء تقاضت دارند. این‌ها جعبه‌های مخصوص‌گری بوده که در سرتاسر تصویر قرار داده شده و در هنگام پیش‌بینی محل‌های شئ در ابتدا، به عنوان مرجع عمل می‌کنند. مثال شکل ۴ را در نزدیکی لنگرهای در مرکز پنجره متوجه هستند. هر پنجره k جعبه لنگر داشته که لنگر در مرکز آن جعبه‌هاست.

۴.۱.۱ نحوه آموزش (RPN)

شبکه پیشنهاد ناحیه برای طبقه‌بندی جعبه لنگر به منظور خروجی دادن نمره شیئیت (حاوی شئ مورد نظر بودن یا نبودن)، و نیز تخمین چهار مختصات شئ (پارامترهای محل)، آموزش داده می‌شود. RPN با استفاده از مفسران انسانی آموزش داده شده تا جعبه‌هایی حصورگر جعبه‌های حقیقت مبنای بر چسب زده شوند. برای هر جعبه لنگر، مقدار احتمال هم‌پوشانی (p) محاسبه شده

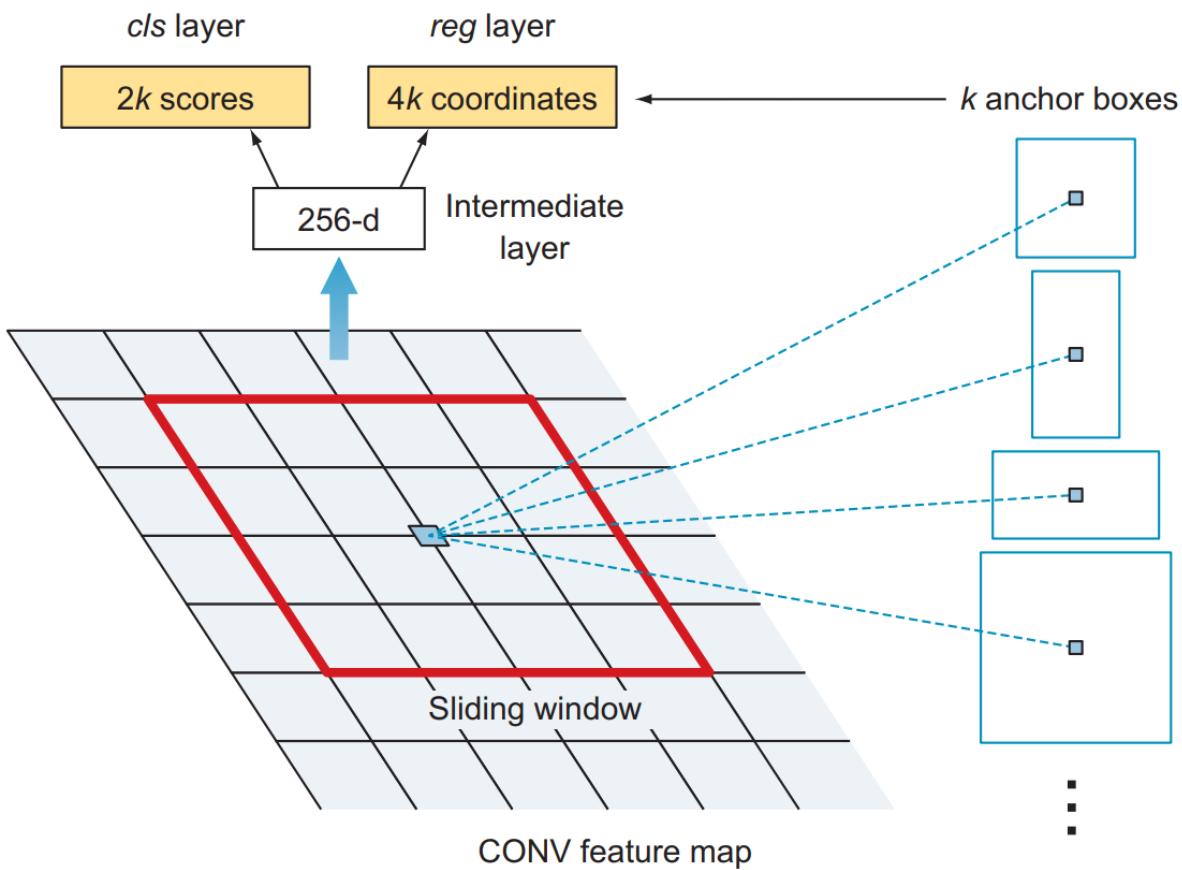


شکل ۴: جعبه‌ای لنگر در مرکز هر پنجره حرکت کننده. IoU محاسبه شده تا جعبه مخصوص‌گر با بیشترین همپوشانی با حقیقت‌بنا انتخاب شود.

که نشانگر میزان همپوشانی این لنگرهای با جعبه‌های مخصوص‌گر حقیقت‌بنا است.

$$p = \begin{cases} 1 & \text{if } \text{IoU} > 0.7 \\ -1 & \text{if } \text{IoU} < 0.3 \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

اگر لنگر میزان همپوشانی بالایی با جعبه مخصوص‌گر حقیقت‌بنا داشته باشد، احتمال دارد که جعبه لنگر حاوی شئ مدنظر بوده، و نسبت به فعالیت دسته‌بندی وجود یا فقدان شئ مثبت برچسب زده می‌شود. مشابه همین حالت، اگر میزان همپوشانی با جعبه حقیقت‌بنا کم باشد، منفی برچسب زده می‌شود. در حین فرآیند آموزش، لنگرهای مثبت و منفی به عنوان ورودی به دو لایه تمام‌امتصال متاظر به دسته‌بندی لنگرهای پارامترهای محل (چهار مختصات) به ترتیب، به عنوان حاوی شئ یا فقدان آن بودن ارسال می‌شود (شکل ۵). متاظر به k لنگر از محل، شبکه RPN نمرات $2k$ و مختصات $4k$ را خروجی می‌دهد. این امر بدین معناست که اگر تعداد لنگر در هر پنجره متاخرک (k) برابر ۹ باشد، RPN تعداد ۱۸ نمره شیئت و ۳۶ مختصات محل خروجی می‌دهد. لایه‌های تمام‌امتصال خروجی هم دو ورودی دریافت می‌کنند. یکی نگاشت ویژگی از شبکه کانولوشن پایه و دیگری نواحی مدنظر از RPN. نواحی منتهب دسته‌بندی شده و طبقه پیش‌بینی آن‌ها و پارامترهای جعبه مخصوص‌گر خروجی داده می‌شود. لایه ظبقه‌بندی شئ در Faster RCNN از فعال‌سازی سافت‌مکس استفاده کرده، در حالی که، لایه رگرسیون محل از رگرسیون خطی رو مختصات تعريف‌کننده محل به عنوان جعبه مخصوص‌گر استفاده می‌کند. همه پارامترهای شبکه با استفاده از اتلاف چندفعالیتی با یکدیگر آموزش داده می‌شوند.



شکل ۵: شبکه پیشنهاد ناحیه.

۵.۱.۱ تابع اتلاف چندفعالیتی

مشابه Faster RCNN، Fast RCNN برای تابع اتلاف چندفعالیتی بهینه شده که در آن تابع اتلاف، اتلاف‌های طبقه‌بندی و رگرسیون جعبه محصورگر را تلفیق می‌کند:

$$L = L_{cls} + L_{loc} \quad (2)$$

$$L (\{p_i\}, \{t_i\}) = \frac{1}{N_{cls}} \sum L_{cls} (p_i, p_i^*) + \frac{\lambda}{N_{loc}} \sum p_i^* \cdot L1_{smooth} (t_i - t_i^*)$$

توضیح پارامترهای رابطه ۶ در شکل ۶ آورده شده است: در نهایت مقایسه‌های گرافیکی جالبی در خصوص خانواده RCNN‌ها در شکل ۷ آورده شده است.

۶.۱.۱ معیارهای ارزیابی

شاخص تلاقی روی اجتماع یا IoU هم پوشانی بین دو جعبه محصورگر حقیقت‌منما و پیش‌بینی را ارزیابی می‌کند (شکل ۸). هرچه این هم پوشانی بیش‌تر باشد بهتر است و میزان IoU هم بزرگ‌تر خواهد بود. این شاخص برای تعریف پیش‌بینی صحیح

Symbol	Explanation
p_i and p_i^*	p_i is the predicted probability of the anchor (i) being an object and the ground, and p_i^* is the binary ground truth (0 or 1) of the anchor being an object.
t_i and t_i^*	t_i is the predicted four parameters that define the bounding box, and t_i^* is the ground-truth parameters.
N_{cls}	Normalization term for the classification loss. Ren et al. set it to be a mini-batch size of ~256.
N_{loc}	Normalization term for the bounding box regression. Ren et al. set it to the number of anchor locations, ~2400.
$L_{cls}(p_i, p_i^*)$	The log loss function over two classes. We can easily translate a multi-class classification into a binary classification by predicting whether a sample is a target object: $L_{cls}(p_i, p_i^*) = -p_i^* \log p_i - (1 - p_i^*) \log (1 - p_i)$
$L1_{smooth}$	As described in section 7.2.2, the bounding box loss measures the difference between the predicted and true location parameters (t_i, t_i^*) using the smooth L1 loss function. It is a robust function and is claimed to be less sensitive to outliers than other regression losses like L2.
λ	A balancing parameter, set to be ~10 in Ren et al. (so the L_{cls} and L_{loc} terms are roughly equally weighted).

شکل ۶: توضیح پارامترهای رابطه ۲.

به عنوان نوعی حد آستانه به کار می‌رود. متداول‌ترین سنجه مورد استفاده برای ارزیابی در فعالیت‌های آشکارسازی شئ AP یا میانگین متوسط دقت است. درصدی بین ۰ و ۱۰۰ که هرچه بزرگ‌تر باشد بهتر است. این سنجه معمولاً با یک مقدار آستانه IoU بیان می‌شود که به IoU مربوط است. این آستانه معمولاً ۰.۵ یا ۰.۷۵. انتخاب می‌شود و به آن معناست که اگر IoU بالاتر از این حد آستانه باشد، مورد مثبت صحیح یا TP است و اگر زیر آن باشد، مورد مثبت کاذب یا FP است. با محاسبه دقت و پادآوری بر مبنای منطق گفته شده و روابط رابطه ۳، منحنی PR برای همه طبقات رسم می‌شود (شکل ۹). ساخت AP با محاسبه زیر سطح این منحنی محاسبه می‌شود و در نهایت mAP میانگین AP محاسبه شده برای تمام طبقات است. بنابراین حالا مشخص است که چرا با وجود این که نتایج آورده شده در ۱۶ در آستانه‌های ۰.۵ و ۰.۷۵ خوب بوده، بعضًا در حالت پیش‌بینی، جعبه‌های محصورگر پیش‌بینی با وجود جایابی درست تمام شئ را نگرفته‌اند. به نوعی می‌توان گفت که آستانه ۰.۵ تا ۰.۹۵ که سختگیرانه‌تر است مربوط به جایابی دقیق شئ است و آستانه ۰.۵ بیش‌تر روی تشخیص شئ تمرکز دارد.

$$\text{Recall} = \frac{TP}{TP + FN}$$

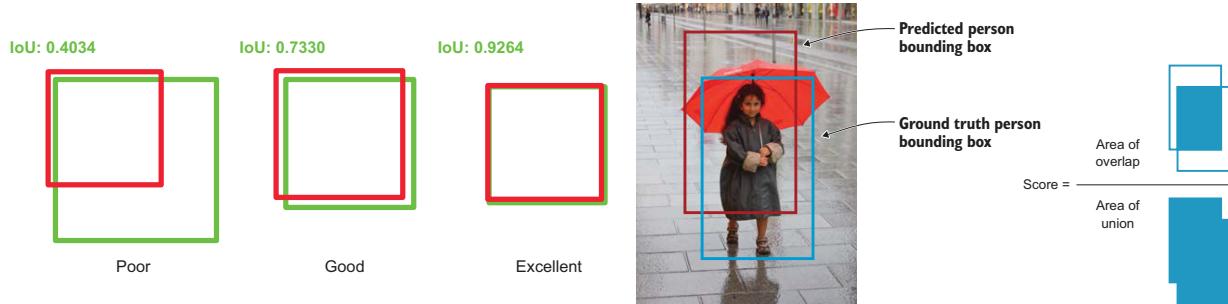
$$\text{Precision} = \frac{TP}{TP + FP} \quad (3)$$

	R-CNN	Fast R-CNN	Faster R-CNN
mAP on the PASCAL Visual Object Classes Challenge 2007	66.0%	66.9%	66.9%
Features	<ol style="list-style-type: none"> 1 Applies selective search to extract RoIs (~2,000) from each image. 2 A ConvNet is used to extract features from each of the ~2,000 regions extracted. 3 Uses classification and bounding box predictions. 	<p>Each image is passed only once to the CNN, and feature maps are extracted.</p> <ol style="list-style-type: none"> 1 A ConvNet is used to extract feature maps from the input image. 2 Selective search is used on these maps to generate predictions. <p>This way, we run only one ConvNet over the entire image instead of ~2,000 ConvNets over 2000 overlapping regions.</p>	<p>Replaces the selective search method with a region proposal network, which makes the algorithm much faster.</p> <p>An end-to-end DL network.</p>
Limitations	High computation time, as each region is passed to the CNN separately. Also, uses three different models for making predictions.	Selective search is slow and, hence, computation time is still high.	Object proposal takes time. And as there are different systems working one after the other, the performance of systems depends on how the previous system performed.
Test time per image	50 seconds	2 seconds	0.2 seconds
Speed-up from R-CNN	1x	25x	250x

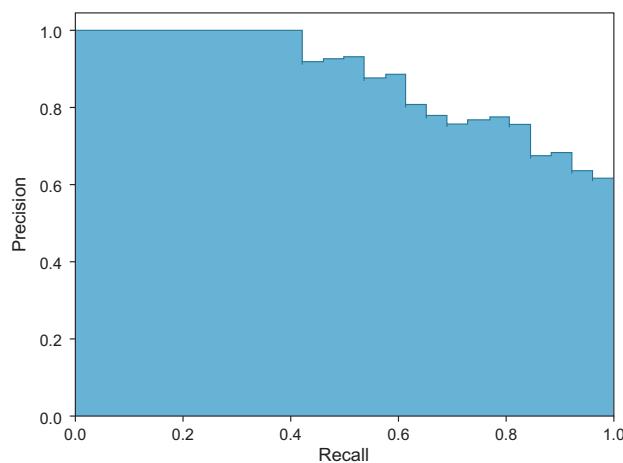
شکل ۷: مقایسه‌های خانواده RCNN‌ها.

۲.۱ پاسخ قسمت ۲ - آموزش شبکه پیش‌آموزش دیده

در گام اول، مجموعه‌داده و مجموعه‌فایل‌های پایتونی موردنیاز را روی [این لینک](#) و [این لینک](#) بارگذاری می‌کنیم. این کار به این دلیل انجام می‌شود که دریافت مستقیم از لینک مذکور به دلیل احتمالاً محدودیت‌های سرور مرجع بسیار کند بوده است. همچنین،



شکل ۸: شاخص IoU



شکل ۹: منحنی دقت-یادآوری.

فراخوانی این فایل بدون نیاز به Mount کردن و استفاده از gdown در محیط گوگل کولب ممکن است با خطا مواجه شود که با استفاده از دستورات زیر مشکل حل می شود ([منبع](#)). در ادامه هم دستوراتی برای خروج داده ها از حالت فشرده و رفتن به یک پوشه مخصوص مجموعه داده نوشته ایم و در انتها فایل فشرده را حذف کردہ ایم.

```

1 !pip install --upgrade --no-cache-dir gdown
2 !gdown 1q-alwIBf1sYToN5D4SAi5kaVMSemuv-J
3
4 import zipfile
5 zip_file_path = '/content/PASCAL.zip'
6 folder_path = '/content/PASCAL'
7 # Extract the zip file to the specified folder
8 with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:

```

```

9     zip_ref.extractall(folder_path)

10

11 import os
12 file_path = "/content/PASCAL.zip"
13 if os.path.exists(file_path):
14     os.remove(file_path)
15     print(f"{file_path} has been deleted successfully.")
16 else:
17     print(f"{file_path} does not exist.")

```

در ادامه، توضیحاتی در مورد کدهایی که در پوشه Required_Files در اختیار ما قرار داده شده ارائه می‌کنیم. ابتدا به سراغ فایل engine.py می‌رویم. این کد، یکتابع به نام train_one_epoch دارد که به منظور آموزش یک مدل شناخته‌شده برای تشخیص و شمارش اشیاء به کار می‌رود. در این تابع، مواردی مانند یک مدل، یک بهینه‌ساز و یک دستگاه (CPU یا GPU) به عنوان ورودی گرفته می‌شوند. این تابع مدل را روی داده‌های آموزشی آموزش می‌دهد، سپس معیارهایی مانند تلفات و نرخ یادگیری را در هر دوره آموزش ثبت و نمایش می‌دهد. این تابع با استفاده از دیتالودر، داده‌های آموزشی را دریافت می‌کند و مدل را در حالت آموزش قرار می‌دهد. در هر دوره، تصاویر و برچسب‌های مربوط به آن‌ها به دستگاه ورودی منتقل می‌شوند. این تابع داده‌های آموزشی را به شبکه ورودی می‌دهد و برای هر گام، خروجی مدل را با برچسب‌های مربوطه مقایسه می‌کند. سپس خطای آموزش را با استفاده از الگوریتم پس انتشار خطای محاسبه و بهینه‌ساز را برای به روزرسانی وزن‌های شبکه آماده می‌کند. بنابراین، با استفاده از مدل، اشیاء موجود در تصاویر تشخیص داده می‌شوند. سپس تلفات محاسبه می‌شود و مدل با استفاده از بهینه‌ساز بهبود می‌یابد. این تابع مدل را در یک دوره آموزشی آموزش می‌دهد. این تابع شامل یک پارامتر به نام scaler است که می‌تواند باعث افزایش سرعت آموزش شود، زیرا از این ضریب برای تغییر دقت محاسباتی در GPU استفاده می‌شود. در ابتدای تابع، مدل به حالت آموزش در می‌آید و یک شیء MetricLogger برای ریدیابی معیارهای آموزشی ساخته می‌شود. این شیء برای نمایش معیارهای مختلف مانند تلفات، نرخ یادگیری و غیره در هر چند دوره چاپ استفاده می‌شود. سپس اگر عدد دوره برابر با صفر باشد، یک برنامه‌ریزی کننده نرخ یادگیری برای مدل ایجاد می‌شود. در واقع یک Scheduler زمانی (lr_scheduler) برای مدیریت نرخ یادگیری ایجاد می‌شود. این تابع، اگر دوره (ایپاک) برابر با صفر باشد، در ابتدای از الگوریتم کاهش پایانی نرخ یادگیری به عنوان یک الگوی گرم‌کننده برای شبکه استفاده می‌کند. با استفاده از تابع torch.optim.lr_scheduler.LinearLR، میزان نرخ یادگیری در هر دوره گرم‌کننده به صورت خطی افزایش پیدا می‌کند و در پایان دوره، به میزان نرخ یادگیری اصلی مدل می‌رسد.

در هر دوره آموزش، معیارهایی نظری تلفات، نرخ یادگیری و معیارهای دیگر ثبت شده و با استفاده از تابع metric_logger.update به روزرسانی می‌شوند. در پایان، معیارهایی مانند تلفات کلی آموزش، نرخ یادگیری و دیگر معیارهای مهم را به عنوان خروجی تابع بازگردانده می‌شود. برای هر بار بازگذاری داده، تصاویر و برچسب‌ها به دستگاه منتقل شده و در صورت استفاده از روش Mixed Precision training با استفاده از autocast، عملیات‌های مربوط به مدل با دقت نیمه‌ای (half-precision) اجرا می‌شود. می‌توان میزان دقت مختلط یک روش برای سرعت بخشیدن به آموزش شبکه‌های عصبی با استفاده از GPU است. در این روش، برای محاسبات عملیات پیچیده و سنگین، از دقت ۱۶ بیتی به جای دقت ۳۲ بیتی استفاده می‌شود. بدین ترتیب، حجم داده‌های انتقالی بین CPU و GPU کاهش می‌یابد و عملیات‌های محاسباتی سریع‌تر انجام می‌شود. هم یکی از ابزارهای موجود در کتابخانه پایه تورچ برای پیاده‌سازی Mixed Precision training است. با استفاده از این ابزار، می‌توان بدون نیاز به دسترسی به تجهیزات سخت‌افزاری خاص، عملیات‌های پردازشی مورد نیاز در دقت نیمه‌دقیقه (half-precision) انجام داد. به این ترتیب، سرعت پردازش بالا رفته و حجم حافظه مصرفی نیز کاهش می‌یابد. در این روش، برای انجام محاسبات، از نوع داده‌ای ۱۶ بیتی استفاده می‌شود. در حالت عادی، از داده‌ای ۳۲ بیتی برای انجام محاسبات

استفاده می‌شود. با استفاده از داده‌های نیمه‌دقیق، حجم حافظه مورد استفاده کاهش می‌یابد و همچنین سرعت پردازش نیز افزایش می‌یابد. اما در برخی موارد، استفاده از داده‌های نیمه‌دقیق می‌تواند باعث کاهش دقت مدل شود، بنابراین، این انتخاب و کار باید با دقت انجام شود.

در ادامه، تصاویر و برچسب‌ها به مدل داده می‌شود و نتیجه تصمیم‌گیری مدل (`loss_dict`) به دست می‌آید. با استفاده از تابع `reduce_dict`، مقدار تلفات به دست آمده از همه دستگاه‌ها جمع‌آوری و برای نمایش در `MetricLogger` محاسبه می‌شود. سپس مقدار کلی تلفات و مقدار تلفات هر نوع برچسب تعیینی در `MetricLogger` بهروزرسانی می‌شود. همچنین نرخ یادگیری نیز در `MetricLogger` بهروزرسانی می‌شود. بنابراین، از آن‌جا که لازم است که معیارهایی را برای بررسی عملکرد الگوریتم در هر دوره آموخت بدست آوریم، از دو کلاس آماده `utils.MetricLogger` و `utils.SmoothedValue` استفاده شده است. از `utils.SmoothedValue` برای محاسبه میانگین میانه در مدت زمان بلندمدت استفاده شده است. این تابع پارامترهایی مانند `print_freq` هم دارد که تعیین می‌کنند فرکانس چاپ خروجی در هر چند دوره یکبار باشد.

فایل `engine.py` توابع دیگری هم دارد. تابع `get_iou_types` برای ایجاد لیستی از جنس `Intersection over Union` (IoU) که برای ارزیابی عملکرد الگوریتم استفاده می‌شوند، طراحی شده است. `Intersection over Union` یا IoU در واقع یک معیار اندازه‌گیری دقت در مسائل تشخیص شئ است. این معیار اندازه‌گیری، نسبت مساحت مشترک دو باکس (شئ و یاشی‌ها) به مجموع مساحت هر دو باکس را نشان می‌دهد. برای مثال، در مسئله تشخیص اشیاء در یک تصویر، IoU نشان‌دهنده میزان هم‌پوشانی جعبه‌های محصورگر شئ پیش‌بینی شده با جعبه‌های محصورگر واقعی در تصویر است. از IoU برای ارزیابی عملکرد مدل و محاسبه میزان اطمینان ماشین در تشخیص اشیاء در تصویر استفاده می‌شود. در اینجا، تابع از مدل داده شده استفاده کرده و از طریق بررسی اینکه آیا مدل عادی یا `MaskRCNN` یا `KeypointRCNN` است، نوع‌های مختلف U-`CNNR` `Faster KeypointRCNN` و `MaskRCNN` از مدل `KeypointRCNN` عملکرد مدل در نظر می‌گیرد. در الگوریتم‌های تشخیص شئ `MaskRCNN` و `KeypointRCNN` استفاده می‌شود که قابلیت تشخیص شئ و هم‌چنین نقاط کلیدی روی اشیاء را داراست. این دو الگوریتم بر اساس ورودی خود متفاوت عمل می‌کنند. در الگوریتم `KeypointRCNN` علاوه بر تشخیص محل اشیاء، نقاطی کلیدی هم برای اشیاء پیش‌بینی می‌شود. بنابراین در خروجی مدل مقادیری برای موقعیت نقاط کلیدی مربوط به هر شئ برگردانده می‌شود که می‌تواند شامل عددی برای شناسایی هر نقطه کلیدی باشد. اما در الگوریتم `MaskRCNN`، علاوه بر تشخیص محل اشیاء، ماسک‌هایی نیز برای هر شئ پیش‌بینی می‌شوند. ماسک‌ها معمولاً شامل پیکسل‌هایی هستند که به شئ مربوط می‌شوند و تصویری از شکل شئ را نشان می‌دهند.

فایل `engine.py` در نهایت شامل تابعی به نام `evaluate` است. این تابع برای ارزیابی عملکرد مدل `Faster RCNN` روی داده‌های تست استفاده می‌شود. توابعی مانند `get_iou_types`، `get_coco_api_from_dataset` و `_get_coco_types` از `CocoEvaluator` کتابخانه `detectron2` برای ایجاد و مدیریت دیتاست استفاده شده‌اند. ابتدا با تنظیم حالت مدل به حالت ارزیابی و تعیین تعداد `threads`، یک دسته‌نمونه از داده‌های تست را دریافت می‌کنیم. در ابتدا تعداد `threads` برابر با تعداد `threas` ای که در حال حاضر در حال اجرا هستند ذخیره می‌شود و روی ۱ تنظیم می‌شود. این کار برای جلوگیری از مشکلاتی که در اجرای تابع `paste_masks_in_image` با مدیریت موازی ممکن است پیش بیاید، انجام می‌شود. با بردن داده‌های تست به دستگاه مورد نظر و اعمال مدل به تصاویر، خروجی‌های مدل برای هر یک از تصاویر به دست می‌آیند. سپس خروجی‌های مدل به صورت یک لیست از دیکشنری‌ها به اندازه دستهٔ تست بازنویسی می‌شوند. سپس یک دیکشنری به نام `res` ساخته می‌شود که برای هر تصویر، یک شناسه تصویر و یک دیکشنری از خروجی‌های مدل شامل جعبه‌های محصورگر و اطلاعات مربوط به آن‌ها را در بر می‌گیرد. در نهایت، با استفاده از تابع `coco_evaluator` در `update`، نتایج مدل برای هر تصویر به دیتاست اضافه می‌شود. در نهایت، تمامی پیش‌بینی‌های انجام‌شده برای تمامی تصاویر به صورت جمع‌بندی شده و ارزیابی شده و در معیارهای مختلفی مانند AP و mAP با آستانه‌های مختلف محاسبه و نمایش داده می‌شود.

فایل `pascal_dataset.py` شامل کدهایی است که یک کلاس به نام `PASCALDataset` ایجاد می‌کند که برای خواندن داده‌های موجود در پوشه ارائه شده استفاده می‌شود. در این کلاس، اطلاعات تصاویر با استفاده از کتابخانه `PIL` بارگیری می‌شود. همچنین فایل‌های مربوط به برچسب‌ها و جعبه‌های محصورگر برای هر تصویر با استفاده از کتابخانه `Scipy` بارگیری می‌شوند. در ادامه با استفاده از کتابخانه `Torchvision` و تابع `Compose`، تصاویر تبدیل و در قالب تسور ذخیره می‌شوند. هر تصویر دارای چند شیء است که هر کدام دارای یک جعبه محصورگر و یک برچسب هستند. سپس، با توجه به اطلاعات جعبه‌های محصورگر، ناحیه مورد نظر برای هر شیء در تصویر مشخص می‌شود و در قالب تسور ذخیره می‌شود. همچنین، برای هر تصویر یک شناسه منحصر به فرد نیز اختصاص داده می‌شود. در نهایت، تمام اطلاعات تصویر و برچسب‌ها به صورت دیکشنری در متغیر `target` ذخیره می‌شود و تصویر و `target` به عنوان خروجی تابع بازگردانده می‌شوند.

شامل سه کلاس `Compose`، `RandomHorizontalFlip`، و `ToTensor` است. همه این کلاس‌ها تبدیل‌های مختلفی را برای تصاویر و هدف‌ها در داده‌های ورودی انجام می‌دهند. کلاس `Compose` تبدیل‌های مختلف را ترکیب می‌کند. به عنوان ورودی، تصویر و هدف (`target`) را می‌گیرد و این تبدیل‌ها را به ترتیب روی تصویر و هدف اعمال می‌کند و تصویر و هدف تغییریافته را خروجی می‌دهد. کلاس `RandomHorizontalFlip` بنا به احتمال مشخص شده (`prob`) تصاویر را افقی می‌چرخاند. بنابراین، علاوه بر تغییر تصویر، جعبه‌های محصورگر (`bbox`)، ماسک‌ها (`masks`) و نقاط کلیدی (`keypoints`) نیز به همان صورت تغییر می‌کنند. این مسئله از این جهت اهمیت دارد که اگر برچسب‌های دیگر به تناسب تغییر تصویر تغییر نکنند، هدق ایجادشده غلط خواهد بود. کلاس `ToTensor` تصویر و هدف را به شکل تسور تبدیل می‌کند. با فراخوانی این تبدیل، تصویر به شکل تسور تغییر می‌کند و هدف بدون تغییر برگردانده می‌شود. در کل، این کلاس‌ها به منظور تبدیل تصاویر و هدف‌هایشان به شکلی قابل قبول جهت آموزش مدل‌های یادگیری عمیق برای وظایف شناسایی اشیاء استفاده می‌شوند.

فایل `utils.py` شامل کدهایی است که یک کلاس با نام `SmoothedValue` را تعریف می‌کند که برای ردیابی یک سری از مقادیر استفاده می‌شود. این کلاس دارای چند ویژگی است که می‌توانند مقادیر مختلفی را بازگرداند. این ویژگی‌ها عبارتند از: `deque` که یک صف کاملاً شناور (FIFO) که حداقل تعداد آن توسط پارامتر `window_size` تعیین می‌شود. صف کاملاً شناور یک ساختار داده است که از یک لیست پویا تشکیل شده است و در آن، اولین موردی که به آن اضافه شده، اولین موردی است که بیرون می‌رود (FIFO). این بدين معناست که عناصری که زودتر به صف اضافه می‌شوند، زودتر از صف خارج می‌شوند. `total` مجموع کلی مقادیر اضافه شده به `deque` و `count` تعداد کلی مقادیر اضافه شده به `deque` است. یک رشته قالب‌بندی است که برای چاپ مقادیر استفاده می‌شود. `update` یک مقدار به `deque` اضافه می‌کند و `count` و `total` را بروز می‌کند. `synchronize_between_processes` یک تسور از مقدار `COUNT` و `total` را ایجاد می‌کند و آن را با استفاده از تابع `all_reduce` همگام می‌کند. `avg`، `global_avg`، `median`، `max`، `value` به ترتیب میان‌بان (میانه)، میانگین، میانگین کلی، بیشینه و مقدار اخیر موجود در `avg` را بازمی‌گرداند.

این کد هم‌چنین یک کلاس متريک را به نام `MetricLogger` تعریف می‌کند که وظیفه ثبت و نمایش معیارهای عملکرد یک مدل یادگیری عمیق در حین آموزش را بر عهده دارد. در کلاس `MetricLogger` یک دیکشنری از متريک‌ها درست می‌شود. تابع `add_meter` در اين کلاس برای افروzen متريک جديد تعریف شده است. تابع `all_gather` در اين کد برای جمع‌آوری داده‌ها از تمامی پردازنده‌ها در یک محیط توزیع شده استفاده می‌شود. اين تابع داده‌های دلخواه قبل پیکل شدن را دریافت می‌کند و برای هر پردازه، اندازه تسور معادل با داده‌های دریافتی به همراه داده‌های دریافتی را به عنوان یک لیست به صورت تنظیم شده جمع‌آوری می‌کند. تابع `reduce_dict` نیز برای کاهش مقادیر متريک‌های ثبت شده در هر پردازه به صورت یکدست و محاسبه مقدار متوسط از مقادیر بدست آمده در همه پردازه‌ها استفاده می‌شود.

فایل `coco_utils.py` یک سری تبدیلات برای دیتاست COCO (که شامل تصاویر و دسته‌بندی اشیا در تصاویر و `FilterAndRemapCocoCategories` آن‌ها می‌باشد) انجام می‌دهد. ابتدا یک کلاس با نام `segmentation mask ing box`

تعریف شده است که دسته‌بندی‌های COCO را مشخص می‌کند و در صورت لزوم، دسته‌بندی‌ها را با اعداد صحیح $0 \text{--} n$ نشان می‌دهد. سپس، یک تابع به نام `convert_coco_poly_to_mask` تعریف شده است که برای تبدیل پلی‌گانه‌های COCO به ماسک استفاده می‌شود. در نهایت، یک کلاس به نام `ConvertCocoPolysToMask` تعریف شده است که تبدیلات نهایی را انجام می‌دهد. این کلاس با استفاده از تابع `convert_coco_poly_to_mask`، ماسک متناظر با هر شیء را ایجاد می‌کند و جعبه مخصوصگر و مشخصات دیگر شیء را در یک دیکشنری ذخیره می‌کند. سپس تصویر و دیکشنری حاوی مشخصات شیء خروجی داده می‌شود. `_has_valid_annotation` چک می‌کند که آیا برچسب و توضیحاتی در مورد اشیاء توسط کاربر ارائه شده است یا نه؟ در پایان، برای وظیفه تشخیص نقاط کلیدی، این تابع به دنبال این است که آیا تصویر حاوی حداقل تعدادی از نقاط کلیدی موردنظر است یا نه. این کد یک تابع به نام `get_coco` دارد که در ورودی مسیر مجموعه‌داده‌ها، نوع مجموعه‌داده‌ها (آموزش یا ارزیابی)، تبدیلات و حالت مجموعه‌داده (نوع مسئله مثلاً تشخیص اشیا یا شناسایی نقاط کلیدی) را می‌گیرد و در خروجی شیء از نوع `COCODataset` را بر می‌گرداند. ابتدا مسیر فایل مشخص می‌شود و تبدیلاتی که در صورت نیاز باید انجام می‌شود به تابع `Compose` ارجاع داده می‌شود. سپس یک شیء از نوع `COCODataset` ایجاد می‌شود. این کار با صدا زدن تابع `CocoDetection` انجام می‌شود که یکی از کلاس‌های `torchvision.datasets` است و ویژگی‌های یک مجموعه‌داده `COCO` را از پوشه‌ای که در ورودی مشخص شده است بارگیری می‌کند. تابع `CocoDetection`، تصویر و همچنین اطلاعات مربوط به برچسب‌ها به صورت دیکشنری به همراه شناسه تصویر (`image_id`) ذخیره می‌کند. سپس برای هر برچسب، یک دیکشنری به صورت مجزا ایجاد می‌شود که اطلاعات مربوط به برچسب شامل مواردی از قبیل مختصات `bbox`، نام دسته‌بندی، مساحت، تعداد نقاط کلیدی و ماسک (اگر وجود داشت) را در خود ذخیره می‌کند. در نهایت، دیکشنری ساخته شده شامل اطلاعات برچسب‌ها و تصاویر به صورت شیء `COCO` برگردانده می‌شود.

فایل `coco_eval.py` یک کلاس به نام `CocoEvaluator` پیاده سازی می‌کند که یک شیء از کلاس `COCO` را به عنوان ورودی می‌گیرد و ارزیابی شیء پیش‌بینی شده را انجام می‌دهد. در تابع `__init__`، یک شیء از کلاس `COCO` و نوع `IoU` مورد نظر برای ارزیابی به عنوان ورودی گرفته می‌شود و اولین مرحله از ایجاد یک شیء از کلاس `COCOeval` برای هر نوع `IoU` را انجام می‌دهد. تابع `prepare`، شیء پیش‌بینی شده را به شکلی مناسب برای ورودی `COCOeval` تبدیل می‌کند و می‌تواند برای نوع‌های مختلف `IoU` متفاوت باشد. تابع `prepare_for_coco_detection`، شیء پیش‌بینی شده را به شکلی مناسب برای ورودی `COCOeval` برای مسئله شناسایی (Detection) تبدیل می‌کند. تابع `prepare_for_coco_segmentation`، شیء پیش‌بینی شده را به شکلی مناسب برای ورودی `COCOeval` برای مسئله شناسایی تصاویر با استفاده از ماسک‌های پیش‌بینی شده تبدیل می‌کند. را به همین کار را با تمرکز بر نقاط کلیدی انجام می‌دهد. تابع `update`، شیء پیش‌بینی شده را به عنوان ورودی گرفته و با استفاده از توابع `prepare` و `COCOeval`، نتایج ارزیابی را برای هر نوع `IoU` محاسبه می‌کند. تابع `synchronize_between_processes` برای همگام‌سازی بین فرآیندهای مختلف استفاده می‌شود. تابع `accumulate` برای محاسبه معیارهای ارزیابی و افزایش شمارنده‌ها استفاده می‌شود. تابع `summarize` برای چاپ خلاصه‌ای از معیارهای ارزیابی استفاده می‌شود. تابع `convert_to_xywh` یک تنسور شامل جعبه‌های مستطیلی با ۴ مختصات `x_max`، `y_min`، `x_min` و `y_max` را به عنوان ورودی می‌گیرد و یک تنسور جدید شامل ۴ مختصات جدید به شکل `x`، `y`، `w` و `h` بازمی‌گرداند. تابع `merge` یک لیست از شناسه‌های تصاویر و لیستی از تصاویر مورد ارزیابی را به عنوان ورودی می‌گیرد و این لیست‌ها را از تمام پردازنده‌های موجود در سیستم جمع‌آوری می‌کند. سپس این لیست‌های جمع‌آوری شده را با هم ترکیب می‌کند. سپس عناصر تکراری از لیست‌ها حذف و ترتیب آن‌ها به صورت صعودی مرتب می‌شوند. در نهایت، لیست شناسه تصاویر جدید و تصاویر ارزیابی که شامل نتایج ارزیابی برای تصاویر گوناگون است، برگردانده می‌شود. تابع `create_common_coco_eval` لیست شناسه‌های تصویر و لیست تصاویر ارزیابی را به عنوان ورودی دریافت می‌کند و بعد از ترکیب‌شدن این دو لیست با استفاده از تابع `merge`، شناسه‌های تصاویر جدید و تصاویر ارزیابی مرتبط با آن‌ها برای مقایسه و ارزیابی مشترک مورد توجه قرار می‌گردند.

در نهایت، تابع evaluate یک شی به نام imgs را به عنوان ورودی دریافت می‌کند که نمونه‌هایی از تصاویر و برچسب‌های متناظر با آن‌ها را شامل می‌شود. این تابع با استفاده از ماژول io و redirect_stdout از sys، خروجی متدهای evaluate را دریافت می‌کند و این خروجی را در یک StringIO object می‌نویسد.

با ارائه این توضیحات مشابه کاری که برای بارگیری مجموعه داده انجام دادیم، از دستورات زیر برای داشتن پوشۀ Required_Files در محیط گوگل کولب استفاده می‌کنیم.

```

1 !pip install --upgrade --no-cache-dir gdown
2 !gdown 1QheCGZHOZeAQ5MIu-3_41E-Qf0hu6xx9
3
4 import zipfile
5 zip_file_path = '/content/Required_Files.zip'
6 folder_path = '/content/Required_Files'
7 # Extract the zip file to the specified folder
8 with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
9     zip_ref.extractall(folder_path)
10
11 import os
12 file_path = "/content/Required_Files.zip"
13 if os.path.exists(file_path):
14     os.remove(file_path)
15     print(f"{file_path} has been deleted successfully.")
16 else:
17     print(f"{file_path} does not exist.")
```

در ادامه کتابخانه‌های لازم را فراخوانی کردیم و با استفاده از دستور زیر، مسیر دایرکتوری پوشۀ Required_Files را به لیست مسیرهای سیستم اضافه می‌کنیم. این مسیر ممکن است حاوی برخی فایل‌ها یا پوشۀ‌های مورد نیاز برنامه باشد. با اضافه کردن این مسیر، پایتون به راحتی می‌تواند به این فایل‌ها و پوشۀ‌ها دسترسی پیدا کند.

```
1 sys.path.append('/content/Required_Files/Required_Files/')
```

در ادامه دستورات زیر را برای آماده سازی مجموعه داده‌ها در دسته‌های مختلف می‌نویسیم. این دستورات یک شیء از کلاس PASCALDataset را ایجاد می‌کنند که برای بارگذاری دیتاست PASCAL استفاده می‌شود. هر سه خط کد مشابه هستند ولی هر کدام با پارامترهای متفاوت ایجاد شده‌اند. در هر خط کد، مسیر دایرکتوری دیتاست PASCAL به عنوان ورودی به بخش سازنده کلاس PASCALDataset منتقل می‌شود. از آنجا که این کلاس برای بارگذاری دیتاست PASCAL طراحی شده است، با صدایزن هر کدام از اشیاء ایجاد شده، دیتاست متناظر با آن شیء بارگذاری می‌شود. در اینجا شد اولیه dataset برای بارگذاری دیتاست آموزشی، شیء dataset_val برای بارگذاری دیتاست اعتبارسنجی و شیء dataset_test برای بارگذاری دیتاست آزمون استفاده می‌شوند.

```

1 from pascal_dataset import PASCALDataset
2 # create dataset objects for training, validation, and testing sets
3 dataset = PASCALDataset('/content/PASCAL/PASCAL/train')
4 dataset_val = PASCALDataset('/content/PASCAL/PASCAL/val')
5 dataset_test = PASCALDataset('/content/PASCAL/PASCAL/test')
```

سپس، برای ایجاد DataLoader از ماژول torch.utils.data برای بارگیری داده‌ها و آماده سازی آن‌ها برای استفاده در

شبکه عصبی مورد استفاده قرار می‌گیرند. دستورات استفاده شده پارامترهای مختلفی دارد. پارامتر دیتابست یک شئ را از کلاس داده‌ها بارگیری می‌کند. پارامتر اندازه دسته، تعداد داده‌هایی که در هر دسته قرار می‌گیرند تعیین می‌کند. و پارامتر شافل، امکان اختلاط تصادفی داده‌ها را فراهم می‌آورد. در Faster RCNN، مدل با استفاده از تصاویر و برچسب‌های داده‌های آموزشی، برای تشخیص و شناسایی اشیاء در تصاویر آموزش دیده می‌شود. برای آموزش مدل، باید داده‌های آموزشی در قالب دسته‌ها (batch) و به صورت تصادفی (shuffle) به مدل داده شوند. به علاوه، برای تسريع فرآیند آموزش، می‌توان از چندین روند (process) به صورت موازی برای پردازش داده‌های آموزشی استفاده کرد. با استفاده از پارامتر num_workers، تعداد روندهای موازی برای بارگذاری داده‌ها تعیین می‌شود. با افزایش این پارامتر، تعداد روندهای موازی بیشتر می‌شود و در نتیجه، بارگذاری داده‌ها با سرعت بیشتری انجام می‌شود. با تعیین تابع collate_fn، روشی برای ترکیب داده‌های هر دسته تعیین می‌شود. در Faster RCNN، داده‌ها شامل تصاویر و برچسب‌های مربوط به شئ شناسایی شده در تصویر هستند. برای ترکیب داده‌های هر دسته، باید تصاویر و برچسب‌های مربوط به آنها به صورت جداگانه باشند. به عنوان مثال، برای هر دسته، تصاویر را به صورت یک لیست و برچسب‌های مربوط به هر تصویر را به صورت یک لیست دیگر تعريف می‌کنیم. در نهایت با استفاده از تابع collate_fn این دو لیست ترکیب شده و به صورت یک دسته به مدل داده می‌شوند. تابع utils.collate_fn نیز در کد مربوط به RCNN برای ترکیب داده‌های هر دسته استفاده می‌شود.

```

1 from torch.utils.data import DataLoader
2 # create data loader objects for training, validation, and testing sets
3 data_loader = torch.utils.data.DataLoader(dataset, batch_size=4, shuffle=True, num_workers=4,
4                                         collate_fn=utils.collate_fn)
5 data_loader_val = torch.utils.data.DataLoader(dataset_val, batch_size=4, shuffle=True,
6                                              num_workers=4, collate_fn=utils.collate_fn)
7 data_loader_test = torch.utils.data.DataLoader(dataset_test, batch_size=4, shuffle=True,
8                                               num_workers=4, collate_fn=utils.collate_fn)
```

در ادامه از مازول torchvision.models.detection برای بارگیری یک شبکه عصبی عمیق به نام Faster R-CNN با بکارگیری معماری ResNet-50-FPN پیش‌آموزش دیده استفاده می‌شود. ResNet-50-FPN یک شبکه عصبی عمیق است که توسط محققان شرکت مایکروسافت توسعه داده شده است. با استفاده از پارامتر pretrained=True مدل را به صورت پیش‌آموزش دیده بارگیری می‌کنیم.

```

1 # create an object of the Faster R-CNN model with ResNet50 as backbone and load pretrained
2 weights
model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=True)
```

در ادان دستوراتی برای تعريف یک بهینه‌ساز برای شبکه‌ی عصبی می‌نویسیم. برای این منظور، یک نمونه از کلاس بهینه‌ساز torch.optim.SGD با سه آرگومان ایجاد می‌شود. این سه آرگومان به ترتیب شامل مشخص کردن پارامترهایی هستند که می‌خواهیم در بهینه‌سازی از آنها استفاده کنیم. آرگومان model.parameters مشخص می‌کند که کدام پارامترهای مدل را قصد داریم بهینه‌سازی کنیم. در اینجا، تمام پارامترهای موجود در مدل شامل شبکه‌های پیچشی و غیره، انتخاب شده است. lr=0.001 هم یکی از مهم‌ترین پارامترهای بهینه‌سازی است که مربوط به نرخ یادگیری است. در اینجا، برای بهینه‌سازی مدل از نرخ یادگیری momentum=0.9 نشارنده میزان اطمینان به پارامترهای قبلی است. به طور خاص، مونتمون از ۰.۰۰۱ استفاده شده است. weight_decay=0.0005 شبیه پارامتر L2 Regularization در یادگیری عمیق است و معمولاً برای جلوگیری از بیش‌برازش استفاده می‌شود. در اینجا، برای بهینه‌سازی مدل، مقدار ۰.۰۰۰۵ برای این پارامتر در نظر گرفته شده است. هم‌چنین دستوراتی

به منظور انتخاب دستگاه مورد استفاده (CPU یا GPU) و انتقال مدل به آن دستگاه برای آموزش یا تست، می‌نویسیم. در اینجا، ابتدا با استفاده از تابع `torch.cuda.is_available` بررسی می‌شود که آیا سیستم دارای دستگاه GPU برای محاسبات است یا نه. اگر دستگاه GPU موجود بود، متغیر `device` برابر با `'cuda'` ('قرار می‌گیرد و در غیر این صورت برابر با `'cpu'`) قرار می‌گیرد. سپس، با استفاده از تابع `model.to(device)` مدل به دستگاه مناسب منتقل شده و در آموزش یا تست، محاسبات روی آن دستگاه انجام می‌شود. دستورات مذکور به شرح زیر است:

```

1 # create an optimizer object with stochastic gradient descent (SGD) algorithm
2 # and set learning rate, momentum, and weight decay values
3 optimizer = torch.optim.SGD(model.parameters(), lr=0.001, momentum=0.9, weight_decay=0.0005)
4
5 # move the model to the appropriate device (GPU if available, otherwise CPU)
6 device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
7 model.to(device)

```

نتیجه‌ای به صورت زیر حاصل می‌شود:

```

1 FasterRCNN(
2     (transform): GeneralizedRCNNTransform(
3         Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
4         Resize(min_size=(800,), max_size=1333, mode='bilinear')
5     )
6     (backbone): BackboneWithFPN(
7         (body): IntermediateLayerGetter(
8             (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
9             (bn1): FrozenBatchNorm2d(64, eps=0.0)
10            (relu): ReLU(inplace=True)
11            (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
12            (layer1): Sequential(
13                (0): Bottleneck(
14                    (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
15                    (bn1): FrozenBatchNorm2d(64, eps=0.0)
16                    (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
17                    (bn2): FrozenBatchNorm2d(64, eps=0.0)
18                    (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
19                    (bn3): FrozenBatchNorm2d(256, eps=0.0)
20                    (relu): ReLU(inplace=True)
21                    (downsample): Sequential(
22                        (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
23                        (1): FrozenBatchNorm2d(256, eps=0.0)
24                    )
25                )
26                (1): Bottleneck(
27                    (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
28                    (bn1): FrozenBatchNorm2d(64, eps=0.0)
29                    (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
30                    (bn2): FrozenBatchNorm2d(64, eps=0.0)
31                    (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
32                    (bn3): FrozenBatchNorm2d(256, eps=0.0)
33                    (relu): ReLU(inplace=True)
34                )
35                (2): Bottleneck(
36                    (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
37                    (bn1): FrozenBatchNorm2d(64, eps=0.0)
38                    (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
39                    (bn2): FrozenBatchNorm2d(64, eps=0.0)
40                    (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
41                    (bn3): FrozenBatchNorm2d(256, eps=0.0)
42                    (relu): ReLU(inplace=True)
43                )
44            )
45            (layer2): Sequential(
46                (0): Bottleneck(
47                    (conv1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)

```

```

48     (bn1): FrozenBatchNorm2d(128, eps=0.0)
49     (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
50     (bn2): FrozenBatchNorm2d(128, eps=0.0)
51     (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
52     (bn3): FrozenBatchNorm2d(512, eps=0.0)
53     (relu): ReLU(inplace=True)
54     (downsample): Sequential(
55         (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
56         (1): FrozenBatchNorm2d(512, eps=0.0)
57     )
58   )
59   (1): Bottleneck(
60     (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
61     (bn1): FrozenBatchNorm2d(128, eps=0.0)
62     (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
63     (bn2): FrozenBatchNorm2d(128, eps=0.0)
64     (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
65     (bn3): FrozenBatchNorm2d(512, eps=0.0)
66     (relu): ReLU(inplace=True)
67   )
68   (2): Bottleneck(
69     (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
70     (bn1): FrozenBatchNorm2d(128, eps=0.0)
71     (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
72     (bn2): FrozenBatchNorm2d(128, eps=0.0)
73     (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
74     (bn3): FrozenBatchNorm2d(512, eps=0.0)
75     (relu): ReLU(inplace=True)
76   )
77   (3): Bottleneck(
78     (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
79     (bn1): FrozenBatchNorm2d(128, eps=0.0)
80     (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
81     (bn2): FrozenBatchNorm2d(128, eps=0.0)
82     (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
83     (bn3): FrozenBatchNorm2d(512, eps=0.0)
84     (relu): ReLU(inplace=True)
85   )
86 )
87 (layer3): Sequential(
88   (0): Bottleneck(
89     (conv1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
90     (bn1): FrozenBatchNorm2d(256, eps=0.0)
91     (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
92     (bn2): FrozenBatchNorm2d(256, eps=0.0)
93     (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
94     (bn3): FrozenBatchNorm2d(1024, eps=0.0)
95     (relu): ReLU(inplace=True)
96     (downsample): Sequential(
97       (0): Conv2d(512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=False)
98       (1): FrozenBatchNorm2d(1024, eps=0.0)
99     )
100   )
101   (1): Bottleneck(
102     (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
103     (bn1): FrozenBatchNorm2d(256, eps=0.0)
104     (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
105     (bn2): FrozenBatchNorm2d(256, eps=0.0)
106     (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
107     (bn3): FrozenBatchNorm2d(1024, eps=0.0)
108     (relu): ReLU(inplace=True)
109   )
110   (2): Bottleneck(
111     (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
112     (bn1): FrozenBatchNorm2d(256, eps=0.0)
113     (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
114     (bn2): FrozenBatchNorm2d(256, eps=0.0)
115     (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
116     (bn3): FrozenBatchNorm2d(1024, eps=0.0)
117     (relu): ReLU(inplace=True)
118   )
119   (3): Bottleneck(

```

```

120     (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
121     (bn1): FrozenBatchNorm2d(256, eps=0.0)
122     (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
123     (bn2): FrozenBatchNorm2d(256, eps=0.0)
124     (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
125     (bn3): FrozenBatchNorm2d(1024, eps=0.0)
126     (relu): ReLU(inplace=True)
127   )
128   (4): Bottleneck(
129     (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
130     (bn1): FrozenBatchNorm2d(256, eps=0.0)
131     (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
132     (bn2): FrozenBatchNorm2d(256, eps=0.0)
133     (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
134     (bn3): FrozenBatchNorm2d(1024, eps=0.0)
135     (relu): ReLU(inplace=True)
136   )
137   (5): Bottleneck(
138     (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
139     (bn1): FrozenBatchNorm2d(256, eps=0.0)
140     (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
141     (bn2): FrozenBatchNorm2d(256, eps=0.0)
142     (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
143     (bn3): FrozenBatchNorm2d(1024, eps=0.0)
144     (relu): ReLU(inplace=True)
145   )
146   )
147   (layer4): Sequential(
148     (0): Bottleneck(
149       (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
150       (bn1): FrozenBatchNorm2d(512, eps=0.0)
151       (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
152       (bn2): FrozenBatchNorm2d(512, eps=0.0)
153       (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
154       (bn3): FrozenBatchNorm2d(2048, eps=0.0)
155       (relu): ReLU(inplace=True)
156       (downsample): Sequential(
157         (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False)
158         (1): FrozenBatchNorm2d(2048, eps=0.0)
159       )
160     )
161     (1): Bottleneck(
162       (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
163       (bn1): FrozenBatchNorm2d(512, eps=0.0)
164       (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
165       (bn2): FrozenBatchNorm2d(512, eps=0.0)
166       (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
167       (bn3): FrozenBatchNorm2d(2048, eps=0.0)
168       (relu): ReLU(inplace=True)
169     )
170     (2): Bottleneck(
171       (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
172       (bn1): FrozenBatchNorm2d(512, eps=0.0)
173       (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
174       (bn2): FrozenBatchNorm2d(512, eps=0.0)
175       (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
176       (bn3): FrozenBatchNorm2d(2048, eps=0.0)
177       (relu): ReLU(inplace=True)
178     )
179   )
180   )
181   (fpn): FeaturePyramidNetwork(
182     (inner_blocks): ModuleList(
183       (0): Conv2dNormActivation(
184         (0): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
185       )
186       (1): Conv2dNormActivation(
187         (0): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1))
188       )
189       (2): Conv2dNormActivation(
190         (0): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1))
191       )

```

```

192     (3): Conv2dNormActivation(
193         (0): Conv2d(2048, 256, kernel_size=(1, 1), stride=(1, 1))
194     )
195   )
196   (layer_blocks): ModuleList(
197     (0-3): 4 x Conv2dNormActivation(
198       (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
199     )
200   )
201   (extra_blocks): LastLevelMaxPool()
202 )
203 )
204 (rpn): RegionProposalNetwork(
205   (anchor_generator): AnchorGenerator()
206   (head): RPNHead(
207     (conv): Sequential(
208       (0): Conv2dNormActivation(
209         (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
210         (1): ReLU(inplace=True)
211       )
212     )
213     (cls_logits): Conv2d(256, 3, kernel_size=(1, 1), stride=(1, 1))
214     (bbox_pred): Conv2d(256, 12, kernel_size=(1, 1), stride=(1, 1))
215   )
216 )
217 (roi_heads): RoIHeads(
218   (box_roi_pool): MultiScaleRoIAlign(featmap_names=['0', '1', '2', '3'], output_size=(7, 7), sampling_ratio=2)
219   (box_head): TwoMLPHead(
220     (fc6): Linear(in_features=12544, out_features=1024, bias=True)
221     (fc7): Linear(in_features=1024, out_features=1024, bias=True)
222   )
223   (box_predictor): FastRCNNPredictor(
224     (cls_score): Linear(in_features=1024, out_features=91, bias=True)
225     (bbox_pred): Linear(in_features=1024, out_features=364, bias=True)
226   )
227 )
228 )

```

با فراهم کردن تمام مقدمات لازم، نوبت به نوشتن کد اصلی مربوط به آموزش مدل می‌رسد. در این کد، ابتدا کتابخانه‌های لازم ایمپورت شده است. در خطوط بعدی، داده‌های آموزش و اعتبارسنجی به دو صورت جداگانه با استفاده از کلاس PASCALDataset ساخته شده‌اند و سپس با استفاده از تابع `torch.utils.data.DataLoader`، داده‌ها به صورت دسته‌بندی شده و برای آموزش و اعتبارسنجی در `train_loader` و `val_loader` مرتب شده‌اند. سپس در خطوط بعدی، تعداد ایپاک‌ها و فرکانس نمایش وضعیت در هر ایپاک مشخص شده‌اند. در بلوک اصلی کد، یک حلقه برای تعداد ایپاک اجرا شده است. در هر ایپاک، با استفاده از تابع `train_one_epoch`، مدل با داده‌های آموزشی آموزش داده می‌شود. این تابع موارد متنوعی را به عنوان آرگومان می‌پذیرد که در توضیحات مربوط به فایل کد مرتبط با آن شرح داده شد. در خط بعد، با استفاده از تابع `evaluate`، عملکرد مدل با داده‌های اعتبارسنجی بررسی می‌شود. دستورات به شرح زیر است:

```

1 import torch
2 from pascal_dataset import PASCALDataset
3 import utils
4
5 # Create instances of PASCALDataset for training and validation datasets
6 train_dataset = PASCALDataset('/content/PASCAL/PASCAL/train')
7 val_dataset = PASCALDataset('/content/PASCAL/PASCAL/val')
8
9 # Create data loaders for the training and validation datasets
10 train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=4, shuffle=True, num_workers
11                                         =4, collate_fn=utils.collate_fn)

```

```

11 val_loader = torch.utils.data.DataLoader(val_dataset, batch_size=4, shuffle=False, num_workers=4,
12                                         collate_fn=utils.collate_fn)
13
14 # Define number of epochs and frequency of status updates during training
15 epoch = 5
16 print_freq = 25
17
18 # Train and validate the model for the specified number of epochs
19 for epoch in range(epoch):
20     # Train the model for one epoch using the train_loader
21     train_one_epoch(model, optimizer, train_loader, device, epoch, print_freq, scaler=None)
22
23     # Evaluate the model using the val_loader
24     evaluate(model, val_loader, device)

```

این دستور در دفعات مختلفی اجرا شده و نتایج متنوعی حاصل شده که در این لینک گوگل کولب قابل مشاهده است. نتیجه یکی از این اجرایها به شرح زیر بوده است (برنامه ۱):

Program 1: One of the Training Results

```

1 Epoch: [0] [ 0/251] eta: 0:34:00 lr: 0.000005 loss: 0.2050 (0.2050) loss_classifier: 0.1115 (0.1115) loss_box_reg: 0.0562 (0.0562)
    loss_objectness: 0.0094 (0.0094) loss_rpn_box_reg: 0.0280 (0.0280) time: 8.1288 data: 0.2454 max mem: 4010
2 Epoch: [0] [ 25/251] eta: 0:05:49 lr: 0.000105 loss: 0.1778 (0.1885) loss_classifier: 0.0660 (0.0754) loss_box_reg: 0.0816 (0.0937)
    loss_objectness: 0.0041 (0.0070) loss_rpn_box_reg: 0.0103 (0.0125) time: 1.2738 data: 0.0115 max mem: 7462
3 Epoch: [0] [ 50/251] eta: 0:04:46 lr: 0.000205 loss: 0.1719 (0.1966) loss_classifier: 0.0699 (0.0782) loss_box_reg: 0.0980 (0.0976)
    loss_objectness: 0.0057 (0.0074) loss_rpn_box_reg: 0.0130 (0.0134) time: 1.2929 data: 0.0119 max mem: 7462
4 Epoch: [0] [ 75/251] eta: 0:04:07 lr: 0.000305 loss: 0.1980 (0.1998) loss_classifier: 0.0751 (0.0772) loss_box_reg: 0.0968 (0.0998)
    loss_objectness: 0.0052 (0.0085) loss_rpn_box_reg: 0.0142 (0.0143) time: 1.3584 data: 0.0121 max mem: 7462
5 Epoch: [0] [100/251] eta: 0:03:30 lr: 0.000405 loss: 0.1992 (0.2034) loss_classifier: 0.0775 (0.0784) loss_box_reg: 0.1050 (0.1024)
    loss_objectness: 0.0060 (0.0081) loss_rpn_box_reg: 0.0087 (0.0145) time: 1.2990 data: 0.0125 max mem: 7462
6 Epoch: [0] [125/251] eta: 0:02:53 lr: 0.000504 loss: 0.1839 (0.2007) loss_classifier: 0.0595 (0.0777) loss_box_reg: 0.0865 (0.1005)
    loss_objectness: 0.0040 (0.0078) loss_rpn_box_reg: 0.0117 (0.0147) time: 1.3562 data: 0.0126 max mem: 7462
7 Epoch: [0] [150/251] eta: 0:02:18 lr: 0.000604 loss: 0.1801 (0.2036) loss_classifier: 0.0586 (0.0773) loss_box_reg: 0.0906 (0.1035)
    loss_objectness: 0.0047 (0.0076) loss_rpn_box_reg: 0.0171 (0.0151) time: 1.3262 data: 0.0119 max mem: 7462
8 Epoch: [0] [175/251] eta: 0:01:44 lr: 0.000704 loss: 0.1660 (0.2040) loss_classifier: 0.0611 (0.0764) loss_box_reg: 0.0851 (0.1040)
    loss_objectness: 0.0057 (0.0085) loss_rpn_box_reg: 0.0098 (0.0150) time: 1.3977 data: 0.0116 max mem: 7462
9 Epoch: [0] [200/251] eta: 0:01:09 lr: 0.000804 loss: 0.1929 (0.2071) loss_classifier: 0.0703 (0.0765) loss_box_reg: 0.1143 (0.1064)
    loss_objectness: 0.0053 (0.0088) loss_rpn_box_reg: 0.0126 (0.0154) time: 1.3074 data: 0.0121 max mem: 7462
10 Epoch: [0] [225/251] eta: 0:00:35 lr: 0.000904 loss: 0.1960 (0.2068) loss_classifier: 0.0756 (0.0763) loss_box_reg: 0.1109 (0.1068)
    loss_objectness: 0.0050 (0.0085) loss_rpn_box_reg: 0.0091 (0.0152) time: 1.3699 data: 0.0118 max mem: 7462
11 Epoch: [0] [250/251] eta: 0:00:01 lr: 0.001000 loss: 0.1881 (0.2070) loss_classifier: 0.0625 (0.0761) loss_box_reg: 0.1100 (0.1072)
    loss_objectness: 0.0039 (0.0084) loss_rpn_box_reg: 0.0102 (0.0153) time: 1.3226 data: 0.0113 max mem: 7462
12 Epoch: [0] Total time: 0:05:41 (1.3609 s / it)
13 creating index...
14 index created!
15 Test: [ 0/50] eta: 0:00:47 model_time: 0.6678 (0.6678) evaluator_time: 0.0072 (0.0072) time: 0.9581 data: 0.2794 max mem: 7462
16 Test: [49/50] eta: 0:00:00 model_time: 0.6150 (0.5549) evaluator_time: 0.0053 (0.0062) time: 0.5946 data: 0.0100 max mem: 7462
17 Test: Total time: 0:00:29 (0.5803 s / it)
18 Averaged stats: model_time: 0.6150 (0.5549) evaluator_time: 0.0053 (0.0062)
19 Accumulating evaluation results...
20 DONE (t=0.10s).
21 IoU metric: bbox
22 Average Precision (AP) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.360
23 Average Precision (AP) @[ IoU=0.50 | area= all | maxDets=100 ] = 0.536
24 Average Precision (AP) @[ IoU=0.75 | area= all | maxDets=100 ] = 0.394
25 Average Precision (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.195
26 Average Precision (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.376
27 Average Precision (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.411
28 Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 1 ] = 0.323
29 Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 10 ] = 0.521
30 Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.528
31 Average Recall (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.299
32 Average Recall (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.543

```

```

33 Average Recall (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.559
34 Epoch: [1] [ 0/251] eta: 0:05:47 lr: 0.001000 loss: 0.1819 (0.1819) loss_classifier: 0.0523 (0.0523) loss_box_reg: 0.1134 (0.1134)
    loss_objectness: 0.0059 (0.0059) loss_rpn_box_reg: 0.0103 (0.0103) time: 1.3851 data: 0.2944 max mem: 7462
35 Epoch: [1] [ 25/251] eta: 0:05:12 lr: 0.001000 loss: 0.1808 (0.1964) loss_classifier: 0.0605 (0.0673) loss_box_reg: 0.0822 (0.1077)
    loss_objectness: 0.0047 (0.0095) loss_rpn_box_reg: 0.0095 (0.0118) time: 1.3682 data: 0.0123 max mem: 7462
36 Epoch: [1] [ 50/251] eta: 0:04:37 lr: 0.001000 loss: 0.1622 (0.1884) loss_classifier: 0.0553 (0.0653) loss_box_reg: 0.0895 (0.1016)
    loss_objectness: 0.0039 (0.0079) loss_rpn_box_reg: 0.0119 (0.0135) time: 1.3625 data: 0.0124 max mem: 7462
37 Epoch: [1] [ 75/251] eta: 0:03:57 lr: 0.001000 loss: 0.1375 (0.1776) loss_classifier: 0.0423 (0.0614) loss_box_reg: 0.0910 (0.0966)
    loss_objectness: 0.0038 (0.0067) loss_rpn_box_reg: 0.0101 (0.0129) time: 1.2513 data: 0.0118 max mem: 7462
38 Epoch: [1] [100/251] eta: 0:03:23 lr: 0.001000 loss: 0.1819 (0.1791) loss_classifier: 0.0563 (0.0611) loss_box_reg: 0.0919 (0.0979)
    loss_objectness: 0.0037 (0.0062) loss_rpn_box_reg: 0.0165 (0.0139) time: 1.3319 data: 0.0120 max mem: 7462
39 Epoch: [1] [125/251] eta: 0:02:50 lr: 0.001000 loss: 0.1866 (0.1849) loss_classifier: 0.0580 (0.0624) loss_box_reg: 0.1111 (0.1018)
    loss_objectness: 0.0041 (0.0060) loss_rpn_box_reg: 0.0154 (0.0147) time: 1.3312 data: 0.0120 max mem: 7462
40 Epoch: [1] [150/251] eta: 0:02:16 lr: 0.001000 loss: 0.1715 (0.1868) loss_classifier: 0.0584 (0.0627) loss_box_reg: 0.0949 (0.1036)
    loss_objectness: 0.0032 (0.0056) loss_rpn_box_reg: 0.0130 (0.0149) time: 1.3913 data: 0.0123 max mem: 7462
41 Epoch: [1] [175/251] eta: 0:01:43 lr: 0.001000 loss: 0.1414 (0.1861) loss_classifier: 0.0489 (0.0623) loss_box_reg: 0.0819 (0.1040)
    loss_objectness: 0.0034 (0.0054) loss_rpn_box_reg: 0.0097 (0.0145) time: 1.3832 data: 0.0120 max mem: 7462
42 Epoch: [1] [200/251] eta: 0:01:09 lr: 0.001000 loss: 0.1093 (0.1855) loss_classifier: 0.0337 (0.0619) loss_box_reg: 0.0630 (0.1033)
    loss_objectness: 0.0018 (0.0055) loss_rpn_box_reg: 0.0123 (0.0148) time: 1.3848 data: 0.0121 max mem: 7462
43 Epoch: [1] [225/251] eta: 0:00:35 lr: 0.001000 loss: 0.1398 (0.1833) loss_classifier: 0.0490 (0.0616) loss_box_reg: 0.0791 (0.1019)
    loss_objectness: 0.0026 (0.0053) loss_rpn_box_reg: 0.0107 (0.0145) time: 1.3098 data: 0.0121 max mem: 7462
44 Epoch: [1] [250/251] eta: 0:00:01 lr: 0.001000 loss: 0.1151 (0.1815) loss_classifier: 0.0364 (0.0608) loss_box_reg: 0.0682 (0.1010)
    loss_objectness: 0.0020 (0.0052) loss_rpn_box_reg: 0.0086 (0.0145) time: 1.3289 data: 0.0118 max mem: 7462
45 Epoch: [1] Total time: 0:05:39 (1.3533 s / it)
46 creating index...
47 index created!
48 Test: [ 0/50] eta: 0:00:48 model_time: 0.6769 (0.6769) evaluator_time: 0.0060 (0.0060) time: 0.9658 data: 0.2795 max mem: 7462
49 Test: [49/50] eta: 0:00:00 model_time: 0.6173 (0.5554) evaluator_time: 0.0051 (0.0058) time: 0.5948 data: 0.0108 max mem: 7462
50 Test: Total time: 0:00:29 (0.5813 s / it)
51 Averaged stats: model_time: 0.6173 (0.5554) evaluator_time: 0.0051 (0.0058)
52 Accumulating evaluation results...
53 DONE (t=0.09s).
54 IoU metric: bbox
55 Average Precision (AP) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.355
56 Average Precision (AP) @[ IoU=0.50 | area= all | maxDets=100 ] = 0.537
57 Average Precision (AP) @[ IoU=0.75 | area= all | maxDets=100 ] = 0.399
58 Average Precision (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.184
59 Average Precision (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.375
60 Average Precision (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.410
61 Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 1 ] = 0.321
62 Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 10 ] = 0.511
63 Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.520
64 Average Recall (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.275
65 Average Recall (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.540
66 Average Recall (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.556
67 Epoch: [2] [ 0/251] eta: 0:06:58 lr: 0.001000 loss: 0.1251 (0.1251) loss_classifier: 0.0535 (0.0535) loss_box_reg: 0.0579 (0.0579)
    loss_objectness: 0.0020 (0.0020) loss_rpn_box_reg: 0.0117 (0.0117) time: 1.6659 data: 0.2627 max mem: 7462
68 Epoch: [2] [ 25/251] eta: 0:04:59 lr: 0.001000 loss: 0.1204 (0.1388) loss_classifier: 0.0339 (0.0444) loss_box_reg: 0.0718 (0.0793)
    loss_objectness: 0.0026 (0.0033) loss_rpn_box_reg: 0.0096 (0.0118) time: 1.2912 data: 0.0119 max mem: 7462
69 Epoch: [2] [ 50/251] eta: 0:04:27 lr: 0.001000 loss: 0.1441 (0.1456) loss_classifier: 0.0480 (0.0474) loss_box_reg: 0.0815 (0.0832)
    loss_objectness: 0.0013 (0.0027) loss_rpn_box_reg: 0.0112 (0.0123) time: 1.3421 data: 0.0121 max mem: 7462
70 Epoch: [2] [ 75/251] eta: 0:03:56 lr: 0.001000 loss: 0.1794 (0.1572) loss_classifier: 0.0512 (0.0504) loss_box_reg: 0.1067 (0.0896)
    loss_objectness: 0.0023 (0.0032) loss_rpn_box_reg: 0.0160 (0.0140) time: 1.3772 data: 0.0125 max mem: 7462
71 Epoch: [2] [100/251] eta: 0:03:25 lr: 0.001000 loss: 0.1368 (0.1578) loss_classifier: 0.0426 (0.0509) loss_box_reg: 0.0700 (0.0896)
    loss_objectness: 0.0023 (0.0033) loss_rpn_box_reg: 0.0128 (0.0139) time: 1.4147 data: 0.0119 max mem: 7649
72 Epoch: [2] [125/251] eta: 0:02:51 lr: 0.001000 loss: 0.1587 (0.1629) loss_classifier: 0.0481 (0.0520) loss_box_reg: 0.0908 (0.0935)
    loss_objectness: 0.0034 (0.0035) loss_rpn_box_reg: 0.0120 (0.0140) time: 1.3598 data: 0.0120 max mem: 7649
73 Epoch: [2] [150/251] eta: 0:02:16 lr: 0.001000 loss: 0.1467 (0.1604) loss_classifier: 0.0494 (0.0516) loss_box_reg: 0.0880 (0.0920)
    loss_objectness: 0.0025 (0.0033) loss_rpn_box_reg: 0.0097 (0.0135) time: 1.2869 data: 0.0115 max mem: 7649
74 Epoch: [2] [175/251] eta: 0:01:42 lr: 0.001000 loss: 0.1090 (0.1602) loss_classifier: 0.0339 (0.0513) loss_box_reg: 0.0564 (0.0918)
    loss_objectness: 0.0030 (0.0034) loss_rpn_box_reg: 0.0133 (0.0138) time: 1.3759 data: 0.0131 max mem: 8203
75 Epoch: [2] [200/251] eta: 0:01:09 lr: 0.001000 loss: 0.1204 (0.1582) loss_classifier: 0.0389 (0.0507) loss_box_reg: 0.0785 (0.0906)
    loss_objectness: 0.0018 (0.0034) loss_rpn_box_reg: 0.0091 (0.0134) time: 1.4777 data: 0.0120 max mem: 8203
76 Epoch: [2] [225/251] eta: 0:00:35 lr: 0.001000 loss: 0.1137 (0.1556) loss_classifier: 0.0369 (0.0498) loss_box_reg: 0.0653 (0.0891)
    loss_objectness: 0.0024 (0.0033) loss_rpn_box_reg: 0.0137 (0.0134) time: 1.2978 data: 0.0121 max mem: 8203
77 Epoch: [2] [250/251] eta: 0:00:01 lr: 0.001000 loss: 0.1696 (0.1588) loss_classifier: 0.0517 (0.0507) loss_box_reg: 0.0987 (0.0912)
    loss_objectness: 0.0030 (0.0034) loss_rpn_box_reg: 0.0107 (0.0135) time: 1.2682 data: 0.0123 max mem: 8203
78 Epoch: [2] Total time: 0:05:37 (1.3463 s / it)
79 creating index...
80 index created!
81 Test: [ 0/50] eta: 0:00:48 model_time: 0.6667 (0.6667) evaluator_time: 0.0046 (0.0046) time: 0.9712 data: 0.2960 max mem: 8203
82 Test: [49/50] eta: 0:00:00 model_time: 0.6129 (0.5547) evaluator_time: 0.0047 (0.0055) time: 0.5942 data: 0.0113 max mem: 8203

```

```

83 Test: Total time: 0:00:29 (0.5805 s / it)
84 Averaged stats: model_time: 0.6129 (0.5547) evaluator_time: 0.0047 (0.0055)
85 Accumulating evaluation results...
86 DONE (t=0.09s).
87 IoU metric: bbox
88 Average Precision (AP) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.347
89 Average Precision (AP) @[ IoU=0.50 | area= all | maxDets=100 ] = 0.540
90 Average Precision (AP) @[ IoU=0.75 | area= all | maxDets=100 ] = 0.391
91 Average Precision (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.177
92 Average Precision (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.362
93 Average Precision (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.401
94 Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 1 ] = 0.313
95 Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 10 ] = 0.498
96 Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.506
97 Average Recall (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.276
98 Average Recall (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.522
99 Average Recall (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.540
100 Epoch: [3] [ 0/251] eta: 0:05:43 lr: 0.001000 loss: 0.1269 (0.1269) loss_classifier: 0.0517 (0.0517) loss_box_reg: 0.0679 (0.0679)
    loss_objectness: 0.0021 (0.0021) loss_rpn_box_reg: 0.0052 (0.0052) time: 1.3697 data: 0.2800 max mem: 8203
101 Epoch: [3] [ 25/251] eta: 0:05:06 lr: 0.001000 loss: 0.1497 (0.1446) loss_classifier: 0.0439 (0.0447) loss_box_reg: 0.0915 (0.0841)
    loss_objectness: 0.0024 (0.0029) loss_rpn_box_reg: 0.0132 (0.0129) time: 1.3719 data: 0.0138 max mem: 8203
102 Epoch: [3] [ 50/251] eta: 0:04:32 lr: 0.001000 loss: 0.1319 (0.1558) loss_classifier: 0.0409 (0.0474) loss_box_reg: 0.0722 (0.0912)
    loss_objectness: 0.0030 (0.0032) loss_rpn_box_reg: 0.0141 (0.0141) time: 1.3673 data: 0.0119 max mem: 8203
103 Epoch: [3] [ 75/251] eta: 0:04:01 lr: 0.001000 loss: 0.1253 (0.1488) loss_classifier: 0.0408 (0.0455) loss_box_reg: 0.0667 (0.0867)
    loss_objectness: 0.0023 (0.0030) loss_rpn_box_reg: 0.0110 (0.0136) time: 1.3673 data: 0.0125 max mem: 8203
104 Epoch: [3] [100/251] eta: 0:03:25 lr: 0.001000 loss: 0.1130 (0.1490) loss_classifier: 0.0333 (0.0452) loss_box_reg: 0.0596 (0.0864)
    loss_objectness: 0.0018 (0.0030) loss_rpn_box_reg: 0.0121 (0.0143) time: 1.3311 data: 0.0117 max mem: 8203
105 Epoch: [3] [125/251] eta: 0:02:50 lr: 0.001000 loss: 0.1203 (0.1451) loss_classifier: 0.0337 (0.0442) loss_box_reg: 0.0694 (0.0844)
    loss_objectness: 0.0022 (0.0031) loss_rpn_box_reg: 0.0062 (0.0134) time: 1.3097 data: 0.0125 max mem: 8203
106 Epoch: [3] [150/251] eta: 0:02:16 lr: 0.001000 loss: 0.1228 (0.1453) loss_classifier: 0.0367 (0.0448) loss_box_reg: 0.0740 (0.0842)
    loss_objectness: 0.0023 (0.0031) loss_rpn_box_reg: 0.0087 (0.0132) time: 1.3367 data: 0.0126 max mem: 8203
107 Epoch: [3] [175/251] eta: 0:01:42 lr: 0.001000 loss: 0.1168 (0.1448) loss_classifier: 0.0370 (0.0442) loss_box_reg: 0.0755 (0.0841)
    loss_objectness: 0.0026 (0.0031) loss_rpn_box_reg: 0.0111 (0.0133) time: 1.3260 data: 0.0125 max mem: 8203
108 Epoch: [3] [200/251] eta: 0:01:08 lr: 0.001000 loss: 0.1323 (0.1471) loss_classifier: 0.0416 (0.0450) loss_box_reg: 0.0787 (0.0857)
    loss_objectness: 0.0023 (0.0032) loss_rpn_box_reg: 0.0106 (0.0132) time: 1.2724 data: 0.0122 max mem: 8203
109 Epoch: [3] [225/251] eta: 0:00:34 lr: 0.001000 loss: 0.1208 (0.1448) loss_classifier: 0.0340 (0.0444) loss_box_reg: 0.0699 (0.0841)
    loss_objectness: 0.0031 (0.0032) loss_rpn_box_reg: 0.0114 (0.0132) time: 1.3085 data: 0.0118 max mem: 8203
110 Epoch: [3] [250/251] eta: 0:00:01 lr: 0.001000 loss: 0.1237 (0.1431) loss_classifier: 0.0332 (0.0439) loss_box_reg: 0.0720 (0.0830)
    loss_objectness: 0.0018 (0.0031) loss_rpn_box_reg: 0.0085 (0.0130) time: 1.3125 data: 0.0119 max mem: 8203
111 Epoch: [3] Total time: 0:05:36 (1.3412 s / it)
112 creating index...
113 index created!
114 Test: [ 0/50] eta: 0:00:48 model_time: 0.6662 (0.6662) evaluator_time: 0.0041 (0.0041) time: 0.9696 data: 0.2955 max mem: 8203
115 Test: [ 49/50] eta: 0:00:00 model_time: 0.6162 (0.5551) evaluator_time: 0.0046 (0.0052) time: 0.5934 data: 0.0105 max mem: 8203
116 Test: Total time: 0:00:29 (0.5813 s / it)
117 Averaged stats: model_time: 0.6162 (0.5551) evaluator_time: 0.0046 (0.0052)
118 Accumulating evaluation results...
119 DONE (t=0.08s).
120 IoU metric: bbox
121 Average Precision (AP) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.350
122 Average Precision (AP) @[ IoU=0.50 | area= all | maxDets=100 ] = 0.533
123 Average Precision (AP) @[ IoU=0.75 | area= all | maxDets=100 ] = 0.398
124 Average Precision (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.183
125 Average Precision (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.369
126 Average Precision (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.407
127 Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 1 ] = 0.318
128 Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 10 ] = 0.499
129 Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.506
130 Average Recall (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.262
131 Average Recall (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.531
132 Average Recall (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.541
133 Epoch: [4] [ 0/251] eta: 0:07:21 lr: 0.001000 loss: 0.0458 (0.0458) loss_classifier: 0.0173 (0.0173) loss_box_reg: 0.0256 (0.0256)
    loss_objectness: 0.0012 (0.0012) loss_rpn_box_reg: 0.0017 (0.0017) time: 1.7579 data: 0.3121 max mem: 8203
134 Epoch: [4] [ 25/251] eta: 0:04:57 lr: 0.001000 loss: 0.0958 (0.1144) loss_classifier: 0.0299 (0.0336) loss_box_reg: 0.0605 (0.0680)
    loss_objectness: 0.0018 (0.0025) loss_rpn_box_reg: 0.0068 (0.0102) time: 1.2898 data: 0.0113 max mem: 8203
135 Epoch: [4] [ 50/251] eta: 0:04:27 lr: 0.001000 loss: 0.1015 (0.1193) loss_classifier: 0.0341 (0.0357) loss_box_reg: 0.0627 (0.0709)
    loss_objectness: 0.0013 (0.0022) loss_rpn_box_reg: 0.0081 (0.0104) time: 1.3603 data: 0.0124 max mem: 8203
136 Epoch: [4] [ 75/251] eta: 0:03:54 lr: 0.001000 loss: 0.1381 (0.1309) loss_classifier: 0.0469 (0.0395) loss_box_reg: 0.0768 (0.0774)
    loss_objectness: 0.0025 (0.0024) loss_rpn_box_reg: 0.0096 (0.0116) time: 1.3027 data: 0.0123 max mem: 8203
137 Epoch: [4] [100/251] eta: 0:03:22 lr: 0.001000 loss: 0.1096 (0.1307) loss_classifier: 0.0368 (0.0391) loss_box_reg: 0.0623 (0.0775)
    loss_objectness: 0.0012 (0.0022) loss_rpn_box_reg: 0.0111 (0.0118) time: 1.3742 data: 0.0115 max mem: 8203
138 Epoch: [4] [125/251] eta: 0:02:47 lr: 0.001000 loss: 0.1147 (0.1319) loss_classifier: 0.0295 (0.0393) loss_box_reg: 0.0630 (0.0782)

```

```

loss_objectness: 0.0013 (0.0023) loss_rpn_box_reg: 0.0092 (0.0121) time: 1.2883 data: 0.0118 max mem: 8203
139 Epoch: [4] [150/251] eta: 0:02:14 lr: 0.001000 loss: 0.1108 (0.1322) loss_classifier: 0.0331 (0.0396) loss_box_reg: 0.0692 (0.0781)
loss_objectness: 0.0019 (0.0023) loss_rpn_box_reg: 0.0112 (0.0121) time: 1.3979 data: 0.0126 max mem: 8203
140 Epoch: [4] [175/251] eta: 0:01:42 lr: 0.001000 loss: 0.1144 (0.1337) loss_classifier: 0.0419 (0.0408) loss_box_reg: 0.0645 (0.0786)
loss_objectness: 0.0015 (0.0023) loss_rpn_box_reg: 0.0091 (0.0120) time: 1.4035 data: 0.0121 max mem: 8203
141 Epoch: [4] [200/251] eta: 0:01:08 lr: 0.001000 loss: 0.0862 (0.1320) loss_classifier: 0.0303 (0.0403) loss_box_reg: 0.0463 (0.0771)
loss_objectness: 0.0015 (0.0023) loss_rpn_box_reg: 0.0097 (0.0123) time: 1.3308 data: 0.0119 max mem: 8203
142 Epoch: [4] [225/251] eta: 0:00:34 lr: 0.001000 loss: 0.1135 (0.1306) loss_classifier: 0.0335 (0.0398) loss_box_reg: 0.0633 (0.0762)
loss_objectness: 0.0014 (0.0023) loss_rpn_box_reg: 0.0095 (0.0122) time: 1.3172 data: 0.0120 max mem: 8203
143 Epoch: [4] [250/251] eta: 0:00:01 lr: 0.001000 loss: 0.1063 (0.1306) loss_classifier: 0.0325 (0.0397) loss_box_reg: 0.0634 (0.0761)
loss_objectness: 0.0015 (0.0023) loss_rpn_box_reg: 0.0140 (0.0124) time: 1.3267 data: 0.0114 max mem: 8203
144 Epoch: [4] Total time: 0:05:35 (1.3384 s / it)
145 creating index...
146 index created!
147 Test: [ 0/50] eta: 0:00:48 model_time: 0.6714 (0.6714) evaluator_time: 0.0038 (0.0038) time: 0.9613 data: 0.2819 max mem: 8203
148 Test: [49/50] eta: 0:00:00 model_time: 0.6147 (0.5550) evaluator_time: 0.0045 (0.0049) time: 0.5928 data: 0.0114 max mem: 8203
149 Test: Total time: 0:00:29 (0.5812 s / it)
150 Averaged stats: model_time: 0.6147 (0.5550) evaluator_time: 0.0045 (0.0049)
151 Accumulating evaluation results...
152 DONE (t=0.07s).
153 IoU metric: bbox
154 Average Precision (AP) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.351
155 Average Precision (AP) @[ IoU=0.50 | area= all | maxDets=100 ] = 0.527
156 Average Precision (AP) @[ IoU=0.75 | area= all | maxDets=100 ] = 0.397
157 Average Precision (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.181
158 Average Precision (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.377
159 Average Precision (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.413
160 Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 1 ] = 0.323
161 Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 10 ] = 0.504
162 Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.510
163 Average Recall (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.258
164 Average Recall (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.536
165 Average Recall (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.548

```

در انتهای هم دستوراتی برای ذخیره و انتقال مدل آموزش دیده شده به دستگاه نوشته شده‌اند. در دستور اول، با استفاده از تابع `torch.save`، وضعیت مدل به عنوان یک فایل با نام `model1.pth` در محل جاری ذخیره می‌شود. وضعیت مدل شامل پارامترهای آموزش داده شده است که در ادامه برای اجرای مدل بر روی داده‌های جدید مورد استفاده قرار می‌گیرد. در دستور دوم، با استفاده از تابع `shutil.copyfile`، فایل مدل ذخیره شده را از مسیر جاری به یک مسیر دیگر در درایو گوگل منتقل می‌کند. برای این منظور، مسیر فایل مدل و مسیر جدیدی که می‌خواهیم فایل را در آنجا قرار دهیم را به تابع `shutil.copyfile` ارسال می‌کنیم.

```

1 import torch
2 import shutil
3
4 # Save the state dictionary of the trained model as a file named 'model1.pth'
5 torch.save(model.state_dict(), 'model1.pth')
6
7 # Copy the saved model file from its current location to Google Drive
8 shutil.copyfile('/content/model1.pth', '/content/drive/My Drive/model1.pth')

```

۳.۱ پاسخ قسمت ۳ - ارزیابی

در این قسمت دستوراتی را با هدف آزمایش ساختار و مدل آموزش دیده روی داده‌های ارزیابی (تست) می‌نویسیم. در این بخش از دیتالودر مربوط به دیتابست ارزیابی (تست) استفاده شده است تا داده‌های تست به شبکه داده شود و شبکه با استفاده از این داده‌ها و مدل آموزش دیده شده، نتیجه پیش‌بینی خود را به دست آورده و مقدار خطای خود را محاسبه کند. ابتدا در هر

یک از ایاک‌ها، داده‌ها و برچسب‌های متناظر با داده‌ها به دو صورت تصویر و هدف از دیتالودر گرفته می‌شوند. برای سرعت بخشیدن به محاسبات، تصاویر موجود در `image` و برچسب‌های موجود در `target` روی دستگاهی که پردازش سریع‌تری را فراهم می‌کند مانند GPU انتقال داده می‌شوند. به این منظور، ابتدا با استفاده از دستور `image.to(device)` تصاویر به دستگاه مورد نظر انتقال داده می‌شوند. به منظور اینکه تمامی داده‌های ورودی به شبکه نیز روی دستگاه مورد نظر باشند، برچسب‌های مربوط به هر تصویر را نیز با استفاده از دستورات نوشته شده به دستگاه مورد نظر منتقل می‌کنیم. با انتقال تصاویر و برچسب‌های مربوط به دستگاه مورد نظر، شبکه با استفاده از تصاویر و برچسب‌های ورودی، خروجی خود را بدست می‌آورد و خطای خود را با محاسبه تفاضل بین خروجی و برچسب واقعی محاسبه می‌کند و به شکل یک دیکشنری به نام `loss_dict` برمی‌گرداند. در ادامه نوع معیار IoU برای ارزیابی تعیین می‌شود که در اینجا تنها از "bbox" استفاده می‌شود. سپس با استفاده از مدل آموزش دیده شده، پیش‌بینی برای تصاویر ورودی ایجاد می‌شود. سپس یک دیکشنری ساخته می‌شود که برای هر شناسه تصویر، خروجی پیش‌بینی مربوط به آن به عنوان مقدار دیکشنری ذخیره می‌شود. در نهایت، مقدار هر دیکشنری، به عنوان ورودی به شئ ارزیابی COCO ارسال شده و معیار IoU برای تصاویر پیش‌بینی شده محاسبه می‌شود. در انتها دستوراتی برای ارزیابی دقت الگوریتم شبکه‌های عصبی می‌نویسیم. در این بخش از کد، نتایج پیش‌بینی شده با استفاده از مدل شبکه عصبی را در حافظه روی کلاس `CocoEvaluator` ذخیره می‌کنیم. `synchronize_between_processes` نتایج محاسباتی در حال انجام در هر فرایند را همگام می‌کند و `accumulate` نتایج پیش‌بینی شده را جمع‌آوری می‌کند تا بتوانیم دقت الگوریتم را برای تمام داده‌های ارزیابی به دست آوریم. `summarize` هم نتایج پیش‌بینی را مورد بررسی قرار می‌دهد و نتایج را خلاصه کرده و نمایش می‌دهد. دستورات به شرح زیر است:

```

1 for images, targets in data_loader_test:
2     # Move images to the specified device (e.g. GPU) for faster computation
3     images = list(image.to(device) for image in images)
4
5     # Move target annotations to the same device
6     targets = [{k: v.to(device) for k, v in t.items()} for t in targets]
7
8     # Forward pass of the model to obtain the loss dictionary
9     loss_dict = model(images, targets)
10
11
12 # Import the necessary library functions
13 from torchvision.datasets import CocoDetection
14 from torchvision.datasets.coco import get_coco_api_from_dataset
15
16 # Get the COCO API from the test dataset
17 coco = get_coco_api_from_dataset(data_loader_test.dataset)
18
19
20 # Define the IoU type(s) for evaluation (here, only "bbox" is used)
21 iou_types = ["bbox"]
22
23 # Instantiate the COCO evaluation object with the specified IoU type(s)
24 coco_evaluator = CocoEvaluator(coco, iou_types)
25

```

```

26 # Use the trained model to generate predictions for the input images
27 outputs = model(images)
28
29 # Create a dictionary mapping each image ID to its corresponding prediction output
30 res = {target["image_id"].item(): output for target, output in zip(targets, outputs)}
31
32 # Update the COCO evaluation object with the predicted outputs for the current batch of images
33 coco_evaluator.update(res)
34
35
36 # Synchronize results across multiple processes for COCO evaluation
37 coco_evaluator.synchronize_between_processes()
38
39 # Accumulate results for COCO evaluation
40 coco_evaluator.accumulate()
41
42 # Summarize COCO evaluation results
43 coco_evaluator.summarize()
44
45
46 # Accessing the first statistic from the COCO evaluation object for bounding box detection
47 # The 'coco_evaluator' object is assumed to be an instance of the 'COCOeval' class from the
48 # PyCocoTools library
49 # The 'coco_eval' attribute is a dictionary of COCO evaluation results
50 # The 'bbox' key accesses the evaluation results for bounding box detection
51 # The 'stats' attribute is a list of statistics computed during the evaluation
52 # Indexing with '[0]' returns the first statistic, which is the average precision (AP) across all
      object categories
52 coco_evaluator.coco_eval['bbox'].stats[0]

```

بهترین نتیجه حاصل شده به شرح زیر است:

```

1 Accumulating evaluation results...
2 DONE (t=0.02s).
3 IoU metric: bbox
4 Average Precision (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.752
5 Average Precision (AP) @[ IoU=0.50     | area=   all | maxDets=100 ] = 1.000
6 Average Precision (AP) @[ IoU=0.75     | area=   all | maxDets=100 ] = 1.000
7 Average Precision (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.600
8 Average Precision (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.837
9 Average Precision (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.775
10 Average Recall    (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=   1 ] = 0.662
11 Average Recall    (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.758
12 Average Recall    (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.758
13 Average Recall    (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.600
14 Average Recall    (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.875

```

15 Average Recall (AR) @ [IoU=0.50:0.95 | area= large | maxDets=100] = 0.775

Program 2: One of the Test Results

در نهایت، دستوراتی را می‌نویسیم که نتیجه ارزیابی را به صورتی که در صورت سوال خواسته شده نمایش دهیم. در این نمایش کلاس‌ها، جعبه‌های مخصوصگر و شمارنده حضور دارند و جهت مشاهده یکپارچه و بهتر تصاویر پیش‌بینی و حقیقت‌بنا در کنار هم قرار داده شده‌اند. علاوه بر این، ضریب احتمال مربوط به شئ پیش‌بینی شده را هم بالای آن شئ در تصویر پیش‌بینی چاپ کردہ‌ایم که کمی حرفاً تر عمل کرده باشیم. این کد یک تابع به نام `plot_image_with_bboxes` ایجاد می‌کند که تصویر و مختصات برچسب‌های یک تصویر را به علاوه پیش‌بینی‌های مدل مربوط به همان تصویر را به عنوان ورودی دریافت می‌کند و بر روی تصویر اصلی یک تصویر جدید با برچسب‌هایی که در آن نشان داده می‌شوند رسم می‌کند. این تابع با استفاده از `ax2` و `ax1` ابعاد تصویر را به اندازه 10×5 بر حسب اینچ تعیین می‌کند و سپس تصویر را به صورت ترانسپوزیشن پیمایشی در ماتریس تصویر اصلی نشان می‌دهد. بعد از آن، دو نمودار تصویری در دو محور افقی برای نمایش حقیقت‌بنا و پیش‌بینی رسم می‌شوند. برای برچسب‌های اصلی، مختصات مربوط به جعبه‌محصورگرهایی که باید بر روی تصویر نمایش داده شوند (مختصات واقعی برای شئ) و برچسب‌های مربوطه از ورودی‌های تابع (`targets`) برداشته می‌شوند. سپس یک متغیر جدید `gt_counts` با نام `gt` ایجاد شده و تعداد اشیاء موجود در تصویر بر حسب هر کلاس شمارش می‌شود. با استفاده از تابع `zip`، برای هر شئ موجود در تصویر، مختصات مربوط به آن برای نمایش به همراه برچسب در نمودار حقیقت‌بنا رسم می‌شوند. برای نمایش جعبه‌های مخصوصگر و برچسب هر شئ، تابع به اطلاعات داده شده از مجموعه داده هدف یا پیش‌بینی استفاده می‌کند. این اطلاعات شامل جعبه مخصوصگرکنده‌ها، برچسب‌ها و امتیازهای پیش‌بینی شده در مجموعه داده هدف یا پیش‌بینی است. برای هر جعبه مخصوصگرکنده و برچسب، تابع، مستطیلی رسم می‌کند و برای هر مستطیل، نام دسته و امتیاز آن را نیز روی تصویر مشخص می‌کند. در صورتی که امتیاز پیش‌بینی کمتر از 75% باشد، آن مستطیل را نادیده می‌گیرد. تعداد پیش‌بینی‌های هر دسته شمارش می‌شود و برای هر دسته یک خلاصه‌نمایی از تعداد آن دسته به همراه نام آن ساخته می‌شود. در نهایت هم دستوری نوشته می‌شود تا تصاویر ذخیره شوند. لازم به ذکر است که دستوری به صورت تجربی نوشته شده تا هر شماره عددی به عنوان کلاسی درست تخصیص داده شود و یک رنگ مخصوص نیز به آن کلاس اختصاص داده شده است. به منظور بررسی کیفیت مدل، چندین تصویر تصادفی از داده‌های آزمون انتخاب شده و پیش‌بینی واقعی و پیش‌بینی مدل برای آن‌ها رسم شده است (شکل ۱۰ تا شکل ۲۱). همان‌طور که مشاهده می‌شود نتیجه بسیار قابل قبول است و حتی در مواردی مانند شکل ۲۰ مدل موفق به کشف اشیاء درستی شده که در حقیقت مینا برچسب‌گذاری نشده بودند. لازم به ذکر است که با آزمایش‌های متعدد سعی کرده‌ایم تمام حالات ساده و سخت را پوشش دهیم. مواردی هم بوده که تشخیص‌ها درست نبوده و اخلاقی داشته است. مانند شکل ۲۱. لازم به ذکر است که تمامی تصاویر دارای بالاترین کیفیت هستند و به زوم کردن روی آن‌ها باوضوح بسیار بالایی قابل مشاهده هستند. هم‌چنان به دلیل ذخیره‌سازی آن‌ها در قالب PDF و استفاده از \LaTeX ، تمامی تصاویر در عین حجم کم در بالاترین کیفیت و تمام متون و اعداد جزئی از فایل گزارش هستند.

```
1 # Define classes list
2 classes = [
3     "__background__", # 0 index
4     "person",
5     "bicycle",
6     "car",
7     "motorbike",
8     "aeroplane",
9     "bus",
```

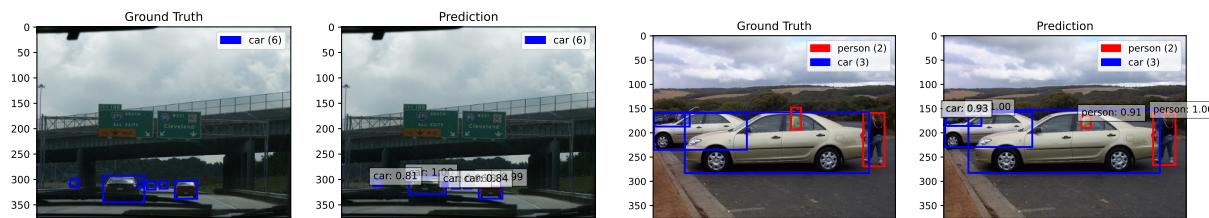
```
10     "bird",
11     "cat",
12     "chair",
13     "cow",
14     "diningtable",
15     "dog",
16     "horse",
17     "boat",
18     "bottle",
19     "pottedplant",
20     "sheep",
21     "sofa",
22     "train",
23     "tvmonitor"
24 ]
25
26 # Define bbox colors for each class
27 colors = {
28     "person": "red",
29     "bicycle": "green",
30     "car": "blue",
31     "motorbike": "yellow",
32     "aeroplane": "cyan",
33     "bus": "magenta",
34     "bird": "orange",
35     "cat": "purple",
36     "chair": "brown",
37     "cow": "pink",
38     "diningtable": "gray",
39     "dog": "olive",
40     "horse": "teal",
41     "boat": "navy",
42     "bottle": "maroon",
43     "pottedplant": "coral",
44     "sheep": "gold",
45     "sofa": "lime",
46     "train": "indigo",
47     "tvmonitor": "darkorange"
48 }
49
50 def plot_image_with_bboxes(image, targets, predictions):
51     fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5))
52     ax1.imshow(image.permute(1, 2, 0))
53     ax2.imshow(image.permute(1, 2, 0))
54     ax1.set_title('Ground Truth')
```

```
55     ax2.set_title('Prediction')
56
57     # Plot ground truth
58     gt_boxes = targets['boxes']
59     gt_labels = targets['labels']
60     gt_counts = dict()
61     for box, label in zip(gt_boxes, gt_labels):
62         x1, y1, x2, y2 = box.cpu().numpy()
63         class_name = classes[label]
64         edgecolor = colors[class_name]
65         ax1.add_patch(plt.Rectangle((x1, y1), x2 - x1, y2 - y1, fill=False, edgecolor=edgecolor,
66                                   linewidth=2))
67         if class_name not in gt_counts:
68             gt_counts[class_name] = 1
69         else:
70             gt_counts[class_name] += 1
71
72     # Add GT legend
73     gt_legend = []
74     for class_name, count in gt_counts.items():
75         gt_legend.append(mpatches.Patch(color=colors[class_name], label=f'{class_name} ({count})'))
76
77     # ax1.legend(handles=gt_legend, bbox_to_anchor=(0.5, -0.1), loc='right', ncol=len(gt_legend))
78     ax1.legend(handles=gt_legend, loc='upper right')
79
80     # Plot predictions
81     pred_boxes = predictions['boxes']
82     pred_labels = predictions['labels']
83     pred_scores = predictions['scores']
84     pred_counts = dict()
85     for box, label, score in zip(pred_boxes, pred_labels, pred_scores):
86         if score < 0.75:
87             continue
88         x1, y1, x2, y2 = box.cpu().numpy()
89         class_name = classes[label]
90         edgecolor = colors[class_name]
91         ax2.add_patch(plt.Rectangle((x1, y1), x2 - x1, y2 - y1, fill=False, edgecolor=edgecolor,
92                                   linewidth=2))
93         ax2.text(x1, y1, f'{class_name}: {score:.2f}', bbox=dict(facecolor='white', alpha=0.5))
94         if class_name not in pred_counts:
95             pred_counts[class_name] = 1
96         else:
97             pred_counts[class_name] += 1
98
99     # Add Prediction legend
100    pred_legend = []
101    for class_name, count in pred_counts.items():
```

```

97     pred_legend.append(mpatches.Patch(color=colors[class_name], label=f'{class_name} ({count
98         })'))
99
100    ax2.legend(handles=pred_legend, loc='upper right')
101
102
103
104 # Evaluate on 3 random test images
105 for i in range(1):
106     image, target = dataset_test[random.randint(0, len(dataset_test))]
107     model.eval()
108     with torch.no_grad():
109         prediction = model([image.to(device)])[0]
110         plot_image_with_bboxes(image, target, prediction)

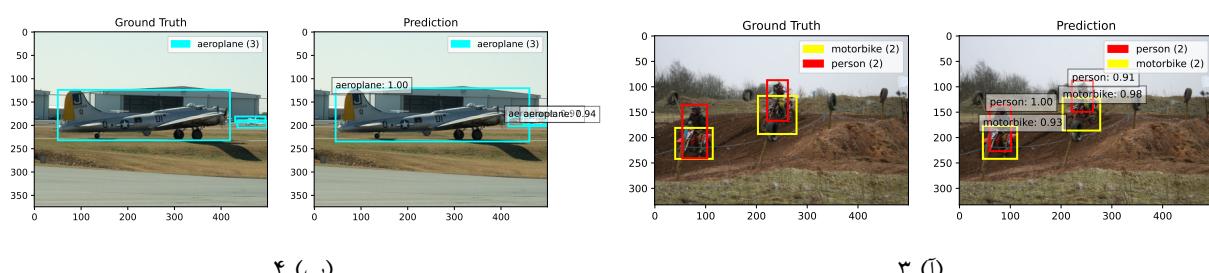
```



(ب)

(ا)

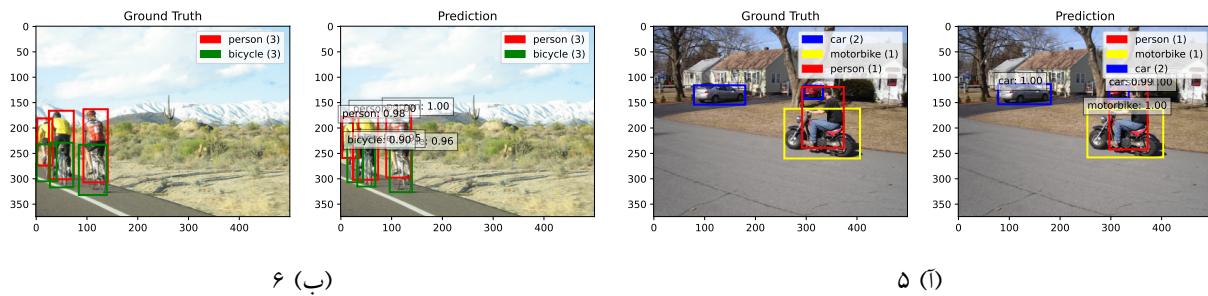
شکل ۱۰: نتیجه ارزیابی مدل روی داده‌های تصادفی از مجموعه ارزیابی.



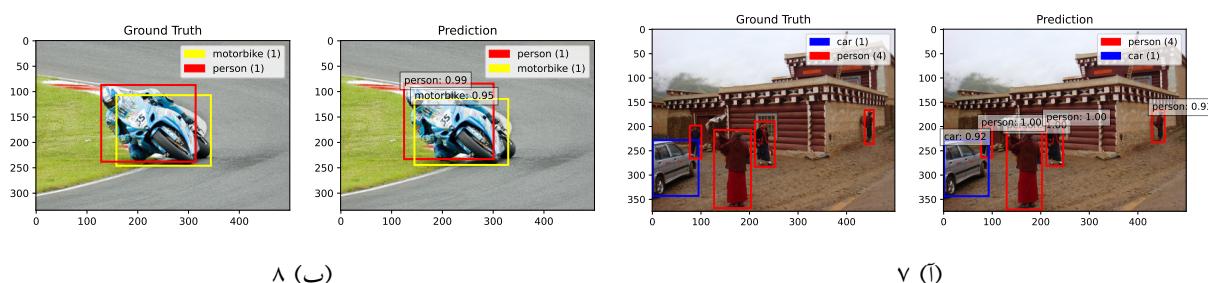
(ب)

(ا)

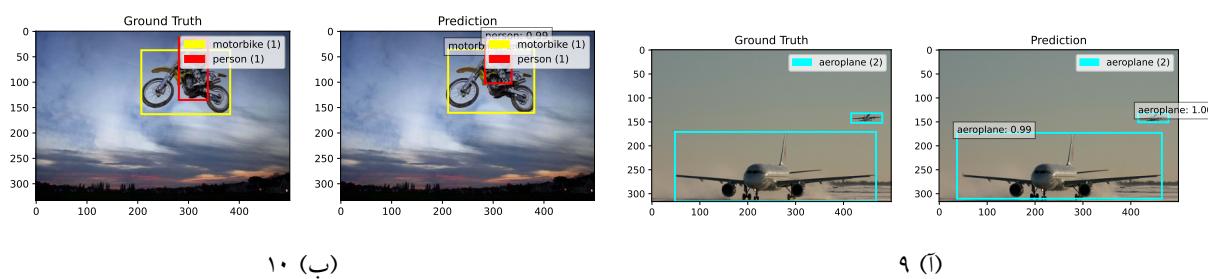
شکل ۱۱: نتیجه ارزیابی مدل روی داده‌های تصادفی از مجموعه ارزیابی.



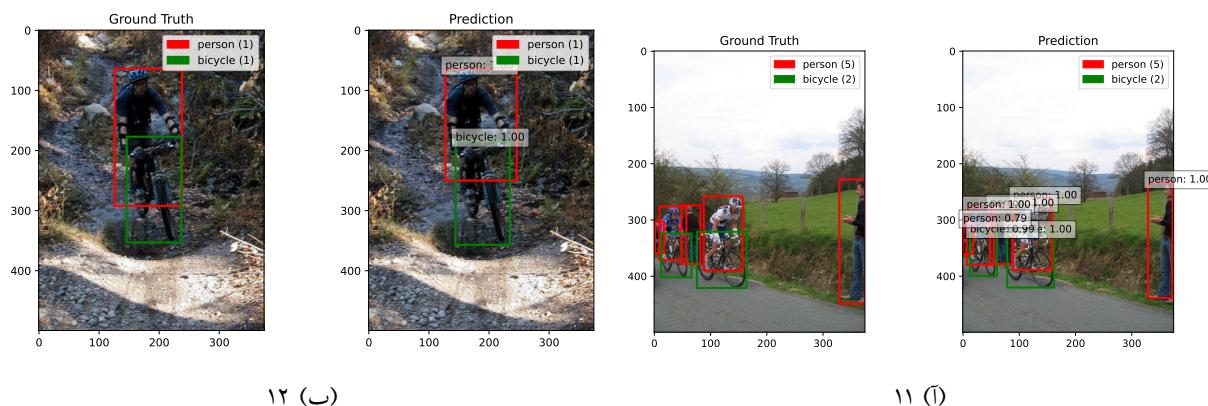
شکل ۱۲: نتیجه ارزیابی مدل روی داده‌های تصادفی از مجموعه ارزیابی.



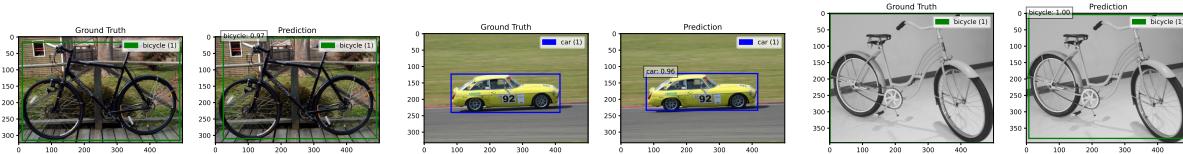
شکل ۱۳: نتیجه ارزیابی مدل روی داده‌های تصادفی از مجموعه ارزیابی.



شکل ۱۴: نتیجه ارزیابی مدل روی داده‌های تصادفی از مجموعه ارزیابی.



شکل ۱۵: نتیجه ارزیابی مدل روی داده‌های تصادفی از مجموعه ارزیابی.

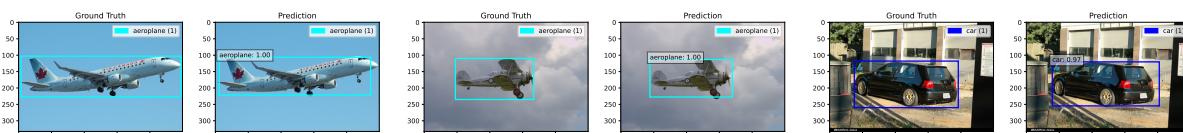


(ج) ۱۴

(ب) ۱۳

۱۲ (د)

شکل ۱۶: نتیجه ارزیابی مدل روی داده‌های تصادفی از مجموعه ارزیابی.

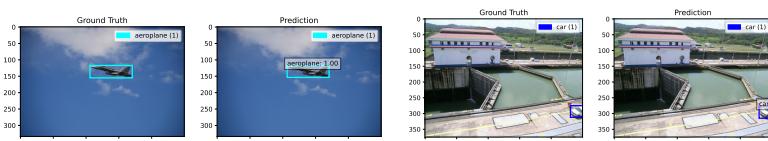


(ج) ۱۷

(ب) ۱۶

۱۵ (د)

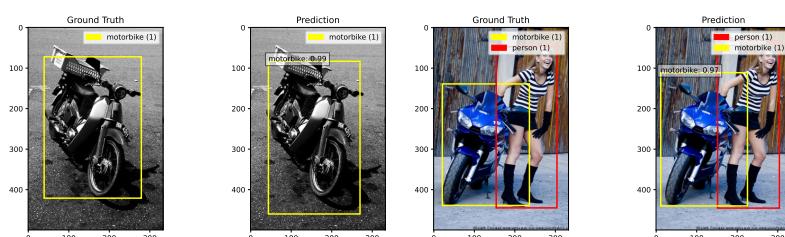
شکل ۱۷: نتیجه ارزیابی مدل روی داده‌های تصادفی از مجموعه ارزیابی.



(ب) ۱۹

۱۸ (د)

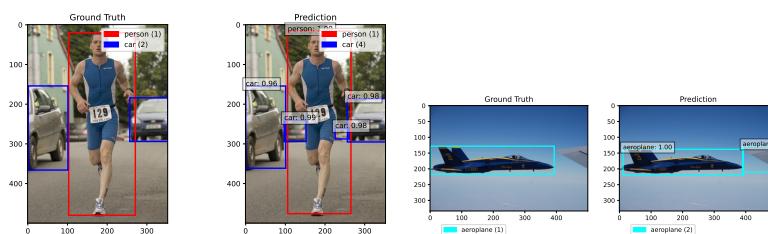
شکل ۱۸: نتیجه ارزیابی مدل روی داده‌های تصادفی از مجموعه ارزیابی.



(ب) ۲۱

۲۰ (د)

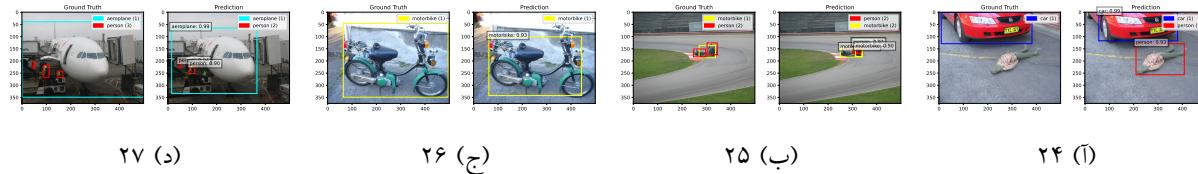
شکل ۱۹: نتیجه ارزیابی مدل روی داده‌های تصادفی از مجموعه ارزیابی.



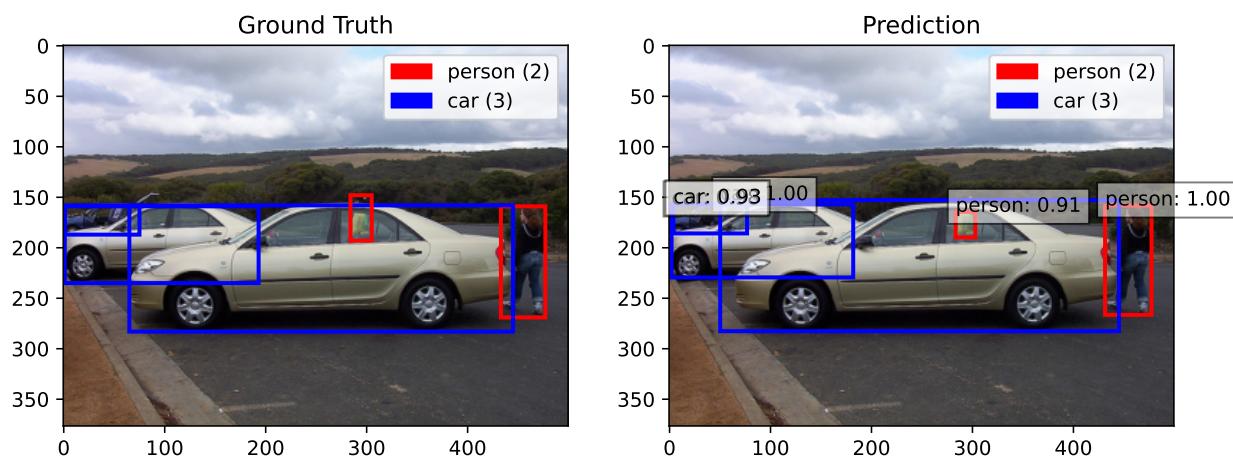
(ب) ۲۳

۲۲ (د)

شکل ۲۰: نتیجه ارزیابی مدل روی داده‌های تصادفی از مجموعه ارزیابی.



شکل ۲۱: نتیجه ارزیابی مدل روی داده های تصادفی از مجموعه ارزیابی.



شکل ۲۲: یک نمونه بزرگ از نتیجه.