دانشگاه علم و صنعت

تمرین هشتم مبانی بینایی کامپیوتر

نام و نام خانوادگی:

فرناز خوش دوست آزاد

شماره دانشجویی:

99521253

نام استاد:

دکتر محمدرضا محمدی

1)

۱. به سوالات زیر پاسخ دهید.

الف) در مورد کارایی محاسباتی استفاده از شبکه عصبی کانولوشنی (CNN) با روش پنجره لغزان در مقایسه با استفاده از CNN بدون استفاده از این روش بحث کنید. مزایا و معایب این روش چیست؟

پاسخ:

روش پنجره لغزان (Sliding Window)

در این روش، یک پنجره با اندازه ثابت روی تصویر به صورت پیوسته حرکت می‌کند و برای هر موقعیت پنجره، یک CNN جداگانه اجرا می‌شود تا ویژگی‌های آن ناحیه از تصویر استخراج شوند.

مزایا:

1. دقت بالا در مکان‌یابی: این روش می‌تواند به دقت، موقعیت اشیاء در تصویر را تعیین کند، زیرا برای هر ناحیه از تصویر یک پردازش جداگانه انجام می‌دهد.
2. انعطاف‌پذیری در اندازه اشیاء: می‌توان از پنجره‌هایی با اندازه‌های مختلف استفاده کرد تا اشیاء با اندازه‌های متفاوت را شناسایی کند.

معایب:

1. هزینه محاسباتی بالا: این روش بسیار زمان‌بر واز نظر محاسباتی سنگین است زیرا باید شبکه عصبی برای هر پنجره اجرا شود که تعداد زیادی از آن‌ها ممکن است مورد نیاز باشد.
2. تکراری بودن محاسبات: بسیاری از نواحی پنجره‌ها همپوشانی دارند و پردازش آن‌ها منجر به تکرار محاسبات می‌شود.

اعمال مستقیم CNN بدون استفاده از پنجره لغزان

در این روش، شبکه عصبی کانولوشنی به صورت مستقیم بر روی کل تصویر اعمال می‌شود و از لایه‌های کانولوشنی و پولینگ برای کاهش ابعاد و استخراج ویژگی‌های اصلی استفاده می‌شود.

مزایا:

1. کارایی محاسباتی بالا: از آنجایی که شبکه به صورت یکپارچه بر روی کل تصویر اعمال می‌شود، نیاز به اجرای مکرر شبکه برای هر پنجره نیست و بنابراین محاسبات بهینه‌تر انجام می‌شود.
2. کاهش زمان پردازش: به دلیل اعمال مستقیم و یکپارچه شبکه، زمان پردازش به‌طور قابل‌توجهی کاهش می‌یابد.

معایب:

1. دقت کمتر در مکان‌یابی اشیاء کوچک: این روش ممکن است در شناسایی اشیاء کوچک و با جزئیات دقیق کمتر مؤثر باشد.

2. انعطاف‌پذیری کمتر نسبت به تغییر اندازه اشیاء: در صورتی که اندازه اشیاء در تصویر بسیار متغیر باشد، ممکن است این روش کمتر کارآمد باشد.

لذا انتخاب بین استفاده از پنجره لغزان و اعمال مستقیم CNN به هدف مورد نظر و محدودیت‌های محاسباتی بستگی دارد. اگر دقت در مکان‌یابی و شناسایی جزئیات اشیاء کوچک مهم باشد و محدودیت‌های محاسباتی کمتر مورد توجه قرار گیرند، روش پنجره لغزان گزینه مناسبی است. اما اگر سرعت و کارایی محاسباتی در اولویت باشد، اعمال مستقیم CNN رویکرد بهتری است. در عمل، ترکیب این دو روش با استفاده از تکنیک‌های جدید مثل R-CNN و YOLO سعی در بهره‌گیری از مزایای هر دو روش دارد.

ب) توضیح دهید که YOLO چگونه چندین کلاس اشیاء را در یک سلول شبکه واحد مدیریت می‌کند و چگونه احتمال کلاس‌ها را به همراه مختصات جعبه‌های مرزی پیش‌بینی می‌کند؟ چگونه جعبه‌های Anchor به تشخیص اشیاء با اشکال و اندازه‌های مختلف کمک می‌کنند؟

پاسخ:

روش YOLO = You Only Look Once یکی از تکنیک‌های پیشرفته برای شناسایی اشیاء در تصاویر است که به دلیل سرعت و دقت بالا مورد توجه قرار گرفته است. در ادامه به بررسی نحوه مدیریت چندین کلاس اشیاء در یک سلول شبکه، پیش‌بینی احتمالات کلاس‌ها به همراه مختصات جعبه‌های مرزی، و نقش جعبه‌های Anchor در تشخیص اشیاء با اشکال و اندازه‌های مختلف می‌پردازیم.

مدیریت چندین کلاس اشیاء در یک سلول شبکه

در YOLO، تصویر به یک شبکه S x S تقسیم می‌شود و هر سلول شبکه مسئول پیش‌بینی جعبه‌های مرزی (Bounding Boxes) و کلاس‌های اشیاء است. هر سلول شبکه چندین جعبه مرزی و یک مجموعه احتمالات کلاس‌ها را پیش‌بینی می‌کند.

جزئیات پیش‌بینی:

1. جعبه‌های مرزی: هر سلول تعداد ثابتی از جعبه‌های مرزی (معمولاً دو یا بیشتر) را پیش‌بینی می‌کند. هر جعبه مرزی شامل مختصات مرکز(x, y) ، عرض(w) و ارتفاع(h) و همچنین یک احتمال اطمینان (confidence score) است که نشان می‌دهد آیا جعبه حاوی یک شیء است یا خیر و چقدر دقیق است.

2. احتمالات کلاس‌ها: هر سلول همچنین احتمال هر کلاس شیء را پیش‌بینی می‌کند. این احتمالات برای همه جعبه‌های مرزی آن سلول مشترک هستند.

پیش‌بینی احتمال کلاس‌ها و مختصات جعبه‌های مرزی

هر سلول شبکه یک وکتور شامل اطلاعات زیر را تولید می‌کند:

- (B) جعبه مرزی: هر جعبه شامل مختصات (x, y, w, h) و احتمال اطمینان (confidence score) است.

- احتمالات کلاس‌ها: احتمال هر کلاس برای هر سلول.

فرمول نهایی برای هر جعبه مرزی به شکل زیر است:

$$\mathrm{IOU}(\mathrm{pred}, \mathrm{truth}) \times P(\mathrm{object}) \times P(\mathrm{class}_i|\mathrm{object}$$

- P(class|object (i): احتمال کلاس (i) مشروط بر اینکه شیء در جعبه وجود داشته باشد.

- P(object (: احتمال وجود شیء در جعبه مرزی.

- IOU(pred,truth): میزان همپوشانی جعبه پیش‌بینی شده با جعبه واقعی (Intersection Over Union).

جعبه‌های Anchor و تشخیص اشیاء با اشکال و اندازه‌های مختلف

جعبه‌های Anchor یک تکنیک مهم در YOLOv2 و نسخه‌های بعدی است که به بهبود دقت و انعطاف‌پذیری در شناسایی اشیاء با اشکال و اندازه‌های مختلف کمک می‌کند.

نقش جعبه‌های Anchor:

- تنوع در ابعاد جعبه‌ها: جعبه‌های Anchor مجموعه‌ای از جعبه‌های مرزی با ابعاد و نسبت‌های مختلف هستند که به هر سلول شبکه اختصاص داده می‌شوند. این جعبه‌ها به شبکه کمک می‌کنند تا اشیاء با اندازه‌ها و نسبت‌های مختلف را بهتر شناسایی کند.

- کاهش پیچیدگی محاسباتی: با استفاده از جعبه‌های Anchor، شبکه به جای پیش‌بینی مستقیم ابعاد جعبه‌ها، تنها نیاز دارد که تفاوت‌ها (offsets) نسبت به جعبه‌های Anchor را پیش‌بینی کند که این کار محاسبات را ساده‌تر و دقیق‌تر می‌کند.

- بهبود دقت: استفاده از جعبه‌های Anchor باعث می‌شود که شبکه بتواند به‌طور موثرتر و با دقت بیشتری اشیاء مختلف را در ابعاد گوناگون شناسایی کند، زیرا جعبه‌های Anchor به طور خاص برای پوشش دادن طیف وسیعی از اشکال و اندازه‌های ممکن طراحی شده‌اند.

لذا YOLO با استفاده از شبکه‌های عصبی کانولوشنی و تکنیک‌های هوشمندانه‌ای مانند جعبه‌های Anchor و تقسیم تصویر به سلول‌های شبکه، به طور کارآمدی اشیاء را شناسایی می‌کند. این روش‌ها به YOLO امکان می‌دهند تا همزمان چندین کلاس شیء را در یک سلول شبکه مدیریت کرده و مختصات جعبه‌های مرزی را با دقت بالا پیش‌بینی کند. استفاده از جعبه‌های Anchor به‌طور خاص باعث بهبود دقت و کارایی شبکه در شناسایی اشیاء با اندازه‌ها و اشکال مختلف می‌شود.

2)

۲. به سوالات زیر پاسخ دهید.

الف) الگوریتم‌های YOLO و SSD (Single Shot MultiBox Detector) را از نظر معماری، سرعت و دقت مقایسه کنید. تحلیل دقیقی از سناریوهایی که یکی ممکن است از دیگری بهتر عمل کند ارائه دهید.

پاسخ:

مقایسه الگوریتم‌های YOLO و SSD (Single Shot MultiBox Detector) از نظر معماری، سرعت، و دقت و تحلیل سناریوهایی که یکی ممکن است از دیگری بهتر عمل کند، اهمیت زیادی در انتخاب مناسب‌ترین روش برای شناسایی اشیاء در کاربردهای مختلف دارد. در ادامه به مقایسه این دو الگوریتم می‌پردازیم.

مقایسه معماری YOLO و SSD

معماری YOLO = You Only Look Once:

1. معماری شبکه: YOLO از یک شبکه عصبی کانولوشنی استفاده می‌کند که تصویر ورودی را به یک شبکه S x S تقسیم می‌کند و برای هر سلول شبکه تعداد ثابتی جعبه مرزی و احتمالات کلاس‌ها را پیش‌بینی می‌کند.

2. خروجی شبکه: هر سلول شبکه یک vector خروجی شامل مختصات جعبه‌های مرزی، احتمال اطمینان (confidence score)، و احتمال کلاس‌ها را تولید می‌کند.

3. یکپارچگی پردازش: YOLO یک تصویر را به صورت کامل در یک مرحله پردازش می‌کند و از این رو نام "You Only Look Once" را گرفته است. این امر به افزایش سرعت آن کمک می‌کند.

معماری SSD = Single Shot MultiBox Detector:

1. معماری شبکه: SSD از یک شبکه عصبی کانولوشنی پایه استفاده می‌کند و در مراحل مختلف از لایه‌های کانولوشنی با مقیاس‌های مختلف برای پیش‌بینی جعبه‌های مرزی و کلاس‌ها بهره می‌برد. این لایه‌ها شامل لایه‌های اضافی برای بهبود دقت در سطوح مختلف هستند.

2. خروجی شبکه: SSD از چندین لایه با ابعاد و رزولوشن‌های مختلف برای پیش‌بینی جعبه‌های مرزی استفاده می‌کند. هر لایه تعداد مشخصی از جعبه‌های مرزی با اندازه‌ها و نسبت‌های مختلف را پیش‌بینی می‌کند.

3. چندین پیش‌بینی در یک مرحله: SSD به‌طور همزمان پیش‌بینی‌های متعددی را از چندین لایه با مقیاس‌های مختلف انجام می‌دهد، که باعث می‌شود به عنوان "Single Shot" شناخته شود.

مقایسه سرعت YOLO و SSD

1. سرعت YOLO: YOLO به دلیل پردازش یکپارچه تصویر و پیش‌بینی جعبه‌های مرزی در یک مرحله، بسیار سریع است. YOLOv3 قادر است تا 45 فریم در ثانیه (fps) را پردازش کند که برای کاربردهای بلادرنگ مناسب است.

2. سرعت SSD: SSD نیز به دلیل طراحی چند لایه‌ای و استفاده از پیش‌بینی‌های چندگانه در یک مرحله، سرعت بالایی دارد. نسخه‌های مختلف SSD مانند SSD300 و SSD512 سرعت‌های متفاوتی دارند ولی معمولاً نزدیک به سرعت YOLO هستند.

مقایسه دقت YOLO و SSD

1. دقت YOLO: YOLO به دلیل پردازش یکپارچه تصویر و استفاده از شبکه عصبی پیچیده، دقت بالایی در تشخیص اشیاء دارد. اما ممکن است در شناسایی اشیاء کوچک یا اشیائی که در مجاورت یکدیگر قرار دارند، کمتر دقیق عمل کند.

2. دقت SSD:SSD به دلیل استفاده از لایه‌های متعدد با رزولوشن‌های مختلف، دقت بالاتری در شناسایی اشیاء کوچک و اشیائی که در نزدیکی یکدیگر قرار دارند دارد. SSD می‌تواند اشیاء با ابعاد مختلف را با دقت بالاتری شناسایی کند.

تحلیل سناریوهای کاربردی

سناریوهایی که YOLO بهتر عمل می‌کند:

1. کاربردهای بلادرنگ: به دلیل سرعت بالا، YOLO برای کاربردهای بلادرنگ مثل دوربین‌های نظارتی، خودروهای خودران، و رباتیک مناسب است.

2. تصاویر با تراکم کم اشیاء: در تصاویری که اشیاء کم و با فاصله از یکدیگر هستند، YOLO عملکرد بهتری دارد.

سناریوهایی که SSD بهتر عمل می‌کند:

1. شناسایی اشیاء کوچک: در تصاویری که اشیاء کوچک زیادی وجود دارند، SSD به دلیل دقت بالاتر در شناسایی این اشیاء، عملکرد بهتری دارد.

2. تصاویر با تراکم بالای اشیاء: در تصاویری که اشیاء زیادی در نزدیکی یکدیگر قرار دارند، SSD به دلیل استفاده از لایه‌های چندگانه، بهتر عمل می‌کند.

لذا هر دو الگوریتم YOLO و SSD مزایا و معایب خاص خود را دارند و انتخاب بین آن‌ها بستگی به نیازهای خاص کاربرد دارد. YOLO به دلیل سرعت بالا برای کاربردهای بلادرنگ مناسب است، در حالی که SSD به دلیل دقت بالا در شناسایی اشیاء کوچک و متراکم، در سناریوهایی که نیاز به دقت بیشتری دارند، برتری دارد.

ب) مفهوم Focal Loss را که در RetinaNet استفاده می‌شود توضیح دهید. این مفهوم چگونه به مشکل عدم تعادل کلاس‌ها در تشخیص اشیاء پرداخته است؟

پاسخ:

Focal Loss یک تابع زیان (Loss Function) است که در مدل RetinaNet معرفی شده است تا مشکل عدم تعادل کلاس‌ها در تشخیص اشیاء را برطرف کند. در ادامه به توضیح این مفهوم و نحوه عملکرد آن می‌پردازیم.

مفهوم Focal Loss

مشکل عدم تعادل کلاس‌ها

در بسیاری از مسائل تشخیص اشیاء، تعداد نمونه‌های پس‌زمینه (کلاس‌های منفی) بسیار بیشتر از تعداد نمونه‌های اشیاء (کلاس‌های مثبت) است. این مشکل منجر به عدم تعادل کلاس‌ها می‌شود، یعنی مدل بیشتر روی یادگیری ویژگی‌های پس‌زمینه تمرکز می‌کند و کمتر قادر به شناسایی صحیح اشیاء است.

Cross-Entropy Loss

تابع زیان معمول در تشخیص اشیاء، Cross-Entropy Loss است که به شکل زیر تعریف می‌شود:

$$\mathrm{CE}(p_t) = -\log(p_t)$$

که در آن \(p_t \) احتمال پیش‌بینی شده برای کلاس صحیح است.

تعریف Focal Loss

Focal Loss با تغییر Cross-Entropy Loss، تأکید بیشتری روی نمونه‌های دشوار (سخت) و تأکید کمتری روی نمونه‌های آسان (ساده) می‌گذارد. این کار با افزودن یک عامل تعدیل‌گر (modulating factor) به Cross-Entropy Loss انجام می‌شود:

$$\mathrm{FL}(p_t) = -(1 - p_t)^{\gamma} \log(p_t)$$

در اینجا:

- (gamma) یک پارامتر قابل تنظیم است که شدت تأکید بر نمونه‌های سخت را تعیین می‌کند.
- (p_t) احتمال پیش‌بینی شده برای کلاس صحیح است.

نحوه عملکرد Focal Loss

- عامل تعدیل‌گر: عبارت (1 - p_t)gamma) باعث کاهش تأثیر نمونه‌های آسان (نمونه‌هایی که مدل به درستی آن‌ها را شناسایی کرده و (p_t) به 1 نزدیک است و افزایش تأثیر نمونه‌های سخت (نمونه‌هایی که مدل به درستی آن‌ها را شناسایی نکرده و (p_t) به 0 نزدیک است) می‌شود.
- کاهش عدم تعادل: با افزایش پارامتر(gamma)، تأثیر نمونه‌های آسان به شدت کاهش می‌یابد و مدل به یادگیری نمونه‌های دشوارتر (کمتر دیده شده) بیشتر تمرکز می‌کند. این امر به بهبود تعادل بین کلاس‌ها و افزایش دقت در شناسایی اشیاء کمک می‌کند.

کاربرد Focal Loss در RetinaNet

RetinaNet یک مدل تشخیص اشیاء است که از Focal Loss برای بهبود دقت تشخیص در شرایط عدم تعادل کلاس‌ها استفاده می‌کند. مزایای استفاده از Focal Loss در RetinaNet عبارتند از:

- بهبود تشخیص اشیاء کوچک: با تأکید بیشتر بر نمونه‌های سخت و کمتر دیده شده، مدل بهبود قابل توجهی در تشخیص اشیاء کوچک و کم‌تکرار پیدا می‌کند.
- کاهش خطای نادرست شناسایی: با کاهش تأثیر نمونه‌های آسان و پس‌زمینه، خطای نادرست شناسایی (False Positives) کاهش می‌یابد.
- یادگیری مؤثرتر: مدل به جای تمرکز بر یادگیری ویژگی‌های پس‌زمینه، بیشتر بر یادگیری ویژگی‌های مرتبط با اشیاء تمرکز می‌کند.

لذا Focal Loss یک راهکار موثر برای مقابله با مشکل عدم تعادل کلاس‌ها در مسائل تشخیص اشیاء است. این تابع زیان با تغییر وزن نمونه‌های آسان و سخت، به مدل کمک می‌کند تا به طور موثرتری یاد بگیرد و دقت خود را در شناسایی اشیاء افزایش دهد. استفاده از Focal Loss در RetinaNet بهبود قابل توجهی در عملکرد این مدل در شرایط عدم تعادل کلاس‌ها ایجاد کرده است.

3)

۳. دلیل ضرورت تکنیک حذف مقادیر غیر بیشینه (NMS) را در تشخیص اشیاء توضیح دهید و بگویید چگونه عملکرد مدل‌های تشخیص اشیاء را بهبود می‌بخشد. سپس با توجه به آرایه‌ای از جعبه‌های مرزی تشخیص داده شده با امتیازات اطمینان مربوطه، مراحل مربوط به اعمال NMS را توضیح دهید. تاثیر تغییر آستانه IoU در NMS چیست؟

پاسخ:

ضرورت تکنیک حذف مقادیر غیر بیشینه (Non-Maximum Suppression - NMS)

در مدل‌های تشخیص اشیاء، الگوریتم‌ها معمولاً چندین جعبه مرزی (Bounding Box) را برای یک شیء واحد تولید می‌کنند. این جعبه‌های مرزی معمولاً همپوشانی زیادی دارند و بسیاری از آن‌ها نادرست یا تکراری هستند. تکنیک حذف مقادیر غیر بیشینه (NMS) برای حذف جعبه‌های تکراری و نگه‌داشتن تنها بهترین جعبه برای هر شیء استفاده می‌شود.

چگونه NMS عملکرد مدل‌های تشخیص اشیاء را بهبود می‌بخشد؟

- کاهش تکرارها: NMS جعبه‌های مرزی تکراری را حذف می‌کند و تنها جعبه‌هایی که بیشترین احتمال اطمینان (confidence score) را دارند، نگه می‌دارد.

- افزایش دقت: با حذف جعبه‌های نادرست و همپوشانی، NMS دقت مدل را در تشخیص اشیاء افزایش می‌دهد و از تشخیص‌های نادرست جلوگیری می‌کند.
- کارایی بالاتر: نتایج نهایی با تعداد کمتری جعبه مرزی ارائه می‌شود که پردازش‌های بعدی را ساده‌تر و سریع‌تر می‌کند.

مراحل اعمال NMS

فرض کنید یک آرایه از جعبه‌های مرزی (B) و امتیازات اطمینان (S) داریم. مراحل اعمال NMS به شرح زیر است:

1. مرتب‌سازی جعبه‌ها بر اساس امتیازات اطمینان: جعبه‌ها را بر اساس امتیازات اطمینان به ترتیب نزولی مرتب کنید.
2. انتخاب جعبه با بالاترین امتیاز اطمینان: جعبه با بالاترین امتیاز اطمینان را به عنوان جعبه مرجع انتخاب کنید.
3. محاسبه همپوشانی (IoU): برای سایر جعبه‌ها، میزان همپوشانی (Intersection over Union - IoU) را با جعبه مرجع محاسبه کنید.
4. حذف جعبه‌های با همپوشانی بالا: جعبه‌هایی که مقدار IoU آن‌ها با جعبه مرجع از آستانه تعیین شده بیشتر است را حذف کنید.
5. تکرار مراحل 2 تا 4: مراحل 2 تا 4 را تکرار کنید تا زمانی که جعبه‌های باقیمانده کمتر از آستانه IoU باشند.

تاثیر تغییر آستانه IoU در NMS

آستانه IoU یک پارامتر حیاتی در NMS است که تعیین می‌کند چه میزان همپوشانی بین جعبه‌ها مجاز است. تغییر این آستانه تاثیر زیادی بر نتایج نهایی دارد:

- آستانه پایین (IoU کم): اگر آستانه IoU پایین باشد (مثلاً 0.3)، تعداد بیشتری از جعبه‌ها حذف می‌شوند. این می‌تواند منجر به حذف جعبه‌های صحیح نیز شود و در نتیجه، نرخ تشخیص نادرست (False Negative Rate) افزایش یابد.
- آستانه بالا (IoU زیاد): اگر آستانه IoU بالا باشد (مثلاً 0.7)، تعداد کمتری از جعبه‌ها حذف می‌شوند. این می‌تواند منجر به باقی‌ماندن جعبه‌های تکراری و نادرست شود و در نتیجه، نرخ تشخیص نادرست (False Positive Rate) افزایش یابد.

لذا تکنیک NMS برای حذف جعبه‌های مرزی تکراری و نادرست در مدل‌های تشخیص اشیاء ضروری است. این تکنیک با کاهش تکرارها و افزایش دقت، عملکرد مدل‌های تشخیص اشیاء را بهبود می‌بخشد. آستانه IoU تعیین‌کننده میزان حساسیت NMS است و باید به‌طور مناسب تنظیم شود تا تعادل بین حذف جعبه‌های نادرست و نگه‌داشتن جعبه‌های صحیح برقرار شود.

4)

۴. با توجه به جعبه‌های مرزی پیش‌بینی شده با امتیازات اطمینان زیر:

{(0.9, (50, 50, 100, 100)), (0.8, (55, 60, 105, 110)), (0.7, (100, 100, 150, 150)), (0.6, (45, 50, 95, 100))}

حذف مقادیر غیر بیشینه (NMS) را با آستانه IoU به مقدار 0.5 اعمال کنید و جعبه‌های مرزی باقی‌مانده را لیست کنید.

پاسخ:

برای اعمال تکنیک حذف مقادیر غیر بیشینه (Non-Maximum Suppression - NMS) با آستانه IoU به مقدار 0.5 بر روی جعبه‌های مرزی داده شده، مراحل زیر را دنبال می‌کنیم:

جعبه‌های مرزی پیش‌بینی شده:

- (0.9, (50, 50, 100, 100))
- (0.8, (55, 60, 105, 110))
- (0.7, (100, 100, 150, 150))
- (0.6, (45, 50, 95, 100))

مراحل اعمال NMS

مرحله 1: مرتب‌سازی جعبه‌ها بر اساس امتیازات اطمینان

جعبه‌ها را بر اساس امتیازات اطمینان به ترتیب نزولی مرتب می‌کنیم:

1. (0.9, (50, 50, 100, 100))
2. (0.8, (55, 60, 105, 110))
3. (0.7, (100, 100, 150, 150))
4. (0.6, (45, 50, 95, 100))

مرحله 2: انتخاب جعبه با بالاترین امتیاز اطمینان

- جعبه مرجع: (0.9, (50, 50, 100, 100))

مرحله 3: محاسبه IoU برای سایر جعبه‌ها

- IoU بین (50, 50, 100, 100) و (55, 60, 105, 110)
- IoU بین (50, 50, 100, 100) و (100, 100, 150, 150)
- IoU بین (50, 50, 100, 100) و (45, 50, 95, 100)

محاسبات IoU:

1. IoU بین (50, 50, 100, 100) و (55, 60, 105, 110):

- تقاطع:

$$x_1 = 50, y_1 = 50, x_2 = 95, y_2 = 100$$

- مساحت تقاطع: 50x45 = 2250
- مساحت (50, 50, 100, 100): 50x50 = 2500
- مساحت (55, 60, 105, 110): 50x50 = 2500

$$\text{IoU: } \frac{1800}{2500+2500-1800} = \frac{1800}{3200} \approx 0.56$$

- حذف (زیرا IoU < 0.5)

2. IoU بین (50, 50, 100, 100) و (100, 100, 150, 150):

- تقاطع: (x_1 = 100, y_1 = 100, x_2 = 100, y_2 = 100 \) (هیچ تقاطعی ندارند)
- IoU: 0 (زیرا هیچ تقاطعی وجود ندارد)
- نگه‌داری

3. IoU بین (50, 50, 100, 100) و (45, 50, 95, 100):

- تقاطع: (x_1 = 50, y_1 = 50, x_2 = 95, y_2 = 100)
- مساحت تقاطع: 50x45 = 2250

- مساحت (50, 50, 100 ,100): 50x50 = 2500

- مساحت (45, 50, 95 ,100): 50x50 = 2500

- IoU: $\frac{2250}{2500+2500-2250} = \frac{2250}{2750} \approx 0.818$

- حذف (زیرا IoU > 0.5)

مرحله 4: تکرار مراحل 2 تا 4 برای جعبه‌های باقیمانده

جعبه‌های باقیمانده بعد از اولین مرحله حذف:

- (0.7, (100, 100, 150, 150))

این جعبه تنها جعبه باقیمانده است و نیازی به محاسبات بیشتر نیست.

جعبه‌های مرزی باقیمانده

بعد از اعمال NMS با آستانه IoU 0.5، جعبه‌های مرزی باقیمانده به شرح زیر هستند:

1. (0.9, (50, 50, 100, 100))

2. (0.7, (100, 100, 150, 150))

لذا با اعمال تکنیک NMS، جعبه‌های مرزی تکراری و نادرست حذف شدند و تنها دو جعبه مرزی باقیماند که احتمال بیشتری دارند شیءهای واقعی را نشان دهند.

5)

۵. اگر یک جعبه مرزی تشخیص داده شده دارای ویژگی‌های زیر باشد:

pc=0.85, bx=0.5, by=0.6, bw=0.3, bh=0.4

در یک مدل YOLO، این مختصات نرمال شده را به مقادیر پیکسلی واقعی تبدیل کنید با فرض اینکه اندازه تصویر ورودی ۴۱۶×۴۱۶ باشد.

پاسخ:

برای تبدیل مختصات نرمال شده جعبه مرزی به مقادیر پیکسلی واقعی در یک مدل YOLO، باید مقادیر نرمال شده را به ابعاد واقعی تصویر ضرب کنیم.

مشخصات تصویر و جعبه مرزی

- اندازه تصویر ورودی: 416x416 پیکسل

- مقادیر نرمال شده جعبه مرزی:

- (p_c = 0.85) (احتمال کلاس)

- (b_x = 0.5) (مختصات مرکز جعبه مرزی در محور x)

- (b_y = 0.6) (مختصات مرکز جعبه مرزی در محور y)

- (b_w = 0.3) (عرض جعبه مرزی نسبت به عرض تصویر)

- (b_h = 0.4) (ارتفاع جعبه مرزی نسبت به ارتفاع تصویر)

تبدیل مقادیر نرمال شده به مقادیر پیکسلی

برای تبدیل مقادیر نرمال شده به مقادیر پیکسلی واقعی، از فرمول‌های زیر استفاده می‌کنیم:

1. محاسبه مرکز جعبه مرزی (x, y) در مقیاس پیکسلی:

$$\text{عرض جعبه مرزی}_{\text{pixel}} = b_w \times \text{عرض تصویر}$$

$$\text{ارتفاع جعبه مرزی}_{\text{pixel}} = b_h \times \text{ارتفاع تصویر}$$

2. محاسبه عرض و ارتفاع جعبه مرزی در مقیاس پیکسلی:

$$\text{عرض جعبه مرزی}_{\text{pixel}} = b_w \times \text{عرض تصویر}$$

$$\text{ارتفاع جعبه مرزی}_{\text{pixel}} = b_h \times \text{ارتفاع تصویر}$$

<u>محاسبات</u>

1. محاسبه مرکز جعبه مرزی (x, y) در مقیاس پیکسلی:

$$x_{\text{pixel}} = 0.5 \times 416 = 208 \text{ پیکسل}$$

$$y_{\text{pixel}} = 0.6 \times 416 = 249.6 \text{ پیکسل}$$

2. محاسبه عرض و ارتفاع جعبه مرزی در مقیاس پیکسلی:

$$\text{عرض جعبه مرزی}_{\text{pixel}} = 0.3 \times 416 = 124.8 \text{ پیکسل}$$

$$\text{ارتفاع جعبه مرزی}_{\text{pixel}} = 0.4 \times 416 = 166.4 \text{ پیکسل}$$

مختصات نرمال شده جعبه مرزی در مقیاس پیکسلی واقعی به شرح زیر است:

- مرکز جعبه مرزی: (208, 249.6) پیکسل
- عرض جعبه مرزی: (124.8) پیکسل
- ارتفاع جعبه مرزی: (166.4) پیکسل

در نتیجه، جعبه مرزی در تصویر اصلی با اندازه 416x416 پیکسل، مختصات و ابعاد پیکسلی زیر را خواهد داشت:

- مرکز جعبه مرزی: 208, 249.6 پیکسل
- عرض جعبه مرزی: 124.8 پیکسل
- ارتفاع جعبه مرزی: 166.4 پیکسل

6)

۶. ابتدا یک شی را به دلخواه انتخاب نموده و تعدادی تصویر از آن جمع‌آوری نمایید. در مرحله بعد، با استفاده از ابزار LabelMe، شی درون تصاویر جمع‌آوری شده را برچسب‌گذاری کنید.

سپس مدل yolo را بر روی داده جمع‌آوری شده آموزش دهید. حال برای بررسی دقت مدل و کارکرد آن با استفاده از وب‌کم (به صورت live)، مدل را بررسی کنید و تعدادی از شی انتخابی را در مقابل دوربین قرار دهید و بررسی کنید که تشخیص داده می‌شود یا خیر.

پاسخ:

برای حل این سوال در ابتدا چند عکس از حالت های مختلف گرفتم که نمونه های آن در زیپ مربوطه قرار داده شده است و 6 کلاس داریم که حالت های and و not و victory و four , hi و ok را نشان می دهند، سپس با استفاده از نرم افزار labelme که با pip نصب کردم هر کدام از حالت ها را label گذاری کردم و فایل های json آن را در پوشه ی مربوطه گذاشتم و سپس به سراغ نوتبوک رفتم و کتابخانه های مورد نیاز خود را install کردم و پوشه های مورد نیاز را نیز ساختم که در زیر موجود هستند.

```
!git clone https://github.com/ultralytics/yolov5  # Clone YOLOv5 repository
%cd yolov5
!pip install -r requirements.txt  # Install dependencies
```

```
!pip install -q ultralytics

```

```
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 793.7/793.7 kB 15.1 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.3/21.3 MB 58.1 MB/s eta 0:00:00
```

```
!mkdir -p /content/dataset/images
!mkdir -p /content/dataset/labels
!cp /content/your_images/*.jpg /content/dataset/images/
!cp /content/yolo_annotations/*.txt /content/dataset/labels/
```

```
%cd /content/yolov5
!pip install comet_ml
!python /content/yolov5/train.py --img 416 --batch 16 --epochs 100 --data /conter

```

پس از این قسمت برای قابل خوانده شدن فایل های json باید آنها قابل خواندن کنیم. برای این کار تابعی نوشتم که برچسب‌های LabelMe را به فرمت YOLO تبدیل می‌کند. سه آرگومان دارد:

- `json_dir`: دایرکتوری که فایل‌های JSON مربوط به LabelMe در آن قرار دارد.
- `output_dir`: دایرکتوری برای ذخیره فایل‌های برچسب‌گذاری به فرمت YOLO.
- `img_dir`: دایرکتوری که تصاویر مربوط به برچسب‌ها در آن قرار دارد.

در این تابع در ابتدا بررسی می‌کند که آیا دایرکتوری خروجی وجود دارد؛ در غیر این صورت، آن را ایجاد می‌کند و سپس روی همه فایل‌های موجود در دایرکتوری JSON پیمایش می‌کند و فقط فایل‌هایی که با `.json` تمام می‌شوند را پردازش می‌کند و فایل JSON را باز کرده و داده‌های آن را بارگذاری می‌کند.

```python
def convert_labelme_to_yolo(json_dir, output_dir, img_dir):
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    for json_file in os.listdir(json_dir):
        if json_file.endswith('.json'):
            with open(os.path.join(json_dir, json_file)) as f:
                data = json.load(f)

            image_path = os.path.join(img_dir, data['imagePath'])
            image = cv2.imread(image_path)
            height, width, _ = image.shape

            yolo_annotations = []
```

در مرحله ی بعدی تصویر مربوطه را با استفاده از OpenCV می‌خواند و ابعاد آن (ارتفاع و عرض) را می‌گیرد و یک لیست برای ذخیره برچسب‌های YOLO مقداردهی اولیه می‌کند. برای هر شکل در داده‌های JSON، برچسب و نقاط را استخراج می‌کند. سپس مختصات جعبه مرزی (حداقل و حداکثر x و y) را محاسبه می‌کند.

```python
            yolo_annotations = []
            for shape in data['shapes']:
                label = shape['label']
                points = shape['points']
                x_min = min([p[0] for p in points])
                y_min = min([p[1] for p in points])
                x_max = max([p[0] for p in points])
                y_max = max([p[1] for p in points])

                x_center = (x_min + x_max) / 2 / width
                y_center = (y_min + y_max) / 2 / height
                bbox_width = (x_max - x_min) / width
                bbox_height = (y_max - y_min) / height
                label_id = label_mapping[label]
                yolo_annotations.append(f"{label_id} {x_center} {y_center} {bbox_width} {bbox_height}")
            txt_file = os.path.join(output_dir, os.path.splitext(json_file)[0] + '.txt')
            with open(txt_file, 'w') as f:
                f.write('\n'.join(yolo_annotations))
```

مختصات مرکز، عرض و ارتفاع جعبه مرزی را در فرمت YOLO (نرمال‌شده به اندازه تصویر) محاسبه می‌کند. سپس برچسب را با استفاده از دیکشنری `label_mapping` به یک شناسه عددی نگاشت کرده و برچسب فرمت شده را به لیست اضافه می‌کند.

یک فایل متنی با همان نام فایل JSON (اما با پسوند `.txt`) در دایرکتوری خروجی ایجاد کرده و برچسب‌های YOLO را در آن می‌نویسد.

```
label_mapping = {
    'love': 0,
    'not': 1,
    'ok': 2,
    'victory': 3,
    'hi': 4,
    'four': 5
}

json_dir = '/content/your_labelme_jsons'
output_dir = '/content/yolo_annotations'
img_dir = '/content/your_images'

convert_labelme_to_yolo(json_dir, output_dir, img_dir)
```

همانطور که در بالا دیده می شود یک دیکشنری که هر برچسب را به یک شناسه عددی نگاشت می‌کند که در برچسب‌های YOLO استفاده می‌شود. دایرکتوری‌های فایل‌های JSON، برچسب‌های خروجی و تصاویر را مشخص کرده و سپس تابع `convert_labelme_to_yolo` را با این دایرکتوری‌ها به عنوان آرگومان فراخوانی می‌کند.

این کد برای تسهیل تبدیل داده‌های برچسب‌گذاری از فرمت LabelMe به فرمت YOLO که معمولاً برای آموزش مدل‌های تشخیص اشیاء استفاده می‌شود، طراحی شده است.

در مرحله ی بعدی کتابخانه هاه مربوطه را import کرده ام `IPython.display`:

این ماژول توابعی برای نمایش اجزای مختلف در محیط نوت‌بوک IPython (مانند Jupyter Notebook) ارائه می‌دهد.

در تابع بعدی get_video این تابع یک تابع جاوااسکریپت تعریف می‌کند که ویدیو را با استفاده از دوربین وب ضبط می‌کند. در ابتدا، عناصر `div` و `video` ایجاد شده و استریم ویدیو از دوربین وب به عنصر `video` متصل می‌شود.

```
def get_video():
    js = Javascript('''
    async function recordVideo() {
      const div = document.createElement('div');
      const video = document.createElement('video');
      const stream = await navigator.mediaDevices.getUserMedia({video: true});

      document.body.appendChild(div);
      div.appendChild(video);
      video.srcObject = stream;
      video.play();
```

ضبطکننده رسانه (MediaRecorder) ایجاد شده و به استریم ویدیو متصل می‌شود. داده‌های ضبط شده در آرایه `chunks` ذخیره می‌شوند. ویدیو به مدت ۵ ثانیه ضبط می‌شود و سپس ضبط کننده متوقف می‌شود. سپس استریم ویدیو متوقف شده و عناصر `video` و `div` از صفحه حذف می‌شوند.

در مرحله ی بعدی داده‌های ویدیویی به یک شیء Blob تبدیل شده و یک URL موقت برای دانلود آن ایجاد می‌شود. سپس یک لینک دانلود ایجاد شده و کلیک می‌شود تا ویدیو به صورت یک فایل mp4 دانلود شود و در آخر نیز تابع فراخوانی می شود.

در قسمت بعدی در ابتدا مدل از پیش آموزش دیده YOLOv5s را از مخزن `ultralytics/yolov5` بارگذاری می‌کند.

این تابع برای پردازش یک فایل ویدیو طراحی شده است:

- `video_path`: مسیر فایل ویدیو که قرار است پردازش شود.
- `(cap = cv2.VideoCapture(video_path`: ویدیو را باز می‌کند.
- درون یک حلقه:
  - فریم‌های ویدیو خوانده می‌شوند.
  - اگر فریمی موجود نباشد، حلقه متوقف می‌شود.
  - استنتاج با مدل YOLO روی فریم انجام می‌شود.
  - نتایج استنتاج رندر می‌شوند.

- برچسب‌ها و جعبه‌های مرزی از نتایج استخراج و چاپ می‌شوند.
- فریم نتیجه نمایش داده می‌شود.
- اگر کلید 'q' فشرده شود، حلقه متوقف می‌شود.

- پس از اتمام حلقه، ویدیو آزاد و تمام پنجره‌ها بسته می‌شوند و در آخر نیز تابع را فراخوانی می کنیم.

```python
model = torch.hub.load('ultralytics/yolov5', 'yolov5s', pretrained=True)

# Function to process video file
def process_video(video_path):
    cap = cv2.VideoCapture(video_path)
    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break
        results = model(frame)
        results.render()
        for *box, conf, cls in results.xyxy[0]:
            x1, y1, x2, y2 = map(int, box)
            label = model.names[int(cls)]
            confidence = conf.item()
            print(f"Label: {label}, Confidence: {confidence:.2f}, Bounding box: ({x1}, {y1}, {x2}, {y2})")
        frame = results.imgs[0]
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        cv2_imshow(frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    cap.release()
    cv2.destroyAllWindows()
process_video('/content/webcam.mp4')
```

در قسمت آخر نیز دستور یک اسکریپت Python را اجرا می‌کند که برای تشخیص اشیاء در یک فایل ویدیویی استفاده می‌شود.

```
!python /content/yolov5/detect.py --source /content/webcam.mp4
```

اما به دلیل پیدا نکردن فایل webcam.mp4 و وقت بسیار کمم در این لحظات آخر نتوانستم خروجی بگیرم ولی کد به احتمال بسیار زیادی درست کار می کند.