پروژه تشخیص خودرو های ایرانی

مریم واقعی[[1]](#footnote-1)

مریم صابری

|  |  |  |
| --- | --- | --- |
| اطلاعات گزارش |  | چکیده |
| **تاریخ:** |  | در اين گزارش‏‌ قرار است پروژه ی خود در بحث تشخیص خودرو های ایرانی به همراه مدل های آنها را توضیح دهیم. ما در این گزارش روند کار خود و همچنین نتایج خود را شرح داده ایم. |
| **واژگان كليدي:** |  |

فهرست مطالب

[مقدمه 2](#_Toc106119504)

[بخش اول: جمع آوری مجموعه داده 3](#_Toc106119505)

[بخش دوم: پیش پردازش داده ها 3](#_Toc106119506)

[حذف داده های پرت 3](#_Toc106119507)

[برچسب زدن عکس ها 4](#_Toc106119508)

[بخش سوم: مدل استفاده شده برای تشخیص خودرو 8](#_Toc106119509)

[آشنایی با مدل های تشخیص اشیا 8](#_Toc106119510)

[منابع و مراجع 12](#_Toc106119511)

# مقدمه

اکنون با پیشرفت فناوری و ورود به عصر تکنولوژی و هوش مصنوعی، بشر در مسیر خودکارسازی بسیاری از کار ها من جمله ربات های صنعتی، خودرو های خودران، سیستم های تشخیص چهره و... قرار گرفته است. بشر سعی میکند تا با استفاده از هوش مصنوعی و یادگیری عمیق سیستم هایی با دقت بالا و ایمن برای انسان ها طراحی کنند تا زندگی برای انسان ها راحتتر و ایمن تر شود.

ما نیز در این پروژه سعی کردیم با استفاده از مدل های یادگیری عمیق، مدلی برای تشخیص خودروهای ایرانی آموزش دهیم. در این گزارش ما روند کار خود از ابتدایی ترین قدم یعنی جمع آوری داده ها تا قدم نهایی یعنی ارزیابی مدل را توضیح می دهیم.

# بخش اول: جمع آوری مجموعه داده

# بخش دوم: پیش پردازش داده ها

## حذف داده های پرت

دیتاست ما از سایت دیوار جمع آوری شده بنابراین ممکن است برخی عکس ها مناسب مدل ما نباشد. هدف ما تشخیص خودرو های ایرانی است و از آنجایی که برخی عکس های خودرو ها در سایت دیوار از داخل ماشین ها یا از چرخ و ... است و این داده ها برای مدل ما مناسب نیستند لذا باید این عکس ها را از دیتاست خود حذف کنیم تا دیتاست با کیفیت تری داشته باشیم. برای اینکار لازم است تا مدلی داشته باشیم که عکس های خودرو را از عکس های دیگر (مثل عکس های چرخ خودرو، داخل خودرو و...) را تفکیک کند و تشخیص دهد. برای اینکار ما از داده هایی که داشتیم به صورت رندوم دیتاستی ایجاد کردیم که یک دسته شامل عکس های خودرو ها از زوایای مختلف و دسته دیگر شامل عکس هایی از خودرو ها مثل داخل خودرو یا چرخ و ... است.



تصویر 1- نمونه ای از تصاویر دسته خودرو تصویر 2 - نمونه ای از تصاویر دسته غیر خودرو

ما 3 مدل را برای اینکار در نظر گرفتیم و آموزش دادیم[[2]](#footnote-2) که دقت و مقدار خطای مدل ها به شرح زیر است:

|  |  |  |
| --- | --- | --- |
| نام مدل | دقت مدل | مقدار Loss |
| Inception\_resnet\_v2 | 0.985 | 0.044 |
| Inception\_v3 | 0.980 | 0.053 |
| mobilenet | 0.995 | 0.036 |

جدول 1 - مقدار دقت و خطای 3 مدل برای تشخیص داده های پرت از داده های مناسب

همانطور که میبینید این مدل ها دقت مناسبی دارند اما خب هر کدام از این مدل ها در تعداد بسیار کمی از عکس ها ممکن است به اشتباه کلاس داده را تشخیص دهند ما برای آنکه این خطا را به حداقل برسانیم 3 مدل را با هم ترکیب کردیم و یک مدل به نام مدل ensemble ساختیم که لایه نهایی آن یک لایه Average است که از خروجی 3 مدل میانگین میگیرد. در نهایت دقت مدل ensemble برابر با 0.995 و مقدار خطای آن به 0.028 رسید و از آنجایی که ما میخواهیم داده های پرت را حذف کنیم میخواهیم از یک مدل با حداکثر دقت و حداقل خطا استفاده کنیم که مدل ensemble برای اینکار گزینه مناسبی است.

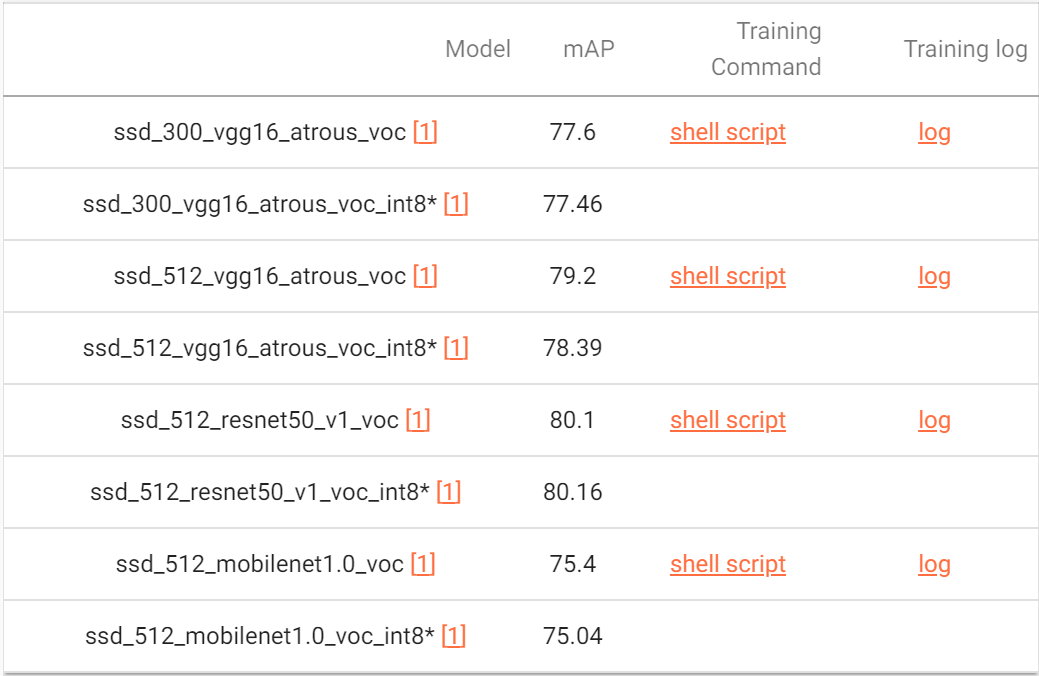
نکته ای که لازم است به آن اشاره کنیم این است که چون ما برای این بخش دیتاست کوچکی در نظر گرفتیم که تنها شامل 500 داده آموزش و 100 داده validation و 200 داده آزمون است بنابراین به روش transfer learning ابتدا وزن های imagenet برای این مدل ها را به عنوان وزن اولیه در نظر گرفتیم و سپس مدل را روی دیتاست خود آموزش دادیم.

## برچسب زدن عکس ها

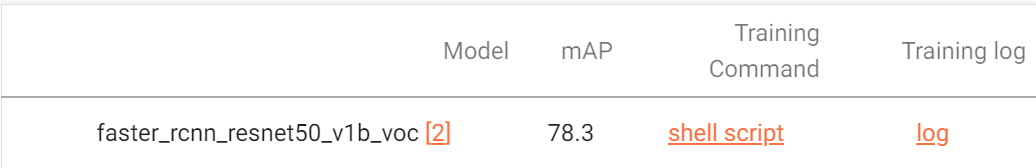
همانطور که میدانید ما حدود 100 هزار عکس از انواع ماشین ها را از سایت دیوار دریافت کرده ایم و حالا نیاز داریم تا لیبل آنها را مشخص کنیم. اگر پروژه تنها شامل طبقه بندی بود، لیبل گذاری بسیار راحت بود و تنها با گذاشتن عکس هر نوع ماشین در پوشه مربوطه، لیبل آن عکس مشخص میشد اما ما در این پروژه علاوه بر طبقه بندی، تشخیص مکان ماشین را نیز داریم پس لازم است تا مختصات ماشین که در اینجا شامل دو نقطه‌ی گوشه بالا سمت چپ و گوشه پایین سمت راست است نیز در برچسب عکس ها مشخص شده باشد. ما اگر بخواهیم برای تک تک عکس ها برچسب گذاری را به صورت دستی انجام دهیم، از آنجایی که تعدادعکس ها بسیار زیاد است در نتیجه برچسب گذاری آنها به شدت زمان خواهد برد. بنابراین باید به دنبال روش های برچسب گذاری اتوماتیک یا شبه اتوماتیک باشیم.

برای همین ما تحقیق کردیم تا ببینیم از بین مدل های از پیش آموزش داده شده، دقت کدام مدل بیشتر است.

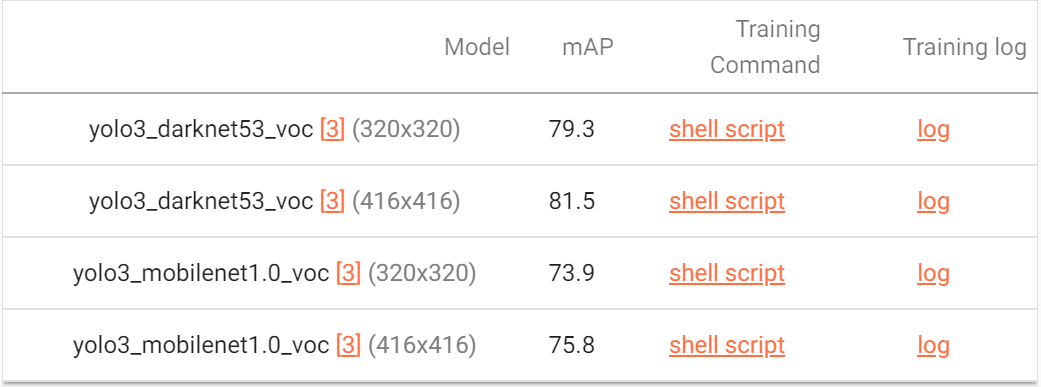
درنهایت پس از بررسی های مختلف به کتابخانه model\_zoo از ماژول gluoncv رسیدیم که این کتابخانه در بحث تشخیص اشیا شامل مدل های مختلف از پیش آموزش دیده روی دیتاست های COCO و Pascal VOC می باشد که در ادامه برخی از آنها را مشاهده میکنید:



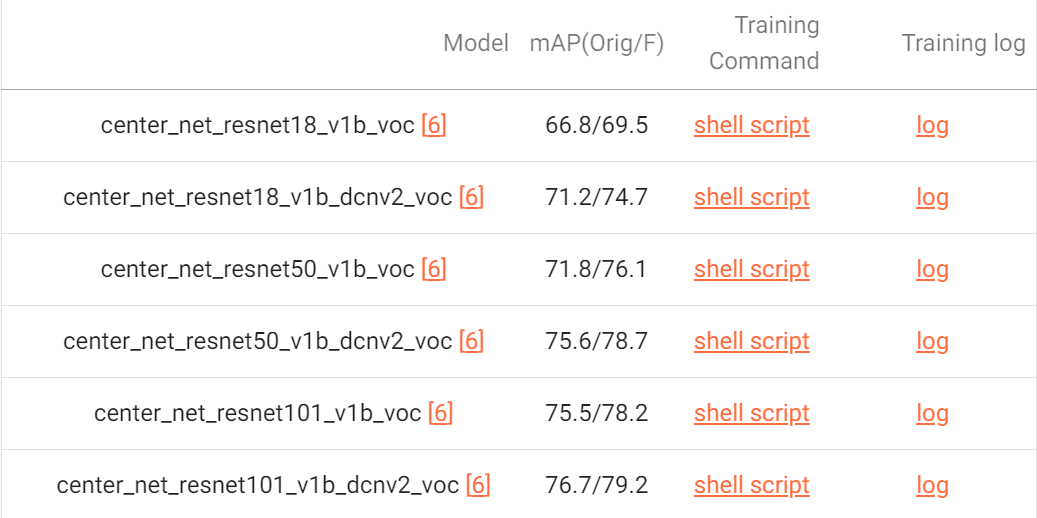
تصویر 1 – مدل های مختلف SSD از پیش آموزش دیده با دقت آن ها روی دیتاست Pascal VOC



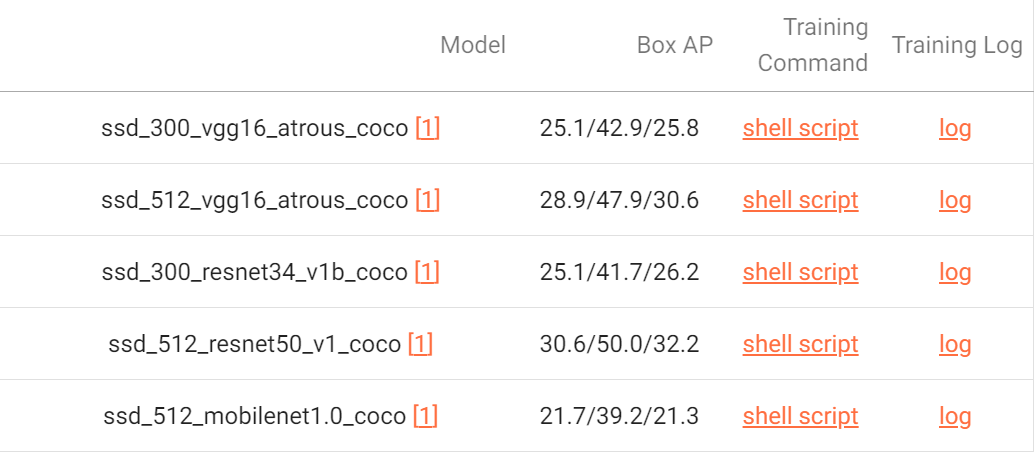
تصویر 2 - مدل های مختلف Faster RCNN از پیش آموزش دیده با دقت آن ها روی دیتاست Pascal VOC



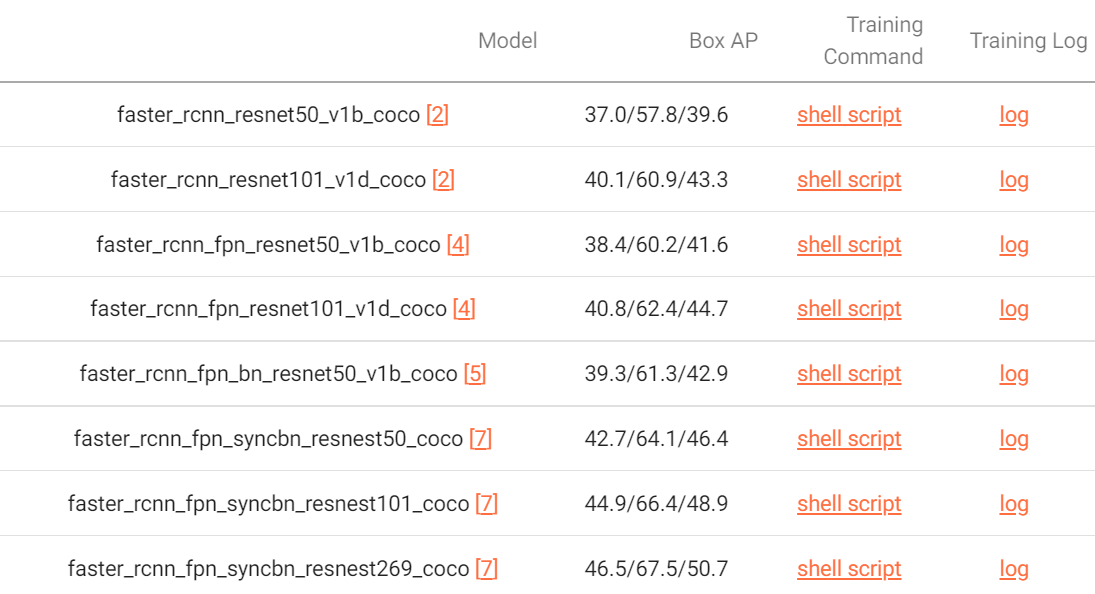
تصویر 3 - مدل های مختلف YOLO V3 از پیش آموزش دیده با دقت آن ها روی دیتاست Pascal VOC



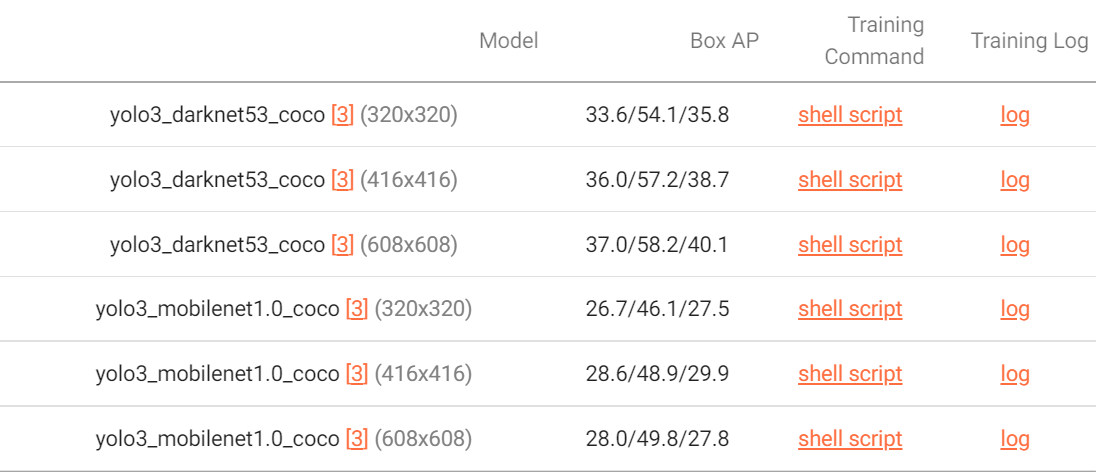
تصویر 4 - مدل های مختلف CenterNet از پیش آموزش دیده با دقت آن ها روی دیتاست Pascal VOC



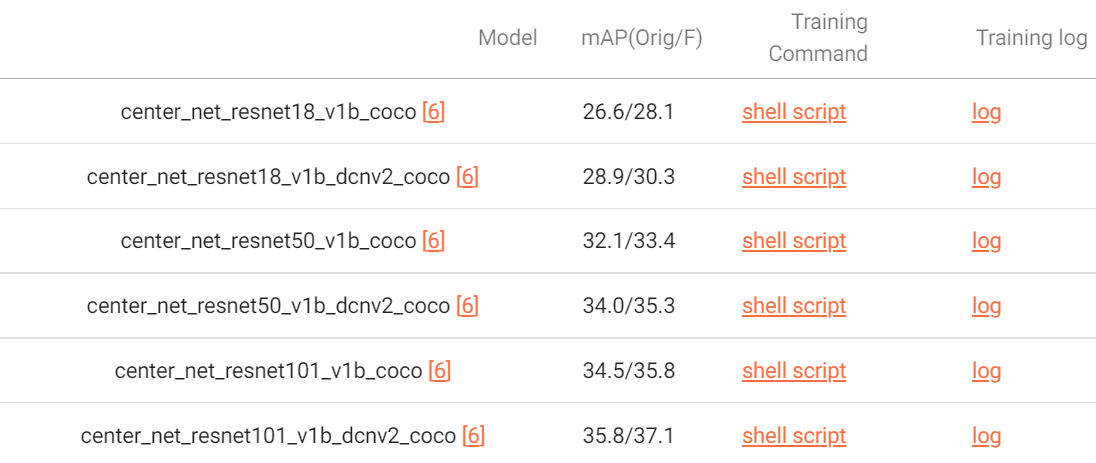
تصویر 5 - مدل های مختلف SSD از پیش آموزش دیده با دقت آن ها روی دیتاست Pascal VOC



تصویر 6 - مدل های مختلف Faster RCNN از پیش آموزش دیده با دقت آن ها روی دیتاست Pascal VOC



تصویر 7 - مدل های مختلف YOLO V3 از پیش آموزش دیده با دقت آن ها روی دیتاست Pascal VOC



تصویر 8 - مدل های مختلف CenterNet از پیش آموزش دیده با دقت آن ها روی دیتاست Pascal VOC

ما تمام مدل های از پیش آموزش دیده‌ی این کتابخانه روی دیتاست Coco و Pascal VOC را تست کردیم و در نهایت 3 تا از مدل ها دقت مناسبی روی ماشین ها و تشخیص مختصات آنها داشتند که به شرح زیر می باشند:

1. faster\_rcnn\_resnet50\_v1b\_voc با دقت 78.3 mAP
2. ssd\_512\_resnet50\_v1\_voc با دقت 80.1 mAP
3. yolo3\_darknet53\_voc با دقت 81.5 mAP

این مدل ها از لحاظ تشخیص مختصات ماشین ها تقریبا مشابه هم بودند و از بین تمام مدل ها دقت بیشتری داشتند.

اما ما به دنبال مدلی هستیم که هم از لحاظ دقت و هم از لحاظ سرعت مدل بهتری باشد. پس این سه مدل را از لحاظ سرعت پیش بینی نیز مقایسه کردیم که به ترتیب سرعت به شرح زیر می باشند:

1. ssd\_512\_resnet50\_v1\_voc با دقت 80.1 mAP
2. yolo3\_darknet53\_voc با دقت 81.5 mAP
3. faster\_rcnn\_resnet50\_v1b\_voc با دقت 78.3 mAP

بنابراین چون بیشترین سرعت مربوط به مدل ssd\_512\_resnet50\_v1\_voc بود لذا تصمیم گرفتیم تا از این مدل برای لیبل گذاری داده های خود استفاده کنیم و اشیائی که امتیاز آنها از یک آستانه (که در اینجا 0.8 است) بیشتر است آن ها را در نظر میگیریم.

در دیتاست ما تعداد کمی از عکس ها شامل ماشین های دیگری در پس زمینه ی ماشین اصلی نیز می باشد و مشخصا مدلی که در نظر گرفتیم، این ماشین های پس زمینه را نیز تشخیص میدهد. اما چالشی که ما به آن برخوردیم و این چالش تنها در تعداد بسیار کمی از عکس ها به چشم میخورد اما با این حال میخواستیم برای آنها نیز تشخیص مناسبی داشته باشیم، این است که در حین تشخیص ماشین های داخل یک عکس، این مدل و خیلی از مدل های دیگری که آنها را تست کردیم ضریب اطمینان مربوط به برخی ماشین های پس زمینه را بیشتر از ماشین اصلی میدادند، لذا ما اگر بر اساس امتیاز (ضریب اطمینان) بخواهیم ماشین ها را انتخاب کنیم ممکن است ماشینی در پس زمینه انتخاب شود که مدل آن الزاما با مدل ماشین مدنظر ما یکی نباشد. مثلا فرض کنید یک تصویر داریم که ماشین اصلی در آن سمند است و در پس زمینه چند ماشین از جمله یک پراید می باشد و ما میدانیم در این عکس به دنبال پیدا کردن سمند و مکان آن در تصویر هستیم. حال اگر ضریب اطمینان پراید از سمند بیشتر باشد (مثلا برای پریاد ضریب اطمینان 0.99 و سمند 0.95 باشد) و ما براساس بیشترین ضریب اطمینان شی مدنظر را انتخاب کنیم در نهایت به اشتباه مکان سمند داخل عکس را تشخیص داده ایم.



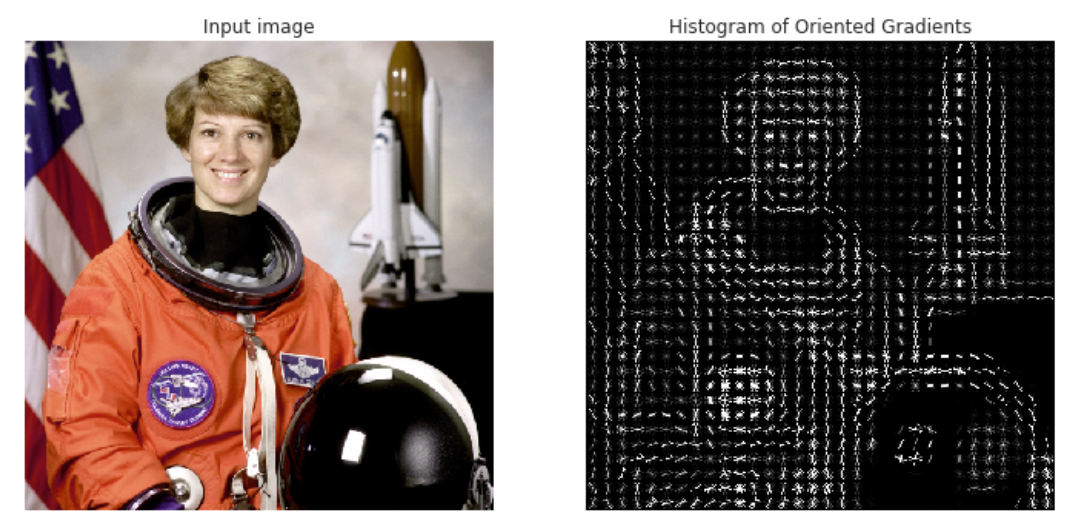
تصویر 9- تصویری که در آن ضریب اطمینان مربوط به ماشین اصلی کمتر از ماشین پس زمینه می باشد.

مدل ما باکس ها را با دقت خیلی بالایی تشخیص میدهد و به گونه ای میتوانیم بگوییم که میزان درستی باکس ها از ضریب اطمینانی که مدل به ما میدهد بیشتر است. درنتیجه ما برای جلوگیری از این مشکل آمدیم از ایده ی انتخاب شی با بیشترین مساحت bounding box استفاده کردیم. به این صورت که از بین تمام ماشین هایی که مقدار ضریب اطمینان آنها از آستانه بیشتر است، از آنجایی که ماشین اصلی در عکس قطعا بیشترین مساحت از عکس را اشغال میکند، لذا ما باکسی که بیشترین مساحت را دارد انتخاب میکنیم.

# بخش سوم: مدل استفاده شده برای تشخیص خودرو

## آشنایی با مدل های تشخیص اشیا

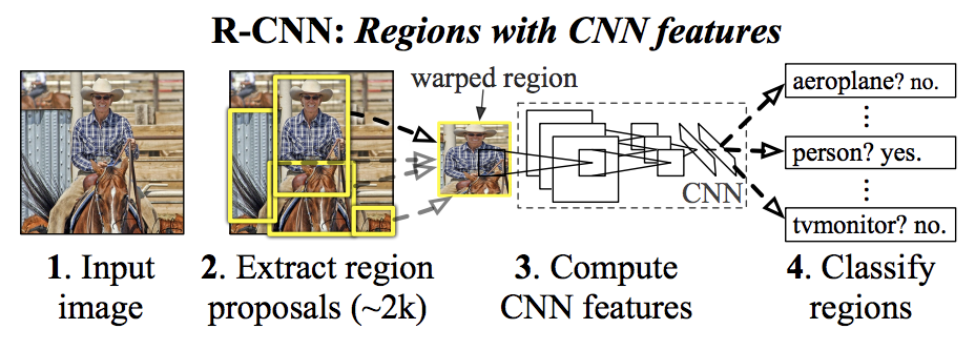
در زمان های قدیم و قبل از اینکه الگوریتم های عمیق ارائه شوند، از مدل هایی مثل HOG، HARR، SIFT، SURF و... برای بحث تشخیص اشیا استفاده میشد که امروز این مدل ها کاربرد زیادی ندارند اما همچنان ممکن است از آنها بسته به نیاز استفاده شود. این مدل ها از هیچ شبکه عمیقی استفاده نمیکنند، مثلا الگوریتم [[3]](#footnote-3) HOG با استفاده از بردار ها از تصاویر ویژگی ها را استخراج میکند به این صورت که در هر پیکسل میاد پیکسل های اطرافش را بررسی میکند و از روشن ترین پیکسل به تیره ترین پیکسل یک بردار رسم میکند و به این ترتیب یکسری بردار بدست می آید. پس از اینکه بردار های زیادی بدست آوردیم فضاهای n\*n مثلا 10\*10 یا 4\*4 یا ... در نظر میگیریم و روی کل بردار ها حرکت میکنیم و در هر بلوک از بردار ها فلشی که بیشترین تعداد را دارد جایگزین در آن بلوک میکنیم.



تصویر 10- استخراج ویژگی از تصاویر در روش HOG با استفاده از بردارها

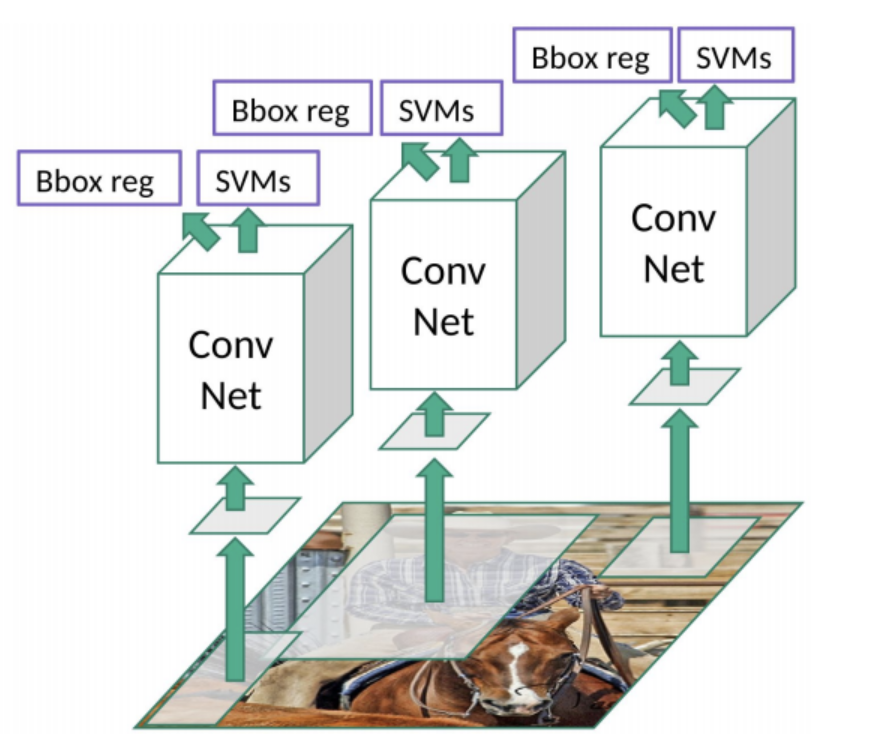
اما خب به دلیل بزرگ بودن مسئله ما، مدل های قدیمی پاسخگو نیستند و ما نیاز به مدل های شبکه عمیق داریم. برای همین سراغ مدل های [[4]](#footnote-4)RCNN و Fast RCNN و Faster RCNN و در نهایت مدل YOLO[[5]](#footnote-5) رفتیم.

الگوریتم RCNN در قدم اول یک سری ناحیه های پیشنهادی که احتمال وجود شی در آن ها هست را با استفاده از الگوریتم های ابتدایی مثل selective search [[6]](#footnote-6) بدست می آورد. بعد از بدست آوردن ناحیه های پیشنهادی و مختصاتشان لازم است آنها را از تصویر اصلی برش دهیم و پس از آنکه به یک سایز مشخص آنها را تغییر دادیم برای استخراج ویژگی ها و آن را به سمت لایه های convolution هدایت میکنیم و پس از آن به کمک الگوریتم های دسته بند مثل SVM ، دسته و نوع شی را تشخیص میدهد.



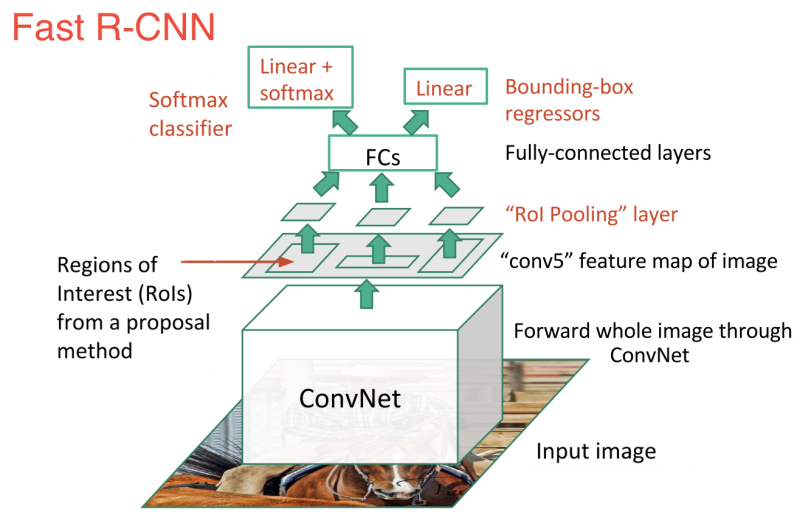
تصویر 11 – نمای کلی از نحوه کار الگوریتم RCNN

اما مشکل RCNN این هست که به دلیل اینکه در چند مرحله کار انجام میدهیم و یکپارچه نیست در نتیجه کند است یعنی ما ابتدا از عکس با الگوریتم سنگین و کند selective search حدودا 2000 تا ناحیه پیشنهادی بدست آوردیم بعد از آن عکس را برش دادیم و بعد از یک شبکه CNN برای بدست آوردن یکسری ویژگی استفاده کردیم، پس از آن با استفاده از دسته بند SVM نوع اشیا را تشخیص میدهیم وبه ازای هر ناحیه پیشنهادی ما این عملیات را انجام میدهیم که بسیار زمان بر است و حدود 47 ثانیه به ازای هر عکس زمان می برد که مدت زمان بالایی است و در نتیجه این الگوریتم برای کار های real time مناسب نیست.



تصویر 12 - نحوه عملیات الگوریتم RCNN روی ناحیه های پیشنهادی

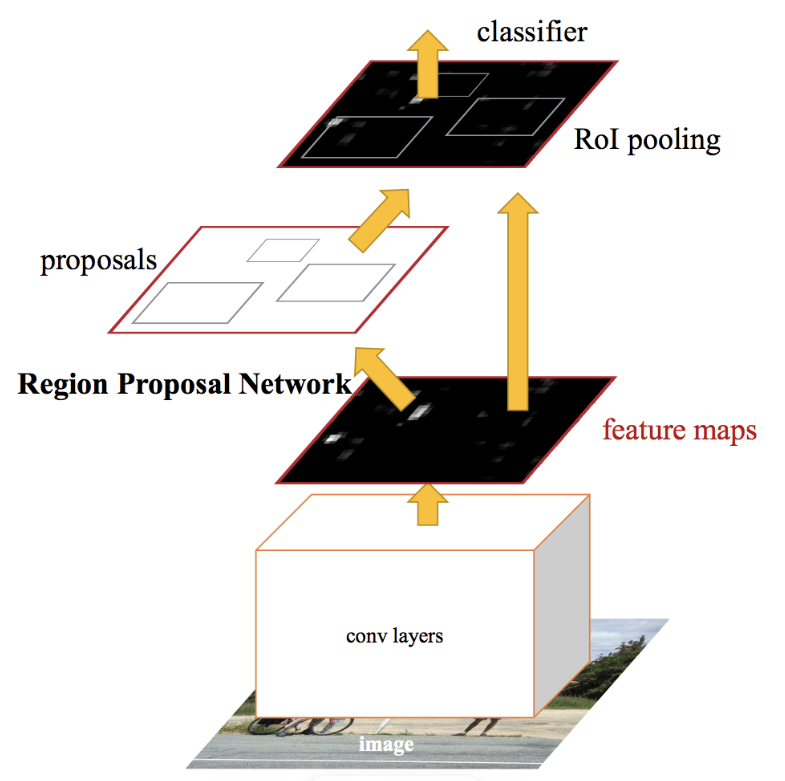
در fast RCNN رویکرد مشابه الگوریتم R-CNN است. اما به جای دادن تک تک ناحیه های پیشنهادی به CNN، تصویر ورودی را به CNN می‌دهیم تا یک نگاشت ویژگی های کانولوشنی ایجاد کنیم. از نگاشت ویژگی کانولوشنال، ناحیه های پیشنهادی را شناسایی کرده و آنها را به شکل مربع در می آوریم و با استفاده از یک لایه RoI Pooling آنها را به اندازه ثابت تغییر شکل می دهیم تا بتوان آن را به یک لایه کاملا متصل وارد کرد. در نهایت از بردار ویژگی RoI، ما از یک لایه softmax برای پیش‌بینی کلاس ناحیه پیشنهادی استفاده می‌کنیم. شکل زیر نمای کلی از نحوه کار Fast RCNN می باشد:



تصویر 13 - نحوه عملیات الگوریتم Fast RCNN روی ناحیه های پیشنهادی

هر دو الگوریتم فوق یعنی R-CNN و Fast R-CNN از selective search برای یافتن ناحیه های منطقه استفاده می کنند. الگوریتم selective search یک فرآیند کند و زمان بر است که بر عملکرد شبکه تأثیر می گذارد. بنابراین در ورژن بعدی RCNN یعنی Faster RCNN یک الگوریتم تشخیص اشیا ارائه کردند که الگوریتم selective search را حذف می‌کند و به شبکه اجازه می‌دهد تا ناحیه های پیشنهادی را یاد بگیرد و این الگوریتم RPN[[7]](#footnote-7) نام دارد.

Faster RCNN مشابه Fast R-CNN، تصویر را به عنوان ورودی به یک شبکه کانولوشن ارائه می کند که این شبکه یک نگاشت ویژگی کانولوشنی ارائه می دهد. به جای استفاده از الگوریتم selective search بر روی feature map برای شناسایی ناحیه های پیشنهادی، از یک شبکه جداگانه برای پیش بینی ناحیه های پیشنهادی استفاده می شود. سپس ناحیه های پیشنهادی پیش‌بینی‌شده با استفاده از یک لایه RoI Pooling تغییر شکل می‌دهند و بعد از آن برای طبقه‌بندی ناحیه های پیشنهادی داخل تصویر به classifier داده میشود.



تصویر 14- نحوه عملیات الگوریتم Faster RCNN

در جدول زیر میزان سرعت و دقت شبکه های خانواده RCNN را مشاهده میکنید:

|  |  |  |  |
| --- | --- | --- | --- |
|  | RCNN | Fast RCNN | Faster RCNN |
| زمان تست به ازای هر تصویر | 50 ثانیه | 2 ثانیه | **0.2 ثانیه** |
| میزان افزایش سرعت | 1x | 25x | **250x** |
| میانگین دقت متوسط (روی دیتاست Pascal VOC 2007) | 66.0 mAP | **66.9 mAP** | **66.9 mAP** |

جدول 2- مقایسه سرعت و دقت شبکه های RCNN و Fast RCNN و Faster RCNN

الگوریتم YOLO که توسط Joseph Redmon در سال 2016 ارائه شد که ایده ای کاملا متفاوت از الگوریتم های RCNN دارد. این الگوریتم دیگر مبتنی بر نواحی پیشنهادی نیست و به صورت مستقیم کل عملیات تشخیص اشیا را انجام میدهد. هدف اصلی در YOLO سرعت بالای تشخیص اشیا است اما دقت YOLO کمتر از faster RCNN می باشد.

نحوه کار YOLO به این صورت است که ما یک تصویر می‌گیریم و آن را به یک شبکه SxS تقسیم می‌کنیم، در هر یک از شبکه‌ها، m تا bounding box را می‌گیریم. برای هر یک از bounding box ها، شبکه یک احتمال کلاس و مختصات را برای bounding box خروجی می دهد. Bounding box هایی که احتمال کلاس بالاتر از مقدار آستانه دارند می شوند.

YOLO نسبت به سایر الگوریتم‌های تشخیص شی سریع‌تر (45 فریم در ثانیه) است. محدودیت الگوریتم YOLO این است که ممکن است اجسام کوچک درون تصویر را به سختی تشخیص دهد یا کلا تشخیص ندهد، به عنوان مثال ممکن است در تشخیص دسته ای از پرندگان مشکل داشته باشد. این به دلیل محدودیت های مکانی الگوریتم است. ورژن های 2و3 YOLO که توسط همین آقای Redmon ارائه شدند در ایده کلی YOLO تغییری ایجاد نکردند بلکه با انجام یکسری تغییرات و آزمایشات بر روی ورژن های قبل به بهبود در سرعت و دقت رسیدند. با این حال، در فوریه 2020، جوزف ردمون، خالق YOLO اعلام کرد که تحقیقات خود را در بینایی کامپیوتر متوقف کرده است.

اما خروج ردمون پایان YOLO نبود و در آوریل سال 2020 ورژن 4 YOLO توسط Alexey Bochkovskiy و همکارانش معرفی شد و سریع ترین و دقیق ترین مدل بلادرنگ برای تشخیص اشیا در نظر گرفته شده است. YOLO v4 همچنین بر اساس Darknet است و مقدار میانگین دقت 43.5 درصد را در مجموعه داده COCO به همراه سرعت بلادرنگ 65 فریم در ثانیه بر روی gpu Tesla V100 به دست آورده است که از نظر سرعت و دقت، سریع ترین و دقیق ترین تشخیص دهنده ها را شکست می دهد. در مقایسه با YOLO v3 میانگین دقت 10 درصد و تعداد فریم در ثانیه 20 درصد افزایش داشته است.

پس از انتشار YOLO v4، تنها در عرض دو ماه، نسخه دیگری از YOLO به نام YOLO v5 منتشر شد! این نسخه توسط Glenn Jocher ایجاد شده که قبلاً در بین جامعه برای ایجاد پیاده‌سازی PyTorch از YOLO v3 شناخته شده است.

در 9 ژوئن 2020، Jocher اظهار داشت که پیاده سازی YOLO v5 به صورت عمومی منتشر شده است و توصیه می شود در پروژه های جدید استفاده شود. با این حال، او هنگام انتشار این نسخه جدید، مقاله ای را برای همراهی با انتشار خود منتشر نکرد.

YOLO v5 با سایر نسخه‌های قبلی متفاوت است، زیرا این یک پیاده‌سازی PyTorch است نه fork ای از Darknet اصلی. همانند YOLO v4، YOLO v5 دارای backbone CSP و neck PA-NET است. پیشرفت‌های عمده شامل mosaic data augmentation و یادگیری خودکار bounding box anchor ها است.

انتشار YOLO v5 توجه زیادی را به خود جلب کرده و بحث های داغی را در پلتفرم های جامعه یادگیری ماشین ایجاد کرده است. این عمدتا به دلیل چندین واقعیت در مقاله منتشر شده توسط تیم Roboflow در مورد YOLO v5 بود.

این مقاله با عنوان "YOLOv5 is Here" در 10 ژوئن 2020 در وبلاگ Roboflow منتشر شده است و چندین واقعیت مهم را بیان می کند. در ادامه چند نقل قول از آن پست وبلاگ توسط جوزف نلسون و یاکوب سولاوتز آمده است.

« با اجرای Tesla P100، زمان استنتاج تا 0.007 ثانیه در هر تصویر، یعنی 140 فریم در ثانیه (FPS) را مشاهده کردیم! در مقابل، YOLO v4 پس از تبدیل شدن به همان کتابخانه Ultralytics PyTorch به 50 FPS دست یافت.»

« YOLO v5کوچک است. به طور خاص، یک فایل وزن برای YOLO v5 27 مگابایت است. فایل وزن ما برای YOLO v5 (با معماری Darknet) 244 مگابایت است. YOLO v5 تقریبا 90 درصد کوچکتر از YOLO v4 است. »

پس از مقایسه و ارزیابی مدل ها به دلیل اینکه ما یک مدل بلادرنگ با حجم بسیار کم نیاز داریم که بتوان آن را روی سیتم های سبک مثل گوشی اجرا کرد، لذا تصمیم گرفتیم تا از مدل YOLO v5 برای مسئله خود استفاده کنیم.

# منابع و مراجع

[1] <https://iq.opengenus.org/object-detection-with-histogram-of-oriented-gradients-hog/>

[2] <https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e>

[3] <https://towardsdatascience.com/yolo-v4-or-yolo-v5-or-pp-yolo-dad8e40f7109>

[4] <https://dzone.com/articles/from-r-cnn-to-faster-r-cnn-the-evolution-of-object>

1. پست الکترونیکی: mwaghei78@gmail.com [↑](#footnote-ref-1)
2. مدل ها شامل 3 مدل inception\_resnet\_v2 و inception\_v3 و mobilenet می باشد. [↑](#footnote-ref-2)
3. histogram of oriented gradients [↑](#footnote-ref-3)
4. Region Based Convolutional Neural Networks [↑](#footnote-ref-4)
5. you only live once [↑](#footnote-ref-5)
6. در روش selective search ناحیه ها بر اساس میزان شباهت رنگ، بافت، سایز و شکل از هم جدا میشوند. [↑](#footnote-ref-6)
7. Region Proposal Network [↑](#footnote-ref-7)