پروژه تشخیص خودرو های ایرانی

مریم واقعی[[1]](#footnote-1)

مریم صابری

|  |  |  |
| --- | --- | --- |
| اطلاعات گزارش |  | چکیده |
| **تاریخ: 3 مرداد 1401** |  | در اين گزارش‏‌ قرار است پروژه ی خود در بحث تشخیص خودرو های ایرانی به همراه مدل استفاده شده برای تشخیص اشیا را توضیح دهیم. ما در این گزارش روند کار خود شامل جمع آوری داده ها و پیش پردازش آنها، روند تحقیقات برای انتخاب مدل مناسب و آماده سازی محیط برای آموزش مدل و ارزیابی مدل و همچنین چالش های مختلف که در حین انجام پروژه به آنها برخوردیم را شرح داده ایم. ما برای انجام بحث تشخیص اشیا از مدل yolov5 و همچنین از متریک های دقت و فراخوانی و میانگین دقت متوسط و همچنین میزان خطای مدل برای ارزیابی مدل آموزش دیده استفاده کرده ایم. |
| **واژگان كليدي:**  **شبکه عصبی**  **یادگیری ماشین**  **بینایی کامپیوتر**  **پیش پردازش داده ها**  **برچسب گذاری عکس ها**  **آموزش مدل**  **مدل yolov5** |  |

فهرست مطالب

[مقدمه 2](#_Toc109639880)

[بخش اول: جمع آوری مجموعه داده 3](#_Toc109639881)

[آماده سازی دیتابیس اولیه 4](#_Toc109639882)

[الگوریتم داده گیری 7](#_Toc109639883)

[جمع بندی 12](#_Toc109639884)

[بخش دوم: پیش پردازش داده ها 12](#_Toc109639885)

[حذف عکس های نامعتبر 12](#_Toc109639886)

[حذف داده های پرت 12](#_Toc109639887)

[برچسب زدن عکس ها 13](#_Toc109639888)

[بخش سوم: مدل استفاده شده برای تشخیص خودرو 17](#_Toc109639889)

[آشنایی با مدل های تشخیص اشیا 17](#_Toc109639890)

[نحوه استفاده از مدل YOLOv5 22](#_Toc109639891)

[چالش های آموزش مدل 23](#_Toc109639892)

[بخش چهارم: ارزیابی و نتایج مدل نهایی 24](#_Toc109639893)

[بهبود های بعدی 26](#_Toc109639894)

[منابع و مراجع 27](#_Toc109639895)

# 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

برای جمع آوری دیتاست ماشین های ایرانی چندین گام طی شده است که شرح زیر می باشد.

الف:جستجو و یافتن اسم انواع مدل های ماشین ایرانی :

Table

Description automatically generatedجهت دریافت این اطلاعات به سایت شرکت های خودرو سازی اصلی و بزرگ در ایران شامل سایپا ، ایران خودرو ، مدیران خودرو ، کرمان موتور و گروه خودروسازی بهمن مراجعه کرده و مطابق با نام محصولات در هر شرکت اسم مدل یا برند مطابق با سایت دیوار را دریافت نمودیم.نتیجه ی بدست آمده در جدول 1 مشخص شده است.

تصویر 1- مدل های ایرانی

ب: دریافت لیست کل مدل های ماشین موجود در سایت دیوار :

در اولین گام مدل های ماشین موجود در سایت دیوار داده گیری شدند تا بتوان از بین آن ها مدل های ماشین ایرانی را جداسازی و فیلتر نمود.۱۳۷۹مدل ماشین در سایت دیوار دریافت شدند.اکسل به دست آمده در پیوست قرار گرفته است.

ج. یافتن تعداد پست های مدل های ماشن ایرانی موجود در سایت دیوار :

با توجه به تنوع مدل های ماشین و اهمیت تعداد عکس های موجود از هر مدل در سایت دیوار در اولین گام می بایست بیابیم که آیا می توان برای همه ی مدل ها به میزانی عکس استخراج نمود که برای اموزش و استفاده در شبکه عصبی ما به کار بیاید.

بدین منظور ما در ابتدا با شهر تهران شروع کردیم و تعداد پست های موجود از هر مدل را دریافت نمودیم.

با در نظر گرفتن اینکه اگر به طور میانگین هر پست 2 تا 3 عکس داشته باشید و ما برای یادگیری ماشین هر مدل به حداقل 1000 عکس از آن مدل احتیاج داشته باشیم ، پس اگر از یک مدل حداقل 350 تا 500 پست موجود باشد کفایت می کند.

با اجرای کد دریافت تعداد پست های شهر تهران مدل های بالای 500 پست داشتند در جدول 2 نمایش داده شدند. هم چنین مدل های مشخص شده در جدول 3 بین 200 تا 500 پست داشتند.

همان طور که مشخص است تعداد کمی از مدل ها در رنج 350 تا 500 پست قرار داشتند.بنابراین احتیاج است برای سایر مدل هایی که پست های آن ها نزدیک به این محدوده می باشد سایر شهر ها را هم دریافت کنیم.

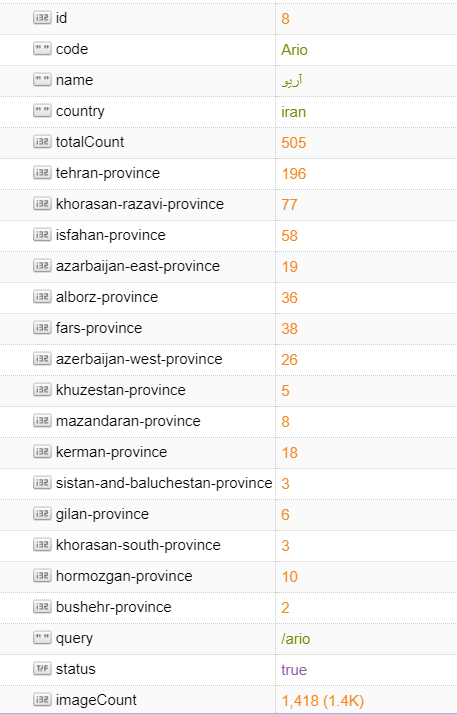
بدین منظور ما الگوریتم را برای مدل هایی که حداقل 100 تا 350 پست داشتند در شهر های سایر استان ها از جمله خراسان رضوی، اصفهان، فارس، خوزستان، آذربایجان شرقی، مازندران، البرز، کرمان، اذربایجان غربی، گیلان، هرمزگان و خراسان جنوبی اجرا نمودیم.

## آماده سازی دیتابیس اولیه

دیتابیس ذخیره سازی اطلاعات مدل ها مونگو می باشد و جهت داده گیری عکس ها و مدل ها از rest api موجود در این سایت استفاده شده است.الگوریتم داده گیری با استفاده از زبان nodejs زده شده است.در این پروژه بسته به نیاز های پیش آمده سه جدول brands ، iran-brand و cars ایجاد گردید.

در ابتدا با دریافت نام همه ی مدل ها جدول brands را ساختیم. هدف از این جدول مقدار دهی id و code منحصر به هر برند است.هم چنین پس از بررسی api های سایت دیوار برای گرفتن پست ها نیاز است کوئری مشخص هر مدل نیز زده شود، با بررسی api ها کوئری مشخص هر مدل نیز گرفته شد و در این دیتابیس قرار گرفت.

پس از مشخص شدن نام استان های مورد نیاز برای داده گیری به ازای هر مدل در جدول brands نام استان ها با تعداد حدودی پست ها که در گام قبلی یافته بودیم مقداردهی کردیم.کلیدهای نهایی در این جدول به شکل زیر مقدار دهی شدند:



مطابق مثال بالا در جدول brands برای هر رکورد فیلدstatus را نیز قرار دادیم که وضعیت دریافت عکس های مربوط به مدل را مشخص می کند.فیلد imageCount نیز مقدار کل عکس های مربوط به مدل را نگهداری می کند.

تصویر 2- جدول brands

جدول دیگری که برای داده گیری از آن استفاده کردیم iran-brands می باشد که تنها برند های ایرانی را در آن ذخیره کردیم.در این جدول برای هر مدل و هر شهر رکورد مشخصی وجود دارد که علاوه بر اطلاعات مخصوص مدل فیلد های count و lastPage را اضافه دارد.

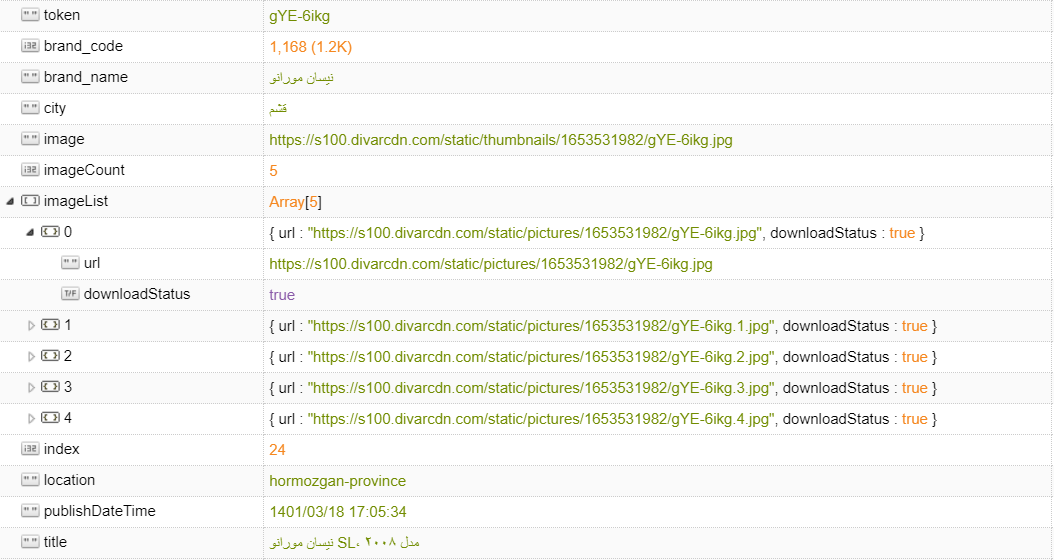
فیلد count تعداد عکس های مربوط به رکورد را نگهداری می کند.از فیلد lastPage جهت ذخیره سازی اخرین صفحه درخواست داده شده برای شهر مدنظر استفاده می کنیم.هم چنین مشابه جدول قبل برای این جدول نیز کلید status جهت مشخص شدن وضعیت داده گیری شهر مدنظر قرار داده شده است.

شکل زیر داده های نمونه ای از جدول iran-brands است.

تصویر 3- جدول iran-brands

اخرین کالکشن جهت داده گیری عکس های ماشین ها از سایت دیوار cars می باشد که در این جدول اطلاعات هر پست از سایت دیوار ذخیره شده است.در این جدول کلید منحصر به فرد token مربوط به هر پست استفاده شده است و هر پست لیستی از url عکس های مربوط به خود را دارد که در کلید imageList می باشد.

نمونه ی از مقادیر این کاکلشن در تصویر زیر قابل مشاهده است:



تصویر 4- جدول cars

## الگوریتم داده گیری

الگوریتم داده گیری در زبان nodejs و به شکل اسکریپت نوشته شد.سپس از یک runner جهت اجرای متمادی این کد استفاده شد.نمای کلی کلاس داده گیری در شکل 5 مشخص شده است.

نحوه ی کار بدین صورت است که با هر بار اجرا از لیست کل برند های موجود برند های ایرانی را انتخاب می کند سپس برای تک تک این برند ها شهر هایی که هنوز داده گیری نشدند (یعنی وضعیت داده گیری آن ها false باشد) را انتخاب کند و برای هر شهر شروع به گرفتن پست ها می کند.هر درخواست 24 پست اخر را به ما می دهد.

کلیه ی اطلاعات مورد نیاز ما از پاسخ درخواست در جدول car ذخیره می شود.هر پست توکن منحصر به فردی دارد که به کمک آن پست ها قابل جداسازی اند.در خروجی این درخواست url های تصاویر مربوط به پست نیز وجود دارد که در دیتابیس ذخیره شده است.

فرایند اصلی دریافت پست ها و دانلود عکس ها در تابع processGetPostAndDownloaddImage صورت می گیرد.این تابع اولین بار برای هر شهر با درخواست از اولین صفحه پست ها شروع می کند.سپس در طی فرایند تشخیص می دهد که آیا صفحه ی بعدی نیز وجود دارد یا خیر. درصورت وجود صفحه ی بعد به صورت تودرتو متد خود را فراخوانی می کند.هر زمان که تعداد کل عکس های دانلود شده مربوط به یک برند از 1700 عکس حداقلی بگذرد وضعیت داده گیری برند مربوطه به true تغییر می یابد تا فرایند ادامه پیدا نکند.

فرایند اصلی گرفتن پست ها و دانلود عکس ها با اجرای 3 متد به صورت وابسته به هم انجام می شود.این سه متد به ترتیب sendRequest ، processPostData و downloadImageCar هستند.در ابتدا درخواست مورد نیاز ایجاد و زده می شود.پاسخ درخواست شامل 24 پست به متد بعدی داده می شود.در متد processPostData تشخیص وجود صفحه ی بعدی،اماده سازی ابجکت اطلاعات هر پست و گرفتن url تمامی عکس های مربوط به هر پست انجام می گیرد.در شکل 6 این تابع نمایش داده شده است.

در متد نهایی downloadImageCar کلیه ی url های تصاویر دانلود نشده از دیتابیس دریافت و برای تک تک آن ها عکس دانلود و در فایل مربوط به آن ذخیره می شود.شکل 7و8 تصاویری از نمایش این تابع می باشد.



تصویر 5- نمای تابع processGetPostAndDownloadImage



تصویر 6- نمای تابع processPostData

اسکریپت autoRun :

با توجه به نیاز برای اجرای طولانی مدت الگوریتم داده گیری از یک اسکریپت جهت اجرای مجدد الگوریتم کمک گرفتیم. به کمک این اسکریپت در صورت توقف در عملیات داده گیری، اجرای آن دوباره از سر گرفته می شود.الگوریتم داده گیری به نحوی نوشته شده است که هر بار اجرا شود ادامه روند قبلی را طی کند.

به علت حجم بالای درخواست های مورد نیاز برای دریافت پست ها،url های تصاویر هر پست و درخواست دانلود هر url در مواردی ارتباط با سرور های سایت دیوار به خطا برخورد می نمود که در اینصورت اجرا متوقف می شد که در این صورت اسکریپت autoRun پس از 300 ثانیه اجرا را از سر می گیرد.



تصویر 7- نمای تابع downloadImageCar

یکی دیگر از چالش هایی به علت حجم بالای درخواست ها محدودیتی است که از سمت سرور های دیوار اعمال شده است و آن هم عدم اجازه درخواست بیشتر از 4 بار در ثانیه است.به همین دلیل در فرایند دانلود عکس ها برای برخورد نکردن با این خطا پس از هر درخواست 250 میلی ثانیه تاخیر ایجاد می شود.

یکی از دیگر از نکات ذخیره سازی و دریافت عکس ها ایجاد status برای هر url در لیست url عکس های هر پست است.دراین روش ممکن است یکی از url ها به خطا بخورد و وضعیت آن به false تغییر می یابد سپس در هر دور اجرا همه ی url هایی که وضعیت دانلود false دارند دریافت می شود و بنابراین تلاش مجدد برای دانلود عکس ها صورت می گیرد.



تصویر 8- نمای تابع downloadImageCar

مطابق شکل 8 مسیر دانلود تصاویر از روی url مربوط به هر تصویر ساخته می شود.در واقع هر پست با توکن منحصر به فرد خود قابل شناسایی است و تصاویر هر پست نیز بر اساس مقدار توکن شماره گذاری شده اند و در url دانلود آن ها قرار دارد که از این مسئله برای ذخیره در فولدرهای مربوط به هر مدل استفاده شده است و هر عکس به اسم توکن و شماره آن عکس در پست مربوطه می باشد.

## جمع بندی

پس از تکمیل الگوریتم داده گیری این کد توسط اسکریپت autoRun اجرا گشت.هدف از اجرای این کد دریافت عکس های 100 مدل ماشین ایرانی بود و از هر مدل 1700 عکس می بایست دریافت شود.در الگوریتم نوشته شده در صورت رسیدن به 1700 عکس فرایند دریافت برای مدل مربوطه متوقف می شود و در غیر این صورت برای تمامی شهر های موجود و همه ی پست های هر شهر گرفتن عکس ها ادامه می یابد.

پس از اجرای فاز اول داده گیری که حدود 1 هفته بدون توقف صورت گرفت برای 30 مدل بالای 1700 عکس جمع آوری گردید و سایر مدل ها تعداد کل عکس های گرفته شده به این تعداد نمی رسید.در خصوص مدل هایی که تعداد عکس های آن ها خیلی کم بود (زیر 500 عکس) راهکاری وجود نداشت اما برای سایر مدل هایی که می توانستند به 1700 عکس برسند ما الگوریتم را با اندکی تغییر مجدد اجرا نمودیم.

در فاز دوم اجرای الگوریتم هر مدلی که کمتر از 1700 عکس گرفته شده داشت انتخاب می شد و فرایند برای آن تکمیل می گردید.پس از اجرای 1 هفته ای این الگوریتم تعداد مدل های مطلوب به 45 مدل رسید.نکته ای که در مورد این مدل ها وجود داشت نرخ ثبت آگهی جدید در سایت دیوار بود که نیاز بود مدت زمان طولانی الگوریتم در اجرا باشد تا هر پست جدید ماشین دریافت شود.

با توجه به کمبود زمان برای اتمام پروژه و نبود سرور هایی که بتوان کد را برای مدت چند ماهه اجرا کرد و هم چنین اینکه در سایت دیوار مدت نگه داشتن اگهی ها تا 1 ماه بیشتر نیست، اجرای الگوریتم داده گیری ما تا رسیدن به این مرحله ادامه یافت.در انتها پوشه تصاویر حاوی 100 فولدی با نام ایدی برند ها به عنوان دیتاست جمع آوری شده برای استفاده در فاز پیش پردازش تکمیل گردید.

حجم نهایی دیتاست جمع اوری شده 18.5 گیگ می باشد.اکسل نهایی اطلاعات برند ها در پیوست قرار گرفته شده است.

# بخش دوم: پیش پردازش داده ها

## حذف عکس های نامعتبر

از آنجایی که دیتاست را از سایت دیوار جمع آوری کردیم و در حین جمع آوری داده ها به مشکلاتی در حین اجرای اسکریپت داده گیری برخوردیم لذا برخی از عکس ها به فرمت معتبر نیستند و امکان خواندن آنها نمی باشد. لذا در اولین مرحله پیش پردازش داده ها ما برای جلوگیری از مشکل در مراحل بعدی پروژه عکس های نامعتبر و غیرقابل استفاده را از دیتاست حذف میکنیم.

## 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.

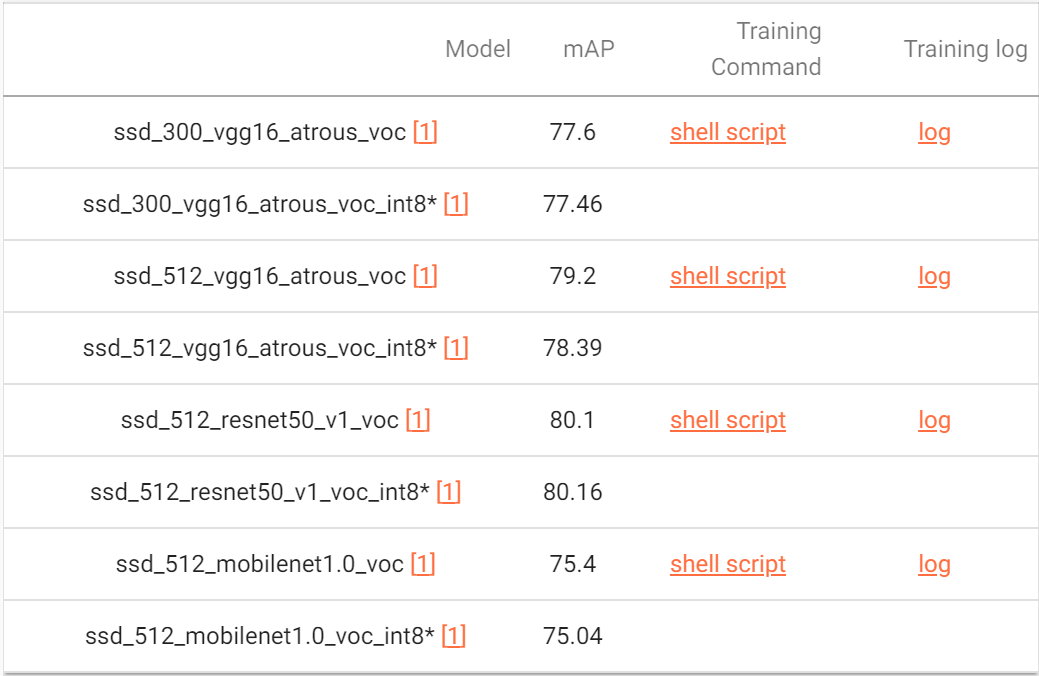
نکته ای که لازم است به آن اشاره کنیم این است که چون ما برای این بخش دیتاست کوچکی در نظر گرفتیم که تنها شامل 500 داده آموزش و 100 داده validation و 200 داده آزمون است بنابراین به روش transfer learning ابتدا وزن های imagenet برای این مدل ها را به عنوان وزن اولیه در نظر گرفتیم و سپس مدل را روی دیتاست خود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

## برچسب زدن عکس ها

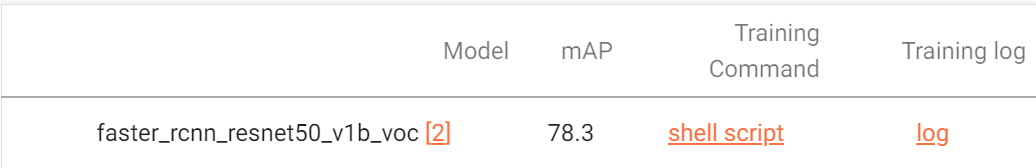
همانطور که میدانید ما حدود 100 هزار عکس از انواع ماشین ها را از سایت دیوار دریافت کرده ایم و حالا نیاز داریم تا لیبل آنها را مشخص کنیم. اگر پروژه تنها شامل طبقه بندی بود، لیبل گذاری بسیار راحت بود و تنها با گذاشتن عکس هر نوع ماشین در پوشه مربوطه، لیبل آن عکس مشخص میشد اما ما در این پروژه علاوه بر طبقه 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.

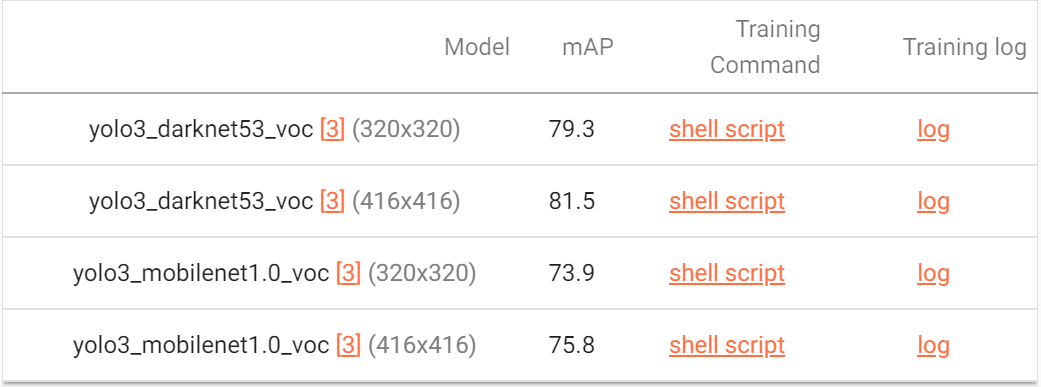
درنهایت پس از بررسی های مختلف به کتابخانه model\_zoo از ماژول gluoncv رسیدیم که این کتابخانه در بحث تشخیص اشیا شامل مدل های مختلف از پیش آموزش دیده روی دیتاست های COCO و Pascal VOC می باشد که در ادامه برخی از آنها را مشاهده میکنید:



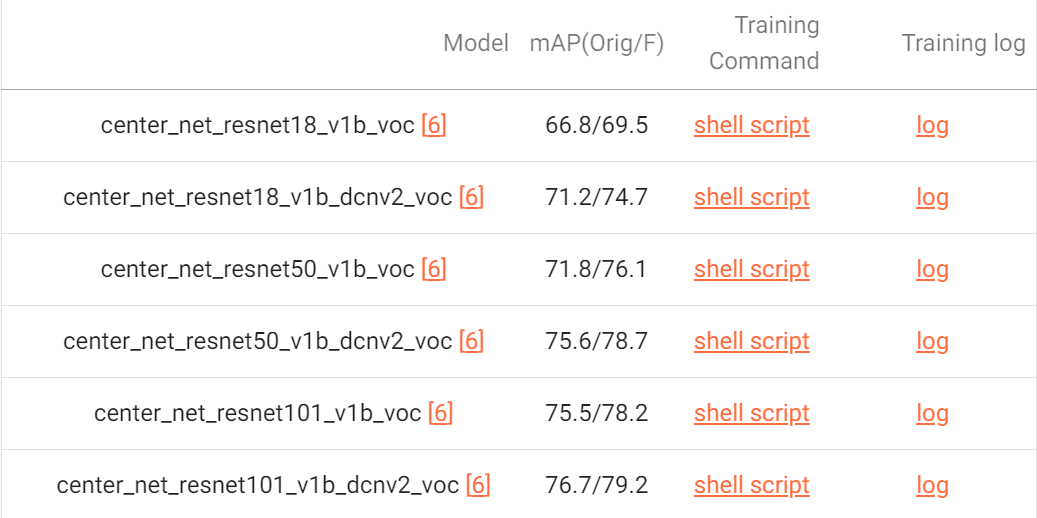
تصویر 9 – مدل های مختلف SSD از پیش آموزش دیده با دقت آن ها روی دیتاست Pascal VOC



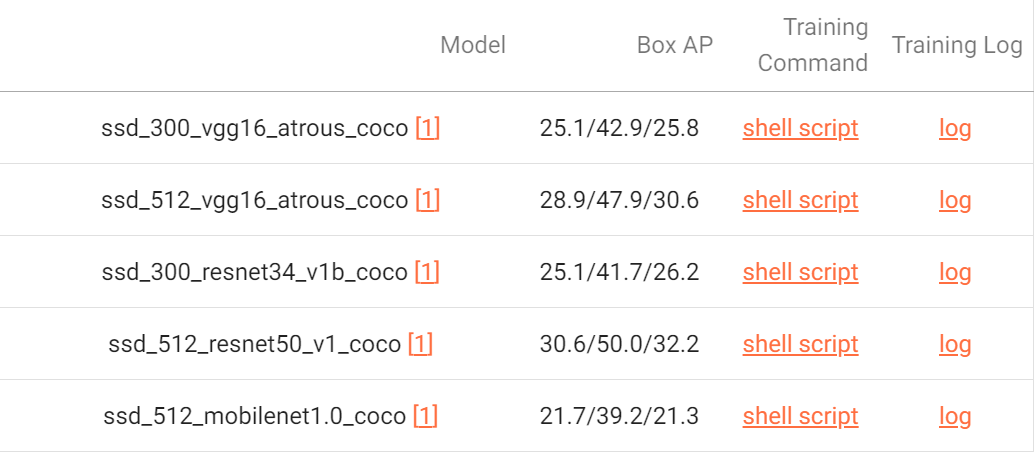
تصویر 10 - مدل های مختلف Faster RCNN از پیش آموزش دیده با دقت آن ها روی دیتاست Pascal VOC



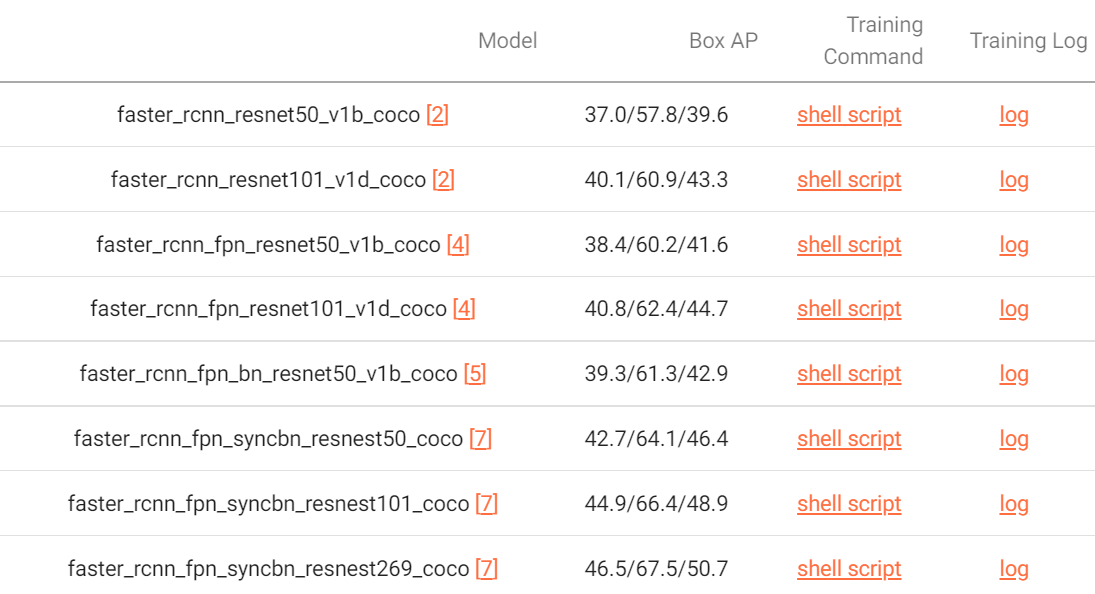
تصویر 11 - مدل های مختلف YOLO V3 از پیش آموزش دیده با دقت آن ها روی دیتاست Pascal VOC



تصویر 12 - مدل های مختلف CenterNet از پیش آموزش دیده با دقت آن ها روی دیتاست Pascal VOC



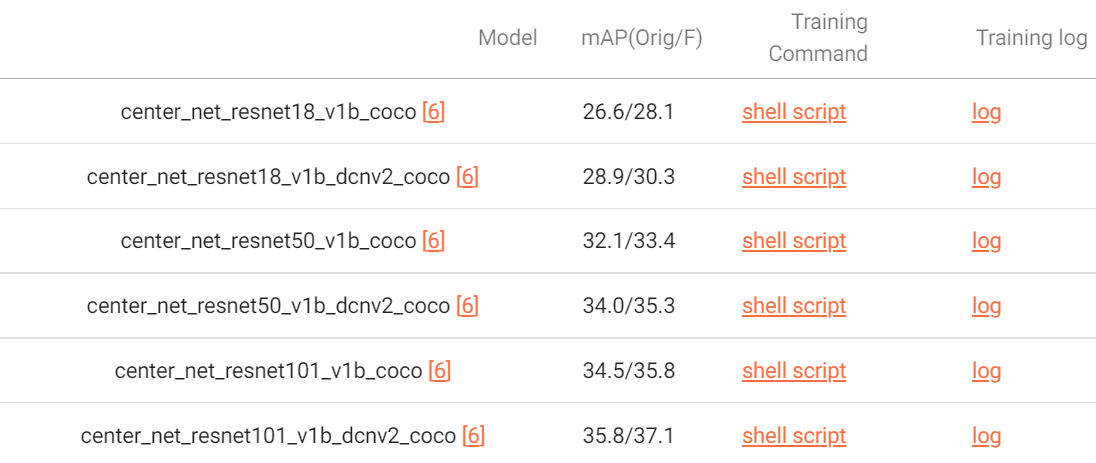
تصویر 13 - مدل های مختلف SSD از پیش آموزش دیده با دقت آن ها روی دیتاست Pascal VOC



تصویر 14 - مدل های مختلف Faster RCNN از پیش آموزش دیده با دقت آن ها روی دیتاست Pascal VOC



تصویر 15 - مدل های مختلف YOLO V3 از پیش آموزش دیده با دقت آن ها روی دیتاست Pascal VOC



تصویر 16 - 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 17- 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

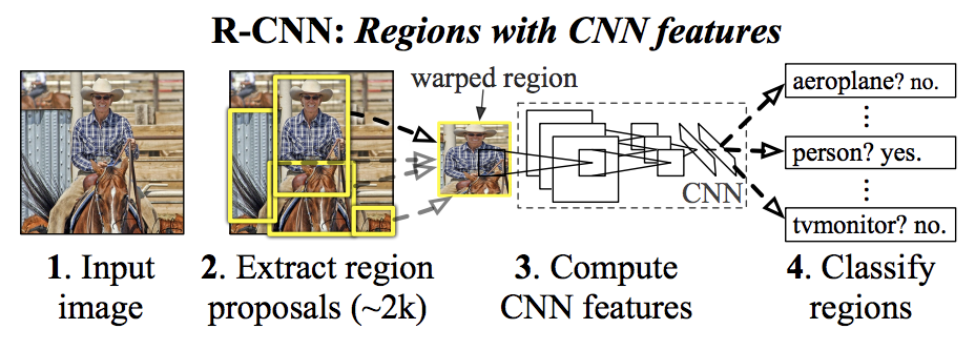
در زمان های قدیم و قبل از اینکه الگوریتم های عمیق ارائه شوند، از مدل هایی مثل HOG، HARR، SIFT، SURF و... برای بحث تشخیص اشیا استفاده میشد که امروز این مدل ها کاربرد زیادی ندارند اما همچنان ممکن است از آنها بسته به نیاز استفاده شود. این مدل ها از هیچ شبکه عمیقی استفاده نمیکنند، مثلا الگوریتم [[3]](#footnote-3) 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 18- استخراج ویژگی از تصاویر در روش HOG با استفاده از بردارها

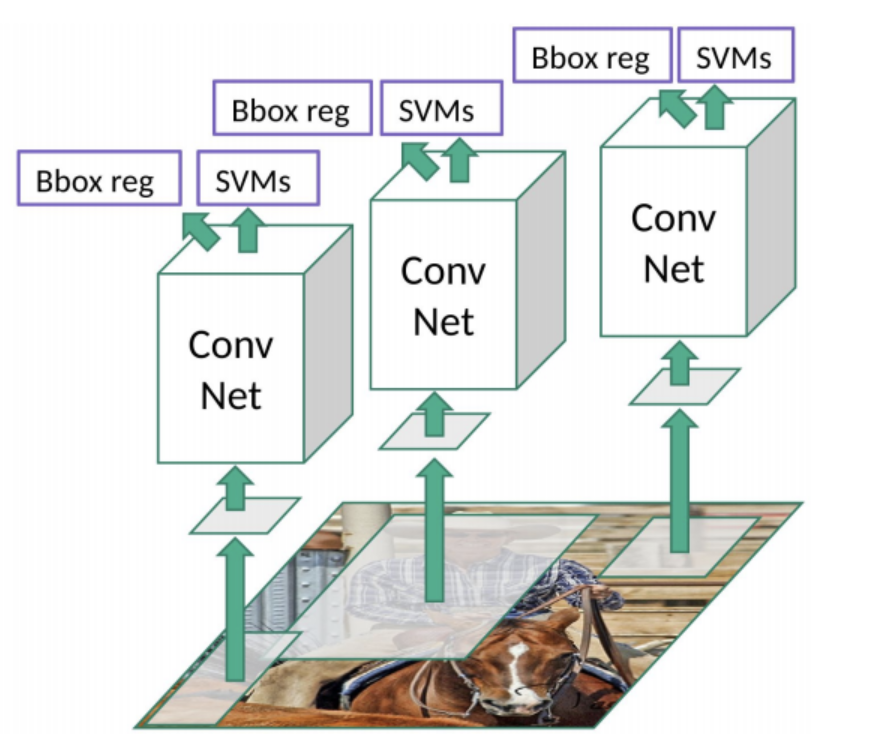
اما خب به دلیل بزرگ بودن مسئله ما، مدل های قدیمی پاسخگو نیستند و ما نیاز به مدل های شبکه عمیق داریم. برای همین سراغ مدل های [[4]](#footnote-4)RCNN و Fast RCNN و Faster RCNN و در نهایت مدل YOLO[[5]](#footnote-5) رفتیم.

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 19 – 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 20 - نحوه عملیات الگوریتم RCNN روی ناحیه های پیشنهادی

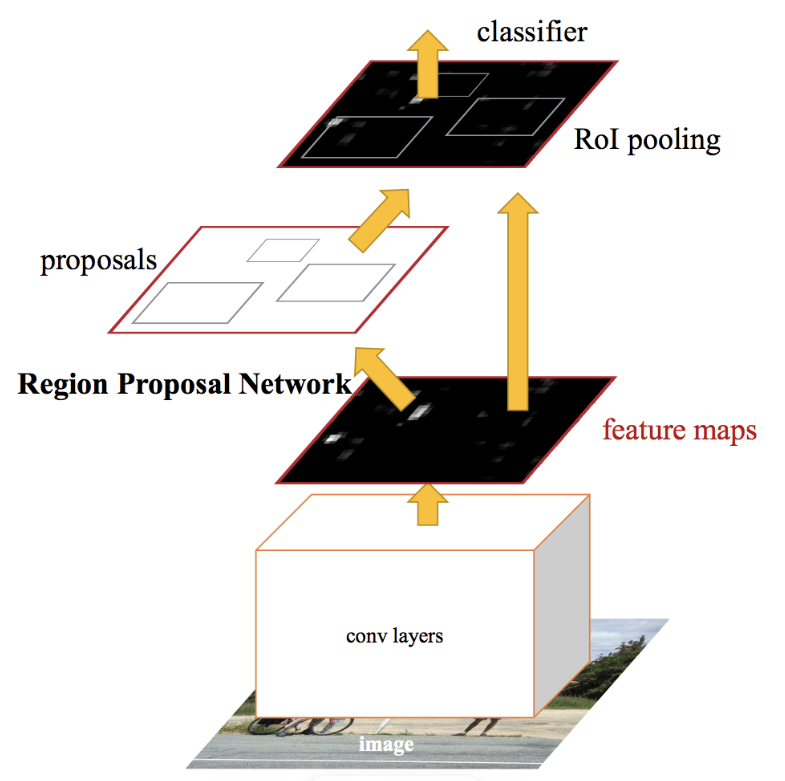
در fast RCNN رویکرد مشابه الگوریتم R-CNN است. اما به جای دادن تک تک ناحیه های پیشنهادی به CNN، تصویر ورودی را به CNN می‌دهیم تا یک نگاشت ویژگی های کانولوشنی ایجاد کنیم. از نگاشت ویژگی کانولوشنال، ناحیه های پیشنهادی را شناسایی کرده و آنها را به شکل مربع در می آوریم و با استفاده از یک لایه RoI Pooling 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 21 - 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 22- 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.

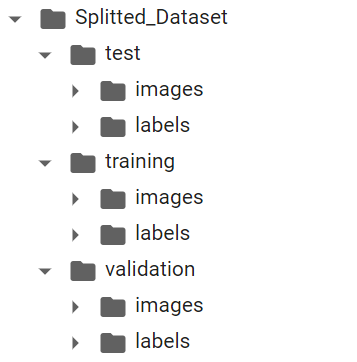
« با اجرای Tesla P100، زمان استنتاج تا 0.007 ثانیه در هر تصویر، یعنی 140 فریم در ثانیه (FPS) را مشاهده کردیم! در مقابل، YOLO v4 پس از تبدیل شدن به همان کتابخانه Ultralytics PyTorch به 50 FPS دست یافت.»

« 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.

## نحوه استفاده از مدل YOLOv5

در ابتدا باید دیتاست را به 3 بخش train و valid و test تقسیم کنیم. که نحوه تقسیم بندی به صورت زیر می باشد:



تصویر 23- نحوه پوشه بندی مناسب دیتاست برای آموزش مدل yolov5

تقسیم بندی ما برای train و test و valid به صورت 70، 20 و 10 درصد می باشد. یعنی 70 درصد از عکس های هر کلاس مربوط به train، 20 درصد برای valid و باقی داده ها برای test می باشد.

سپس یک فایل dataset.yaml می سازیم که در آن مسیر پوشه train، val و test و تعداد کلاس های داده ها و همچنین آرایه ی برچسب های کلاس ها ، نام خودرو ها، به ترتیب شماره کلاس را قرار میدهیم. پس از اینکار لازم است تا ریپازیتوری yolov5 را از گیت هاب clone کنیم.

یکی دیگر از مزایای yolov5 این است که میتوان گزارش نهایی آموزش مدل و نمودار های آن را در سایتی به نام wandb مشاهده کرد. برای اینکار ابتدا لازم است که داخل این سایت اکانت ایجاد کنیم و یک پروژه در آن ایجاد کنیم و پس از آن در محیط آموزش مدل، که ما مدل را در محیط کولب آموزش دادیم، به اکانت wandb وارد میشویم و پس از اتمام آموزش نتایج داخل سایت نمایش داده میشود که در بخش ارزیابی مدل نتایج را نمایش خواهیم داد.

در نهایت پس از ورود به اکانت wandb با استفاده از دستور زیر مدل را آموزش میدهیم:

!python train.py --img 256 --batch-size 16 --epochs 45 --data <PATH\_TO\_DATASET.YAML\_FILE> --cfg /content/yolov5/models/yolov5s.yaml --weights '' --name yolov5s\_results  --cache

هر یک از پارامتر های بالا را در ادامه توضیح میدهیم:

--img اندازه عکس های ورودی به مدل را مشخص میکند که معمولا این سایز را 416 در نظر میگیرند اما ما به دلایلی که در ادامه توضیح خواهیم داد اندازه عکس های ورودی را 256 در نظر گرفتیم.

--batch-size اندازه batch ورودی به مدل را مشخص میکند که ما آن را 16 در نظر گرفتیم.

--epochs تعداد ایپاک ها برای آموزش مدل را مشخص میکند که برای آموزش مدل ما 45 ایپاک برای دقت مناسب کافی است.

--cfg کانفیگ و ساختار مدل را مشخص میکند که ما از مدل yolov5s استفاده کردیم و کانفیگ این مدل در فایل yolov5s.yaml قرار دارد که مسیر این فایل را برای این پارامتر مشخص میکنیم.

--weights وزن های اولیه برای آموزش مدل را مشخص میکند که میتوانیم از وزن های pretrained روی دیتاست های Coco و... استفاده کنیم اما ما در اینجا وزن اولیه مشخص نکردیم و این به این معناست که وزن های اولیه مقادیر رندوم میباشند.

--name نامی است که برای پروژه در نظر میگیریم که ما نام yolov5s\_results در نظر گرفتیم.

--cache عکس ها و برچسب های آموزش و validation را ابتدا در رم cache میکند و سپس مدل را آموزش میدهد. این کار باعث میشود سرعت آموزش مدل به نسبت حالت بدون cache چندین برابر شود.

## چالش های آموزش مدل

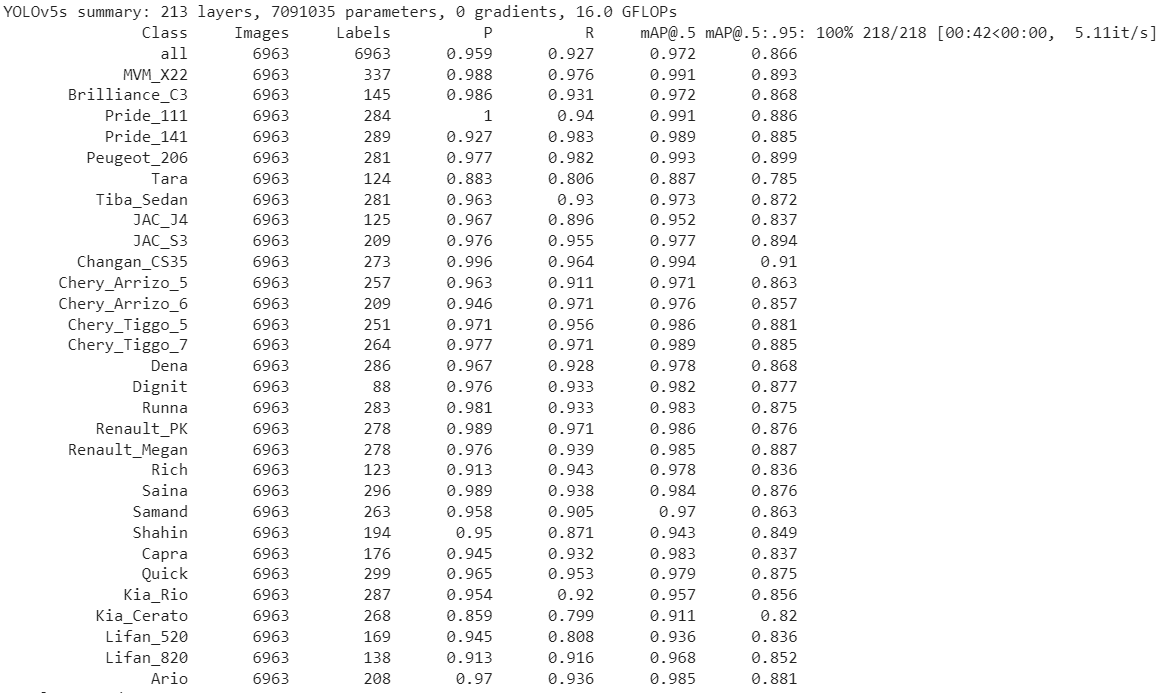
بزرگترین چالش ما برای آموزش مدل کمبود منابع بود. از آنجایی که هزینه سرور ها به شدت بالا بود و gpu کولب نیز محدودیت زمانی برای استفاده داشت و همچنین کمبود رم کولب تصمیم گرفتیم تا یک سوم کلاس های ماشین ها را آموزش دهیم.

در کولب میزان رم 12 گیگ می باشد و از آنجایی که ما میخواستیم داده ها را cache کنیم تا آموزش مدل سریع و در مدت مشخص کولب انجام شود لذا تعداد کلاس ها را 30 و اندازه عکس ورودی را 256 در نظر گرفتیم. اگر اندازه عکس ها و تعداد کلاس ها از این بیشتر در نظر میگرفتیم، از آنجایی که باید تمام داده های آموزش و valid و نه بخشی از آن ها cache شود، حجم رم کولب جوابگوی cache داده ها نبود و به همین دلیل قبل از شروع epoch ها آموزش مدل متوقف میشد. از طرفی چون اندازه خودرو ها در عکس های دیتاست ما معمولا بزرگ است به همین دلیل اگر اندازه عکس را از 416 به 256 تغییر دهیم همچنان مدل ماشین در عکس مشخص است و ما داده زیادی را از دست نداده ایم و در عوض میتوانیم با همین محدودیت هایی که وجود دارد مدل را آموزش دهیم.

پس از آموزش مدل نیز وزن مدل آموزش دیده داخل پوشه runs/train/yolov5s\_results/weights ذخیره شده که بهترین وزن ها در فایل best.pt و آخرین وزن آموزش داده شده در فایل last.pt ذخیره شده است که ما برای پیش بینی از وزن best.pt استفاده میکنیم.

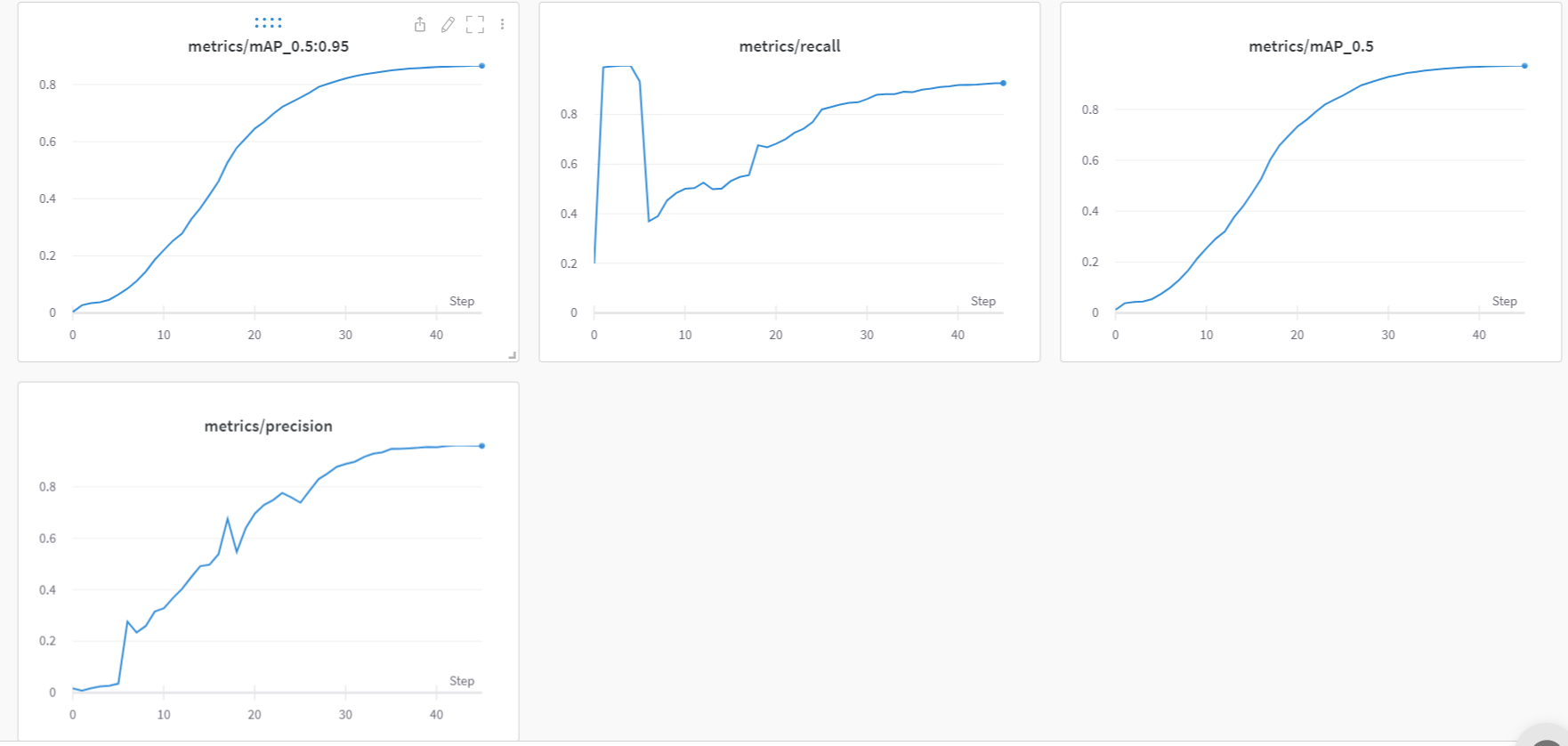
# بخش چهارم: ارزیابی و نتایج مدل نهایی

ما مدل yolov5s را روی 30 مدل ماشین زیر آموزش دادیم و با 45 epoch به میانگین دقت 86 درصد روی داده های valid رسیدیم که دقت مناسبی است.



تصویر 24- میزان دقت و فراخوانی و میانگین دقت متوسط روی داده های آموزش و validation

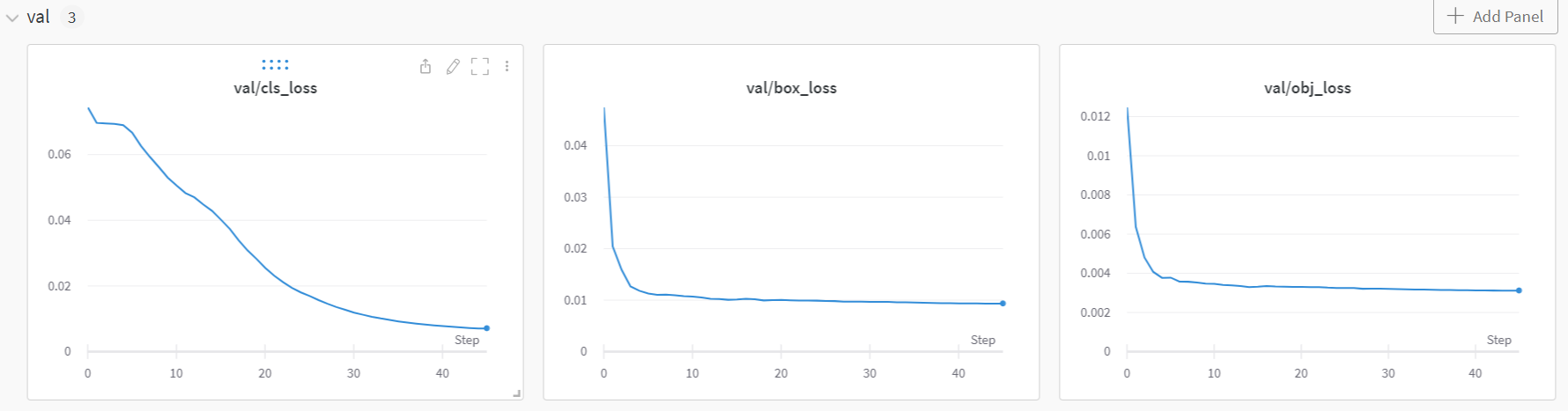
گفتیم نتایج آموزش مدل در سایت wandb ذخیره میشود که به صورت زیر می باشد:



نمودار 1- نمودر دقت و فراخوانی و میانگین دقت متوسط در حین آموزش مدل و بر حسب تعداد epoch ها



نمودار 2- نمودارمیزان loss مربوط به box وکلاس ها و اشیاءروی داده های آموزش



نمودار 3- نمودارمیزان loss مربوط به box وکلاس ها و اشیاءروی داده های valid

همچنین بخشی از لیبل گذاری تصاویر validation به صورت زیر می باشد:



تصویر 25- لیبل گذاری مدل روی تعدادی از داده های valid

# بهبود های بعدی

در ورژن های بعدی از مدل قرار است موارد زیر را در نظر داشته باشیم:

1. بحث data augmentation تا برای کلاس های دارای عکس های کمتر نیز بتوانیم مدل را آموزش دهیم بدون آنکه از تعمیم پذیری مدل کم شود.
2. به محض دسترسی به منابع کافی بتوانیم تمام کلاس های داده که جمع آوری کرده ایم را آموزش دهیم.
3. یک اپلیکیشن اندروید برای تست کردن مدل ایجاد کنیم تا عملکرد مدل روی دستگاه های با منابع محدودتر را نیز بسنجیم.
4. از ورژن های بعدی YOLO یعنی v6 و v7 بتوانیم برای مسئله خود استفاده کنیم و کارایی آن مدل ها را در مقابل مدل کنونی خود ارزیابی 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)