به نام خدا

گزارش پروژه نهایی درس یادگیری عمیق

تشخیص اشیا و عمق آن‌ها در تصویر

نام اعضای گروه:

علیرضا حشمتی ۹۹۲۰۶۰۰۹

پیمان ناصری ۹۶۱۰۰۵۲۲

استاد:

جناب آقای دکتر فاطمی زاده

بهمن ۱۴۰۰

تشخیص اشیا با شبکه **YOLO**

الگوریتم YOLO در سال ۲۰۱۶ در کنفرانس CVPR ارائه شد. این مقاله، ساختار جدیدی را برای سیستم‌های تشخیص اشیا ارائه داد و به‌همین دلیل بسیار مورد توجه قرار گرفت. ما در پروژه از شبکه pretrain شده آن استفاده کرده ایم.

مقدمه

انسان با نگاهی کوتاه به تصویر بلافاصله می‌فهمد چه اشیایی در تصویر وجود دارند، موقعیت‌شان در تصویر کجاست و حتی چه ارتباطی با هم دارند. این عمل‌ها برای انسان بسیار ساده است و سیستم بینایی دقیق و سریع انسان کارهای به‌مراتب پیچیده‌تری مانند رانندگی را می‌تواند به‌آسانی انجام دهد. البته، بخش مهمی از رانندگی، شناسایی و موقعیت‌یابی اشیای اطراف خودرو هست که انسان در این زمینه مهارت بالایی دارد. حال، اگر الگوریتم‌های سریع و دقیقی برای شناسایی و موقعیت‌یابی اشیا داشته باشیم، می‌توان امیدوار بود که ماشین‌های خودرانِ بدون نیاز به سنسورهای مخصوص داشته باشیم. شناسایی و موقعیت‌یابی اشیا از جمله زمینه‌های تحقیقاتی قدیمی و مهم در بینایی کامپیوتر است. در بینایی کامپیوتر، به شناسایی و موقعیت‌یابی اشیا در تصویر Object Detection (تشخیص اشیا) گفته می‌شود. YOLO یکی از سیستم های تشخیص اشیا است. تصویری از خروجی یکی از سیستم تشخیص اشیا را در زیر مشاهده می‌کنید.

تشخیص اشیا = شناسایی اشیا + موقعیت‌یابی اشیا

شکل ۱: نمونه تصویر خروجی یک سیستم تشخیص اشیا؛ موقعیت کادرها نشان‌دهنده بخش موقعیت‌یابی و نام اشیا هم نشان‌دهنده بخش شناسایی است.

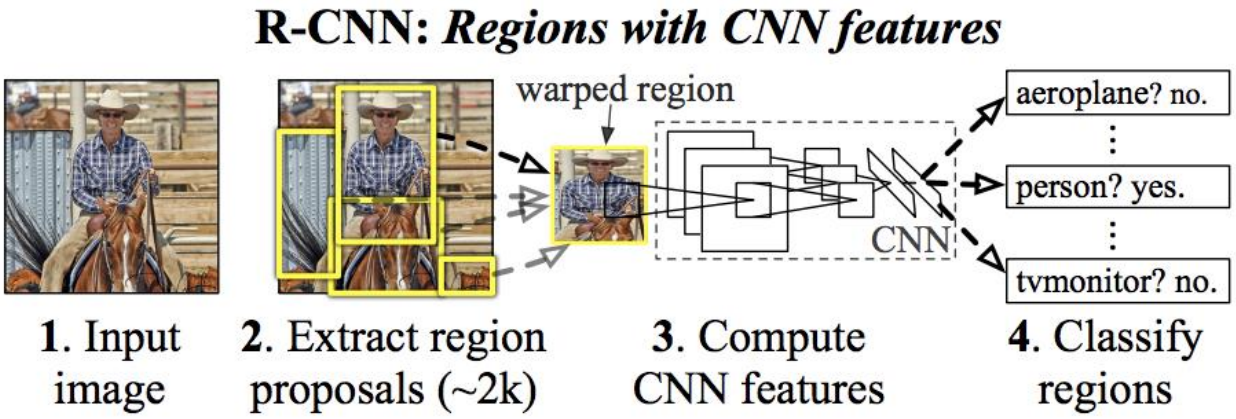
YOLO مخفف عبارت You Only Look Once، به معنای "شما فقط یک‌بار به تصویر نگاه می‌کنید" هست. در واقع، این عبارت به همان قابلیت سیستم بینایی انسان اشاره دارد که با یک نگاه عمل تشخیص اشیا را انجام می‌دهد. بنابراین، سیستم تشخیص اشیای YOLO با هدف ارائه روشی مشابه کارکرد سیستم بینایی انسان طراحی شده است. اما سوال اینجاست که سیستم‌های تشخیص اشیای قبل از YOLO چه ویژگی‌هایی داشتند و چگونه کار می‌کردند؟ یعنی آنها شباهتی به سیستم بینایی انسان نداشتند؟

الگوریتم‌های پیش از **YOLO**

سیستم‌های تشخیص اشیای پیش از YOLO، از کلاسیفایرها در کار تشخیص اشیا استفاده می‌کردند. این سیستم‌ها برای تشخیص یک شی، یک کلاسیفایر را در موقعیت‌ها و مقیاس‌های مختلف به تصویر ورودی اعمال می‌کردند. به‌عنوان مثال، سیستم‌هایی مانند Deformable Part Models یا DPM از پنجره‌های لغزان (Sliding Window) بهره می‌برند که کلاسیفایر را به موقعیت‌های مختلف در سراسر تصویر اعمال می‌کنند. این اِعمال کلاسیفایر به موقعیت‌های مختلف تصویر، کار زمان‌بری است که البته شباهت چندانی هم به سیستم بینایی انسان در تشخیص اشیا ندارد. در شکل زیر، نمونه‌ای از الگوریتم‌های مبتنی بر پنجره لغزان را مشاهده می‌نمایید.

شکل ۲: الگوریتم تشخیص اشیا مبتنی بر پنجره لغزان [لینک مرجع]

دسته دیگری از رهیافت‌ها که نسبت به DPM جدیدتر هستند، رهیافت‌های مبتنی بر پروپوزال ناحیه (Region Proposal) مانند R-CNN است. در شکل ۳، ساختار یک الگوریتم مبتنی بر پروپوزال ناحیه به‌نام R-CNN را مشاهده می‌نمایید. در این روش‌ها، ابتدا مجموعه زیادی پروپوزال یا همان باکس برای هر تصویر تولید می‌شوند (مثلا ۲۰۰۰ پروپوزال برای هر تصویر در مرحله ۲ شکل ۳). سپس، هریک از پروپوزال‌ها به یک سایز مشخص ریسایز می‌شوند و برای استخراج ویژگی در اختیار شبکه‌های CNN قرار می‌گیرند (مرحله ۳ در شکل ۳). در نهایت، یک کلاسیفایر برای کلاسیفای‌کردن این باکس‌های تولیدشده به‌کار برده می‌شود (مرحله ۴ در شکل ۳). بنابراین، به‌همین دلیل است که گفتیم روش‌های تشخیص اشیای پیش از YOLO عمل تشخیص اشیا را با کلاسیفایرها انجام می‌دهند. این مسیر نسبتا پیچیده سرعت پایینی دارد و بهینه‌سازی آن مشکل است، چون هریک از این اجزا که در شکل ۲ مشاهده می‌کنید، باید به‌صورت جداگانه آموزش داده شوند.

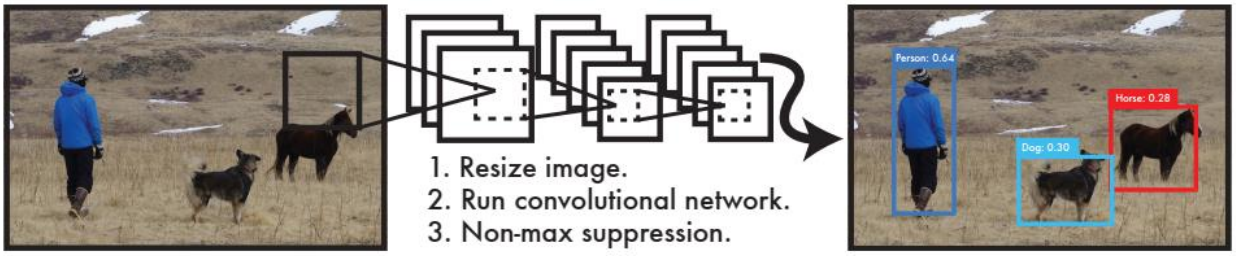


شکل ۳: ساختار کلی الگوریتم تشخیص اشیا *R-CNN*[لینک مرجع]

الگوریتم **YOLO**

الگوریتم YOLO معماری سیستم‌های تشخیص اشیا را دست‌خوش تغییراتی کرده است و به مساله تشخیص اشیا به‌صورت یک مساله رگرسیون می‌نگرد که مستقیم از پیکسل‌های تصویر به مختصات باکس و احتمال کلاس‌ها می‌رسد. با استفاده از سیستم YOLO، برای تشخیص اشیای موجود در تصویر، به هر تصویر شما فقط یک بار می‌نگرید (You Only Look Once).

YOLO بسیار ساده است (به شکل ۴ نگاه کنید). تنها یک شبکه کانولوشنی وجود دارد که تصویر ریسایز ورودی را دریافت (مرحله ۱) و سپس به صورت همزمان چندین باکس را به همراه احتمال کلاس‌ها پیش‌بینی می‌کند (مرحله ۲). YOLO روی تصاویر کامل آموزش می‌بیند و مستقیما کارآیی تشخیص را بهبود می‌دهد.



شکل ۴: ساختار کلی الگوریتم *YOLO* [لینک مرجع]

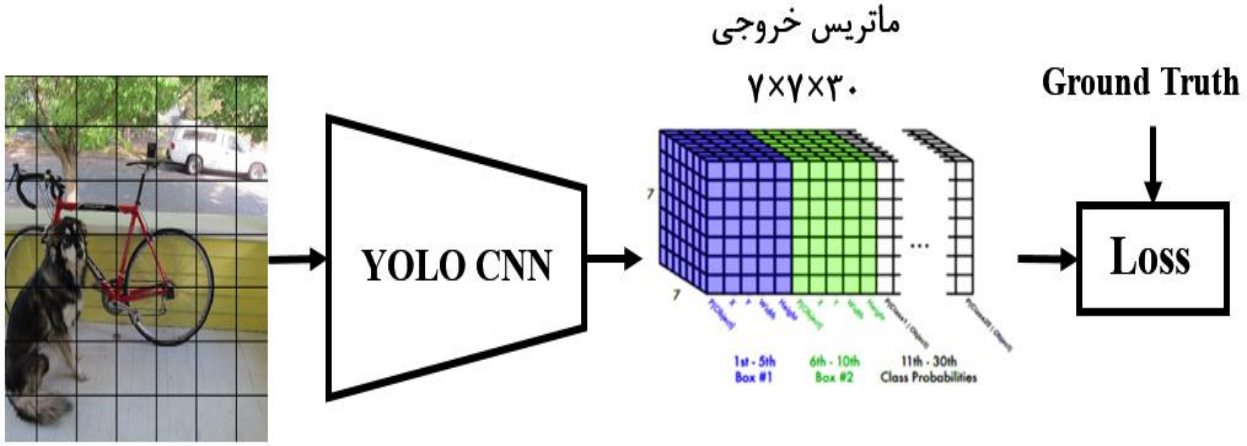
مدل یکپارچه YOLO مزایای زیادی نسبت به روش‌های سنتی تشخیص اشیا دارد که در ادامه اشاره خواهد شد:

- اول، YOLO بسیار سریع است. در اینجا، تنها یک شبکه وجود دارد که خیلی ساده به آن ورودی تصویر داده می‌شود تا شبکه پیش‌بینی‌های تشخیص اشیا را به ما نشان دهد. دو نسخه شبکه YOLO شامل YOLO اصلی و YOLO سریع طراحی شده است. YOLO اصلی با کارت گرافیک Titan X با سرعت ۴۵ فریم‌برثانیه اجرا می‌شود. نسخه سریع YOLO هم سرعتی بیش از ۱۵۰ فریم‌برثانیه دارد. یعنی YOLO می‌تواند در یک ویدئوی ۴۰ فریم‌برثانیه در حالت بلادرنگ به تشخیص اشیا بپردازد. YOLO نسبت به دیگر سیستم‌های تشخیص اشیای بلادرنگ، به mAP یا همان mean Average Precision دوبرابر دست یافته است. دقت کنید، عملکرد بهتر نسبت به سایر سیستم‌های بلادرنگ و نه سیستم‌های تشخیص اشیای قدرتمند مانند Faster R-CNN که بلادرنگ نیستند.

- دوم، YOLO برای پیش‌بینی تشخیص، به صورت کلی (Global) به تصویر نگاه می‌کند. برخلاف تکنیک‌های پنجره‌های لغزان (اسلاید) و پروپوزال، YOLO به کل تصویر نگاه می‌کند.
- سوم، YOLO تعمیم‌پذیری بالایی دارد. زمانی‌که تصاویر به شبکه آموزش داده می‌شوند و سپس شبکه آموزش‌دیده روی کارهای هنری تست می‌شود (در واقع منظورمان همان تغییر حوزه داده‌های ورودی است) شبکه YOLO با فاصله زیادی بهتر از شبکه‌هایی مانند DPM و R-CNN کار می‌کند. بنابراین، YOLO به شدت تعمیم‌پذیر هست و در مقابل حوزه‌های جدید و یا داده‌های ورودی غیرمنتظره با احتمال کمتری نسبت به بقیه سیستم‌ها با شکست مواجه می‌شود.

ساختار کلی الگوریتم **YOLO**

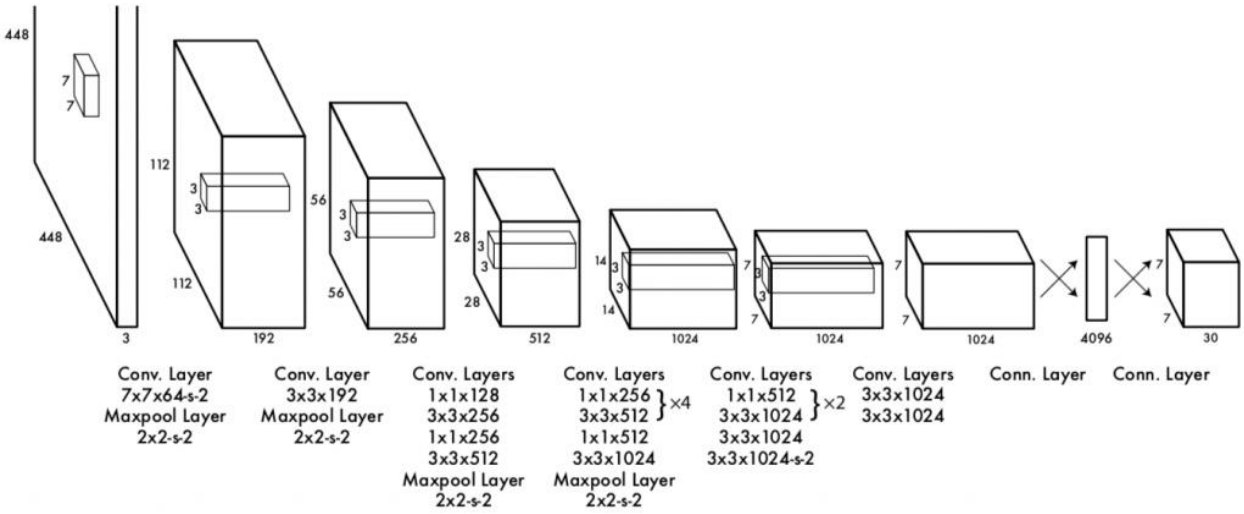
ساختار کلی الگوریتم YOLO در شکل ۵ نشان داده شده است. تصویر ورودی با ابعاد ۳×۴۴۸×۴۴۸ به یک Grid یا شبکه S×S تقسیم‌بندی می‌شود. این تصویر به شبکه YOLO داده می‌شود. خروجی شبکه کانولوشنی، ماتریسی به ابعاد S×S×۳۰ خواهد بود. هریک از درایه‌های ماتریس S×S خروجی معادل با یک سلول در شبکه S×S ورودی است (به ورودی و خروجی در شکل ۵ دقت کنید). خروجی S×S×۳۰ شامل مختصات باکس‌ها و احتمال‌هاست. اگر در فرآیند آموزش (Train) باشیم، خروجی S×S×۳۰ به‌همراه باکس‌های واقعی یا هدف (Ground Truth) به تابع اتلاف داده می‌شود. مقدار S در یولو نسخه ۱، برابر با ۷ درنظر گرفته شده است. اگر در فرآیند آزمایش (Test) باشیم، خروجی S×S×۳۰ به الگوریتم حذف غیر حداکثرها (Non-maximum Suppression) داده می‌شود تا باکس‌های ضعیف از بین بروند و تنها باکس‌های درست در خروجی نمایش داده شوند.



شکل ۵: ساختار کلی الگوریتم *YOLO* همراه با ورودی و خروجی

شبکه **YOLO**

YOLO شامل یک شبکه عصبی کانولوشنی (Convolutional Neural Network) با ۲۴ لایه کانولوشنی برای استخراج ویژگی و همچنین ۲ لایه فولی‌کانکتد (Fully Connected) برای پیش‌بینی احتمال و مختصات اشیا است. معماری شبکه YOLO را در شکل ۶ مشاهده می‌کنید.



شکل ۶: معماری شبکه *YOLO* همراه با ۲۴ لایه کانولوشنی [لینک مرجع]

همچنین، یک نسخه سریع از YOLO برای جابجایی مرزهای تشخیص اشیای سریع طراحی شده است. YOLO سریع، یک شبکه عصبی با تعداد لایه‌های کانولوشنی کمتر است که در آن از ۹ لایه کانولوشنی بجای ۲۴ لایه کانولوشنی (YOLO اصلی) استفاده شده و البته تعداد فیلترهای هر لایه در YOLO سریع نسبت به YOLO اصلی کمتر است. اندازه ورودی هر دو شبکه ۳×۴۴۸×۴۴۸ و خروجی شبکه نیز یک تنسور ۳۰×۷×۷ از پیش‌بینی‌ها است. درتمامی لایه‌ها از Leaky ReLU استفاده شده است.

آموزش شبکه **YOLO**

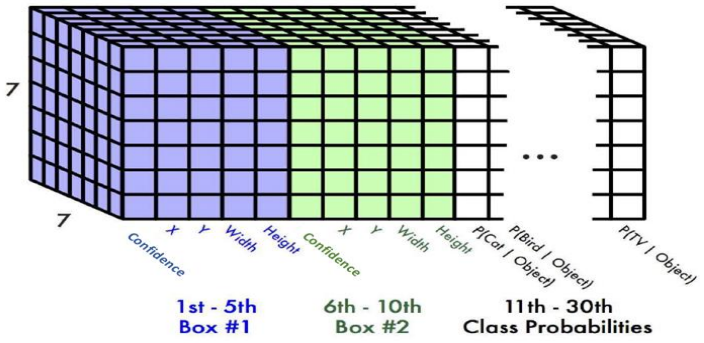
آموزش شبکه تشخیص اشیای YOLO در دو مرحله مجزا انجام می‌شود:

ابتدا، شبکه YOLO با پایگاه داده ۱۰۰۰ کلاسه ImageNet برای عمل کلاس‌بندی آموزش داده شده است. در این فرآیند آموزش، از ۲۰ لایه کانولوشنی ابتدایی YOLO استفاده شده است. در انتهای این ۲۰ لایه، یک لایه پولینگ میانگین (Average Pooling) و یک لایه فولی‌کانکتد قرار گرفته است. تصاویر ورودی در

اندازه ۳×۲۲۴×۲۲۴ به شبکه داده شده‌اند. این شبکه تقریبا به‌مدت یک هفته آموزش داده شده که در نهایت دقت ۸۸٪ در top-۵ در ImageNet حاصل شده است.

در مرحله دوم، برای کار تشخیص اشیا در ساختار مدل تغییراتی ایجاد شده است. تغییرات به این‌صورت است که چهار لایه کانولوشنی و دو لایه فولی کانکتد با وزن‌های تصادفی به انتهای ۲۰ لایه شبکه اضافه شده است. در کار تشخیص اشیا اغلب به اطلاعات با جزئیات بیشتری نیاز است، به‌همین دلیل رزولوشن ورودی شبکه از ۳×۲۴۴×۲۲۴ به ۳×۴۴۸×۴۴۸ افزایش داده شده است. هدف از افزایش اندازه ورودی، بهره‌گیری از جزئیات بیشتر در تصویر است.

اندازه خروجی شبکه ۳۰×۷×۷ است. ابتدا از اندازه ۷×۷ شروع کنیم؛ تصاویر ورودی به یک شبکه ۷×۷ تقسیم‌بندی می‌شوند (در شکل ۵ نشان داده شده است). بنابراین، خروجی ۷×۷ متناظر با تصویر شبکه‌شده ورودی است. هر درایه در ۷×۷ خروجی، متناظر با یک سلول در تصویر شبکه‌شده ورودی است (شکل ۵). هر درایه از این ماتریس ۷×۷ خروجی، یک بردار به‌طول ۳۰ دارد (شکل ۷). این بردار به‌طول ۳۰ شامل اطلاعات پیش‌بینی احتمال‌ها و مختصات باکس است. اما چگونه؟ هر سلول از این آرایه ۷×۷ دو باکس می‌تواند رسم کند. برای رسم هر باکس به ۵ پارامتر (x,y,w,h,confidence) نیاز است. پارامترهای x و y، مختصات سطر و ستون مبدا باکس (مرکز باکس) را نشان می‌دهند. مختصات w و h به‌ترتیب متناظر با پهنا و ارتفاع باکس هستند. با این چهار پارامتر می‌توانیم باکس را ترسیم کنیم، درحالی‌که گفتیم ۵ پارامتر برای ترسیم باکس نیاز است. پارامتر پنجم چه کاربردی دارد؟ پارامتر پنجم confidence هست؛ یک پارامتر احتمالاتی با مقدار بین ۰ تا ۱ که می‌گوید اصلا این باکس شامل شی هست یا اینکه پس‌زمینه تصویر است! طبیعتا ما باکس‌هایی را می‌خواهیم که مقدار confidence بزرگی داشته باشند که نشان می‌دهد این باکس شامل یک شی است. مقدار confidence از طریق رابطه IoU بین باکس پیش‌بینی و باکس واقعی محاسبه می‌شود.



شکل ۷: تنسور سه‌بعدی خروجی *YOLO* که اعداد ۷ نشان‌دهنده سطر و ستون هست. همچنین، هر درایه در ۷×۷ یک بردار به‌طول ۳۰ دارد.

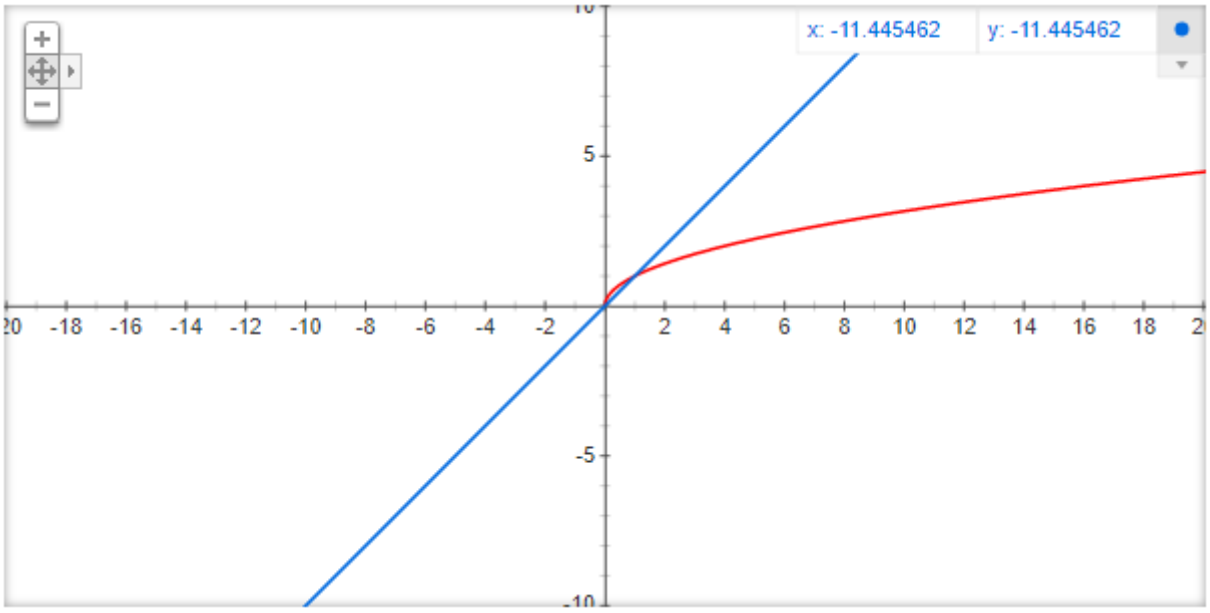
تابع اتلاف در الگوریتم **YOLO**

در YOLO از تابع اتلاف MSE یا Mean Squared Error استفاده شده است، چون بهینه‌سازی این تابع اتلاف آسان است و با مساله رگرسیون که در YOLO مطرح شده سازگار است. الگوریتم YOLO به مساله تشخیص اشیا به‌صورت رگرسیون می‌نگرد. تابع اتلاف MSE نشان‌دهنده دلیل رگرسیون هست. اما لازم است در تابع اتلاف MSE تغییراتی ایجاد شود که بیشتر با خواسته ما برای تشخیص اشیا هم‌راستا باشد. تابع اتلاف نهایی در YOLO به‌شکل زیر است:

$$
\begin{aligned}
&\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \qquad (3)
\end{aligned}
$$

در خط اول، با استفاده از رابطه SSE، موقعیت مبداهای دو باکس پیش‌بینی و واقعی (x,y) باهم مقایسه شده‌اند. اندیس‌های i و j به‌ترتیب نشان‌دهنده سلول‌ها (۴۹ سلول داریم) و باکس‌ها (B) هستند. پشت سیگما یک متغیر ۱obj لحاظ شده است؛ در صورتی ۱ هست که باکس j در سلول i شامل یک شی باشد، درغیراینصورت صفر خواهد بود. دو سیگما داریم که وظیفه‌شان بررسی تک‌تک سلول‌ها و باکس‌ها هست. پشت سیگماها هایپرپارامتر λ قرار دارد.

در خط دوم، رابطه تقریبا مشابهی با خط اول می‌بینیم. اما بجای x و y از w و h استفاده شده است. یعنی در اینجا می‌خواهیم پهنا و ارتفاع باکس پیش‌بینی را با باکس واقعی مقایسه کنیم. w و h داخل یک √ قرار دارند چون در تصویر، اشیای با اندازه‌های مختلف از خیلی کوچک تا خیلی بزرگ داریم. حالا وقتی‌که بخواهیم باکس‌های این اشیا را با باکس‌های واقعی مقایسه کنیم، همه باکس‌ها با هر اندازه‌ای را با یک معیار مقایسه می‌کنیم. درحالی‌که می‌دانیم خطا در باکس‌های بزرگ مانند خطا در باکس‌های کوچک نیست. به‌عبارت دیگر، یک پیکسل خطا در باکس بزرگ باید کمتر مجازات داشته باشد تا یک پیکسل خطا در باکس کوچک. با استفاده از √، ما باکس‌های بزرگ را کمتر از باکس‌های کوچک مجازات می‌کنیم. کافی‌است نمودار y=x و y=√x را در شکل زیر با هم مقایسه کنید.



شکل ۸: مقایسه دو خط $y=x$ و $y=\sqrt{x}$؛ تنبیه کمتر در مقادیر بزرگ در $y=\sqrt{x}$.

خط سوم و چهارم، ضریب اطمینان (Confidence) برای حضور یا عدم حضور یک شی در باکس هست. اول اینکه، خط سوم برای ضرایب اطمینان باکس‌هایی است که شامل شی هستند و خط چهارم متناظر با باکس‌هایی است که شامل هیچ‌گونه شی نیستند. پشت سیگماهای خط چهارم، یک هایپرپارامتر λ قرار داده شده است. مقدار این پارامتر ۰,۵ درنظر گرفته شده است. چون، در هر تصویر بسیاری از باکس‌ها شامل شی نیستند و تعداد باکس‌های بدون شی بیشتر از با شی هست. برای اینکه مقدار اتلاف باکس‌های بدون شی بر باکس‌های با شی غلبه نداشته باشد، ضریب ۰,۵ پشت آن قرار داده شده تا مقدار اتلاف باکس‌های بدون‌شی کاهش یابد. در نهایت، مقدار احتمال کلاس‌ها باهم مقایسه شده‌اند.

تنظیمات آموزش شبکه **YOLO**

شبکه YOLO به اندازه ۱۳۵ ایپوک با داده‌های Train و Validation از پایگاه داده PASCAL VOC ۲۰۰۷ و ۲۰۱۲ آموزش داده شده است. هنگام تست روی پایگاه داده PASCAL VOC ۲۰۱۲، از داده‌های تست PASCAL VOC ۲۰۰۷ هم برای آموزش استفاده شده است. برای آموزش از تنظیمات Batch size=64، Momentum= ۰,۹ و Weight decay=5e-۴ استفاده شده است.

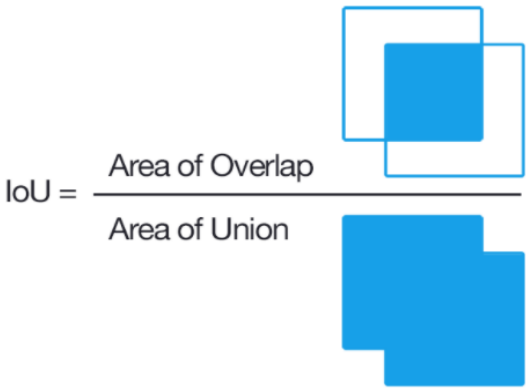
برنامه lr به این‌صورت است که، برای ایپوک‌های اولیه، به صورت آرام نرخ یادگیری از ۰,۰۰۱ تا ۰,۰۱ افزایش می‌یابد (به این فرآیند که نرخ یادگیری از مقدار کوچکی شروع و سپس به‌آرامی زیاد شود،warmup گفته می‌شود که تکنیک بسیار خوبی برای آموزش در ایپوک های اولیه است. چون شبکه در ابتدای فرآیند آموزش در حالت ناپایدار قرار دارد و بنابراین بهتر است نرخ یادگیری کم باشد و سپس به‌آرامی زیاد شود). سپس، آموزش شبکه با نرخ یادگیری ۰,۰۱ به اندازه ۷۵ ایپوک ادامه یافته است، بعد برای ۳۰ ایپوک نرخ یادگیری ۰,۰۰۱ خواهد بود و در نهایت برای ۳۰ ایپوک ۰,۰۰۰۱ خواهد شد.

برای جلوگیری از Overfitting، از Dropout و داده‌سازی (Data Augmentation) استفاده شده است. از Dropout با نرخ ۰,۵ بعد از لایه‌های فولی‌کانکتد استفاده شده است. برای داده‌سازی هم، مقیاس (Scale) و انتقال (Translation) حداکثر ۲۰٪ اندازه تصویر اصلی به‌کار برده شده است. همچنین، به صورت تصادفی Exposure و Saturation در تصویر با فاکتوری حداکثر ۱,۵ در فضای رنگی HSV تنظیم شده است.

معیار سنجش **mAP**

در بسیاری از مقالات تشخیص اشیا برای ارزیابی شبکه خود از معیار mAP (Mean Average Precision) استفاده شده است. ما نیز برای ارزیابی عملکرد شبکه تشخیص اشیا خود (YOLO) از این معیار استفاده می نماییم. ابتدا به سراغ یکسری تعریفات می رویم.

IoU (Intersection over Union): که حاصل از تقسیم اشتراک باندینگ باکس ها پیشبینی‌شده و واقعی به اجتماع آن ها است.



Precision: که میزان صحت خروجی مثبت شبکه را نشان می دهد.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{\text{TP}}{\#\text{ predictions}}$$

Recall: که میزان دقت ویا عملکرد شبکه در میزان پوشش دادگان مثبت نشان می دهد.

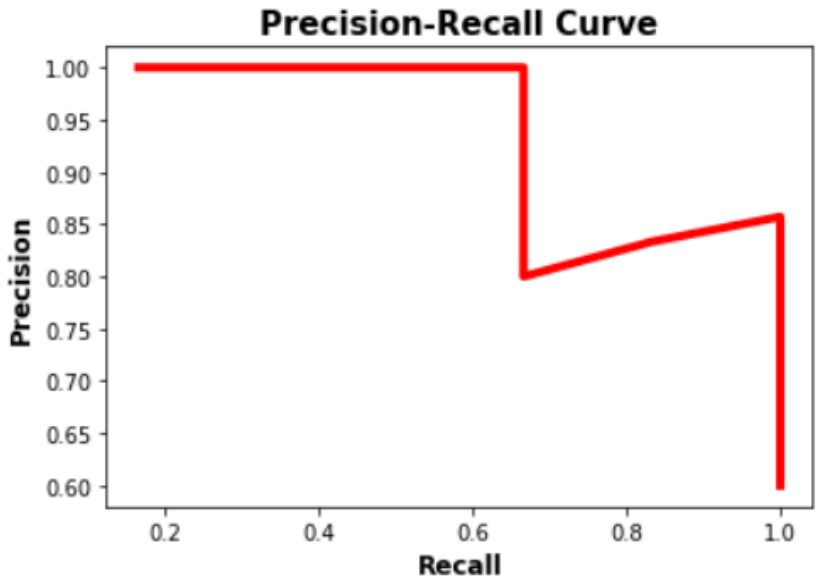
$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{\text{TP}}{\#\text{ ground truths}}$$

TP: پیشبینی درست یک باندیگ باکس واقعی به همراه تشخیص درست شی

FP: تشخیص غلط اشیا و یا پیشبینی اشتباه از باندینگ باکس واقعی

FN: عدم تشخیص باندینگ باکس واقعی

خروجی هر شبکه تشخیص اشیا باید حداقل سه چیز مشخص باشد. که آن‌ها مشخصات مکانی باندینگ باکس های مشخص شده، کلاس تشخیص داده‌شده و confidence score هستند. برای بدست آوردن تشخیص N (Negitive) و P (Positive) از confidence score به عنوان تروشولد استفاده می شود و اگر احتمال تشخیص بزرگتر از confidence score جز تشخیص P حساب می شود در غیر این صورت تشخیص N صورت گرفته است. حال باید تابع Precision P(r) به Recall بدست آورده شود. برای اینکار به ازای تروشولد های متفاوت (به طور مثال از ۰,۲ تا ۰,۹۵) برروی IoU مقادیر TP، FP و FN که در نهایت موجب بدست آمدن Precision و Recall برای هر تروشولد می شود، باید بدست آورده شود که تابع P(r) بدست می آید. این کار برای هر کلاس از شبکه به صورت جدا انجام می شود.



شکل ۹: نمودار یک کلاس $P(R)$.

تعریف AP (Average Precision)

AP برابر است با مساحت زیر نمودار P(r) که برای هر کلاس باید جدا حساب شود. از آنجا که ما مقادیر P(r) را با تروشولد های متفاوت بر روی IoU بدست آوردیم این نمودار گسسته است. در نتیجه ما از میانگین گسسته استفاده می کنیم. در نتیجه مقادیر r، [0,0.1,0.2,...,0.9,1] در نظر گرفته شده است. ولی از آنجا که همه این نقاط در P(r) مقدار ندارند برای محاسبه P در r از درون یابی استفاده می شود. درون یابی مورد استفاده این است که P(r) برابر است با بزرگترین مقدار Precision در Recall موجود بزرگتر از r های تعریف شده.

$$p_{interp(r)} = \max_{\tilde{r}:\tilde{r}\geq r} p(r)$$

فرمول نهایی AP:

$$AP = \frac{1}{11} \sum_{r\in(0,0.1,...,1)} p_{interp(r)}$$

تعریف **mAP** (**mean Average Precision**)

با توجه به اینکه در تشخیص اشیا کلاس های متفاوتی وجود دارد، برای ارزیابی کل نیاز است که میانگین تمام AP های بدست آمده در هر کلاس گرفته شود و به عنوان دقت کل معرفی شود. در نتیجه mAP به صورت زیر تعریف می شود.

$$mAP = \frac{1}{n}\sum_{k=1}^{k=n} AP_k$$

K بیانگر تعداد کلاس است.

آزمایش ارزیابی الگوریتم **YOLO**

پس از فرآیند آموزش YOLO، مشاهده خروجی YOLO برای یک تصویر نمونه بسیار ساده است. یک تصویر نمونه به ابعاد ۳×۴۴۸×۴۴۸ به‌عنوان ورودی به شبکه داده می‌شود و شبکه احتمال‌ها و مختصات باکس‌ها را پیش‌بینی می‌کند. این خروجی‌ها در قالب یک ماتریس ۳۰×۷×۷ برای پایگاه داده PASCAL VOC ارائه می‌شوند. برای هر سلول در ماتریس ۷×۷، دو باکس وجود دارد و در مجموع هم ۴۹ سلول داریم. بنابراین، به‌اندازه ۲×۴۹ باکس در تصویر می‌توانیم رسم کنیم. یعنی برای هر تصویری حداکثر ۹۸ باکس می‌توانیم داشته باشیم، اما با روش‌های آستانه‌گیری و حذف غیر حداکثرها (Non-maximal Supression) بسیاری از این ۹۸ باکس حذف می‌شوند.

محدودیت‌های الگوریتم **YOLO**

با وجود دستاوردهای بزرگی که شبکه YOLO به‌همراه داشته است، اما با محدودیت‌ها و چالش‌هایی هم همراه است. در ادامه، به تعدادی از این محدودیت‌ها اشاره شده است:

- اگرچه YOLO دو باکس برای هر سلول ترسیم می‌کند، اما این باکس از دو کلاس مختلف هستند. یعنی در هر سلول، از یک کلاس دو باکس نمی‌تواند رسم کند.
- YOLO در تشخیص اشیای کوچک در تصویر، مانند دسته پرندگان چالش دارد.

- YOLO در تشخیص اشیای با ابعاد جدید و غیرمعمولی که در فرآیند آموزش ندیده باشد، مشکل دارد.
- YOLO از ویژگی‌های کلی اشیا برای تشخیص بهره می‌برد، چون تنها از ویژگی‌های لایه خروجی برای تشخیص اشیا استفاده می‌کند که این ویژگی‌ها کلی هستند و به‌خاطر کوچک شدن‌های متوالی جزئیات از بین می‌روند.

نتایج ارزیابی الگوریتم YOLO

بهتر است ابتدا به یکی از مهم‌ترین نقطه قوت‌های یولو، یعنی سرعت بپردازیم. در جدول زیر، نتایج مقایسه سرعت و دقت YOLO با سایر کارها را مشاهده می‌کنید. جدول از دو بخش Real-Time Detectors و Less Than Real-Time تشکیل شده است. بخش اول (Real-Time Detectors)، الگوریتم‌های تشخیص اشیای سریع را نشان می‌دهد؛ در این بخش، مشاهده می‌کنید که Fast YOLO و YOLO هردو هم سرعت بسیار بالا (FPS) و هم دقت بالاتر (mAP) نسبت به ۱۰۰Hz DPM و ۳۰Hz DPM دارند.

بخش دوم (Less Than Real-Time) مربوط به الگوریتم‌هایی می‌شود که بلادرنگ نیستند (سریع نیستند). در اینجا بجای YOLO از YOLO VGG-۱۶ استفاده شده است. در YOLO VGG-۱۶ از شبکه VGG-۱۶ برای تشخیص اشیا بجای مدل ۲۴ لایه‌ای YOLO استفاده شده است. VGG-۱۶ دقت بهتری دارد، اما سرعت پایین‌تری دارد. دراین‌حالت، سرعت YOLO به ۲۱ فریم‌برثانیه افت کرده، اما باز هم سریع‌تر از سایر الگوریتم‌هاست. درعین‌حال، در بخش mAP نسبت به بعضی الگوریتم‌ها مانند Fast R-CNN کمی ضعیف‌تر است. البته دقت کنید همین Fast R-CNN، سرعت پردازش ۰٫۵ فریم‌برثانیه دارد!

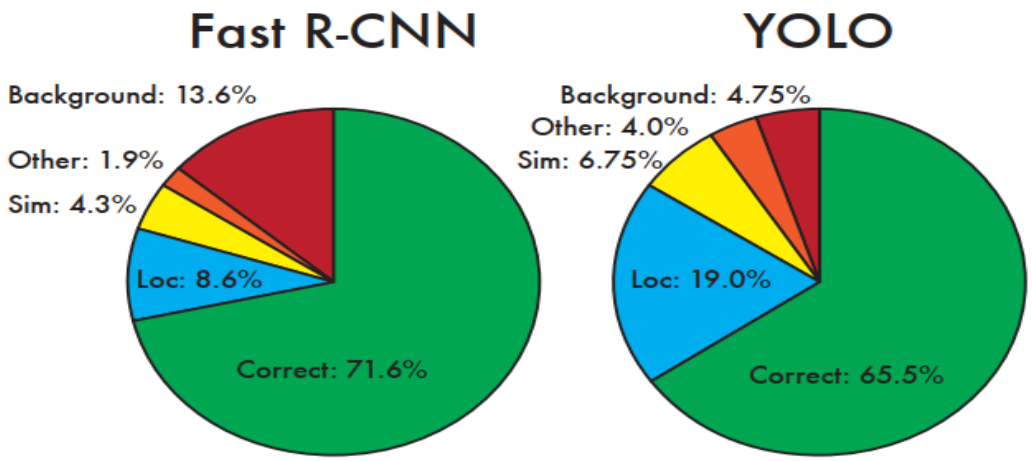
Real-Time Detectors	Train	mAP	FPS
100Hz DPM [31]	2007	16.0	100
30Hz DPM [31]	2007	26.1	30
Fast YOLO	2007+2012	52.7	**155**
YOLO	2007+2012	**63.4**	45
Less Than Real-Time			
Fastest DPM [38]	2007	30.4	15
R-CNN Minus R [20]	2007	53.5	6
Fast R-CNN [14]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[28]	2007+2012	73.2	7
Faster R-CNN ZF [28]	2007+2012	62.1	18
YOLO VGG-16	2007+2012	66.4	21

جدول ۱: مقایسه نتایج تشخیص اشیای *YOLO* با سیستم‌های بلادرنگ و کمتر بلادرنگ [لینک مرجع]

بخش بعدی ارزیابی، مربوط به مقایسه دقیق بین YOLO و Fast R-CNN هست. در اینجا، برای تشخیص اشیا، خطاهای مختلفی تعریف شده و سپس اندازه‌گیری شده که هریک از این دو الگوریتم چقدر خطا دارند. در شکل زیر، نمودار دایره‌ای تحلیل خطا بین YOLO و Fast R-CNN را مشاهده می‌کنید. به‌صورت خلاصه، نمودار شکل ۹ را به‌صورت زیر می‌توان مقایسه نمود:

- correct: اگر کلاس شی درست تشخیص داده شده باشد و مقدار IOU بزرگتر از ۰٫۵ باشد. دراین مورد، Fast R-CNN عملکرد بهتری دارد (ناحیه سبز).
- localization: اگر کلاس شی درست تشخیص داده شده باشد ولی مقدار IOU بین [۰٫۵ ۰٫۱] باشد. دراین خطا، عملکرد Fast R-CNN بهتر است (ناحیه آبی).
- Background: مقدار IOU کمتر از ۰٫۱ باشد. در این خطا، عملکرد یولو بهتر است.

با بررسی سه مورد بالا، می‌توان به این نتیجه رسید که عملکرد Fast R-CNN درکل بهتر از یولو است. YOLO خطای بیشتری در موقعیت‌یابی اشیا دارد و این یک چالش برای YOLO محسوب می‌شود. اما درعین‌حال، YOLO خطای پس‌زمینه کمتری نسبت به Fast R-CNN دارد.



شکل ۱۰: نمودار دایره‌ای مقایسه *YOLO* و *Fast R-CNN*[لینک مرجع]

YOLO در مجموعه تست پایگاه داده VOC ۲۰۱۲، به mAP ۵۷٫۹٪ رسیده است. با توجه به جدول زیر، این مقدار پایین‌تر از بهترین نتایج ارائه شده در حال حاضر است و نزدیک به R-CNN با استفاده از VGG-۱۶ است. در مقایسه با رقیب‌های نزدیک مانند R-CNN، سیستم YOLO بیشتر درگیر چالش تشخیص اشیای کوچک است. به‌عنوان مثال، YOLO در تشخیص اشیایی مانند sheep،bottle و tv/monitor عملکردی

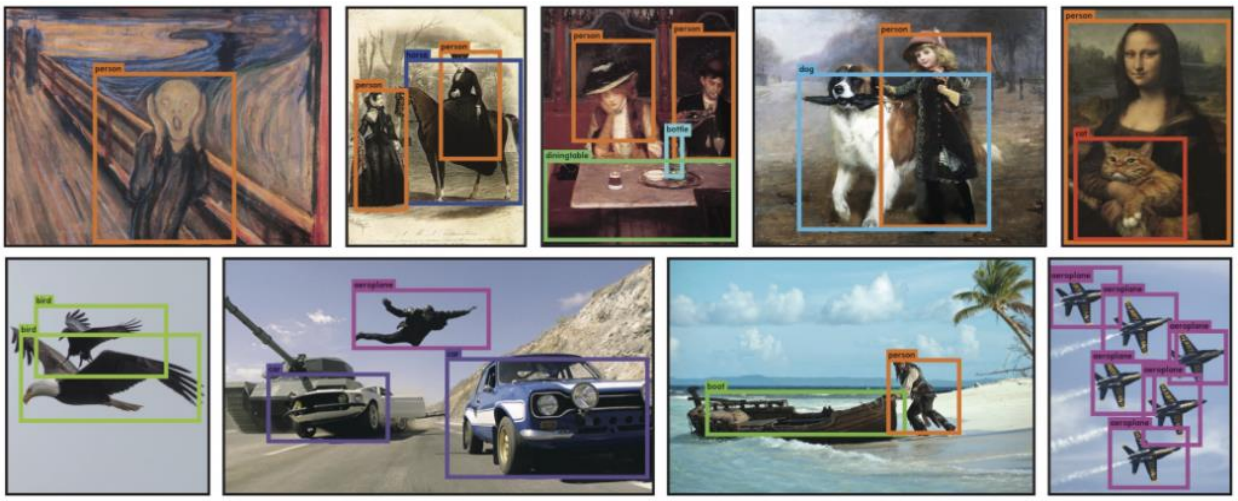
۸-۱۰٪ پایین‌تر نسبت به R-CNN دارد. اما، در سایر کلاس‌ها مانند گربه، YOLO عملکرد بهتری در تشخیص دارد. در بخش قبل، YOLO با Fast R-CNN ترکیب شد که مشاهده می‌کنید این مساله باعث شده که دقت بالایی حاصل شود و رتبه پنجم در جدول را دارد. البته، دراین‌حالت دیگر باید از مقوله سرعت که از ویژگی‌های مهم YOLO است، چشم‌پوشی کنیم.

VOC 2012 test	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
MR_CNN_MORE_DATA [11]	**73.9**	**85.5**	**82.9**	**76.6**	**57.8**	**62.7**	**79.4**	77.2	86.6	**55.0**	**79.1**	**62.2**	87.0	**83.4**	**84.7**	78.9	45.3	73.4	65.8	80.3	74.0
HyperNet_VGG	71.4	84.2	78.5	73.6	55.6	53.7	78.7	**79.8**	87.7	49.6	74.9	52.1	86.0	81.7	83.3	**81.8**	**48.6**	**73.5**	59.4	79.9	65.7
HyperNet_SP	71.3	84.1	78.3	73.3	55.5	53.6	78.6	79.6	87.5	49.5	74.9	52.1	85.6	81.6	83.2	81.6	48.4	73.2	59.3	79.7	65.6
Fast R-CNN + YOLO	70.7	83.4	78.5	73.5	55.8	43.4	79.1	73.1	**89.4**	49.4	75.5	57.0	**87.5**	80.9	81.0	74.7	41.8	71.5	68.5	**82.1**	67.2
MR_CNN_S_CNN [11]	70.7	85.0	79.6	71.5	55.3	57.7	76.0	73.9	84.6	50.5	74.3	61.7	85.5	79.9	81.7	76.4	41.0	69.0	61.2	77.7	72.1
Faster R-CNN [28]	70.4	84.9	79.8	74.3	53.9	49.8	77.5	75.9	88.5	45.6	77.1	55.3	86.9	81.7	80.9	79.6	40.1	72.6	60.9	81.2	61.5
DEEP_ENS_COCO	70.1	84.0	79.4	71.6	51.9	51.1	74.1	72.1	88.6	48.3	73.4	57.8	86.1	80.0	80.7	70.4	46.6	69.6	**68.8**	75.9	71.4
NoC [29]	68.8	82.8	79.0	71.6	52.3	53.7	74.1	69.0	84.9	46.9	74.3	53.1	85.0	81.3	79.5	72.2	38.9	72.4	59.5	76.7	68.1
Fast R-CNN [14]	68.4	82.3	78.4	70.8	52.3	38.7	77.8	71.6	89.3	44.2	73.0	55.0	**87.5**	80.5	80.8	72.0	35.1	68.3	65.7	80.4	64.2
UMICH_FGS_STRUCT	66.4	82.9	76.1	64.1	44.6	49.4	70.3	71.2	84.6	42.7	68.6	55.8	82.7	77.1	79.9	68.7	41.4	69.0	60.0	72.0	66.2
NUS_NIN_C2000 [7]	63.8	80.2	73.8	61.9	43.7	43.0	70.3	67.6	80.7	41.9	69.7	51.7	78.2	75.2	76.9	65.1	38.6	68.3	58.0	68.7	63.3
BabyLearning [7]	63.2	78.0	74.2	61.3	45.7	42.7	68.2	66.8	80.2	40.6	70.0	49.8	79.0	74.5	77.9	64.0	35.3	67.9	55.7	68.7	62.6
NUS_NIN	62.4	77.9	73.1	62.6	39.5	43.3	69.1	66.4	78.9	39.1	68.1	50.0	77.2	71.3	76.1	64.7	38.4	66.9	56.2	66.9	62.7
R-CNN VGG BB [13]	62.4	79.6	72.7	61.9	41.2	41.9	65.9	66.4	84.6	38.5	67.2	46.7	82.0	74.8	76.0	65.2	35.6	65.4	54.2	67.4	60.3
R-CNN VGG [13]	59.2	76.8	70.9	56.6	37.5	36.9	62.9	63.6	81.1	35.7	64.3	43.9	80.4	71.6	74.0	60.0	30.8	63.4	52.0	63.5	58.7
YOLO	57.9	77.0	67.2	57.7	38.3	22.7	68.3	55.9	81.4	36.2	60.8	48.5	77.2	72.3	71.3	63.5	28.9	52.2	54.8	73.9	50.8
Feature Edit [33]	56.3	74.6	69.1	54.4	39.1	33.1	65.2	62.7	69.7	30.8	56.0	44.6	70.0	64.4	71.1	60.2	33.3	61.3	46.4	61.7	57.8
R-CNN BB [13]	53.3	71.8	65.8	52.0	34.1	32.6	59.6	60.0	69.8	27.6	52.0	41.7	69.6	61.3	68.3	57.8	29.6	57.8	40.9	59.3	54.1
SDS [16]	50.7	69.7	58.4	48.5	28.3	28.8	61.3	57.5	70.8	24.1	50.7	35.9	64.9	59.1	65.8	57.1	26.0	58.8	38.6	58.9	50.7
R-CNN [13]	49.6	68.1	63.8	46.1	29.4	27.9	56.6	57.0	65.9	26.5	48.7	39.5	66.2	57.3	65.4	53.2	26.2	54.5	38.1	50.6	51.6

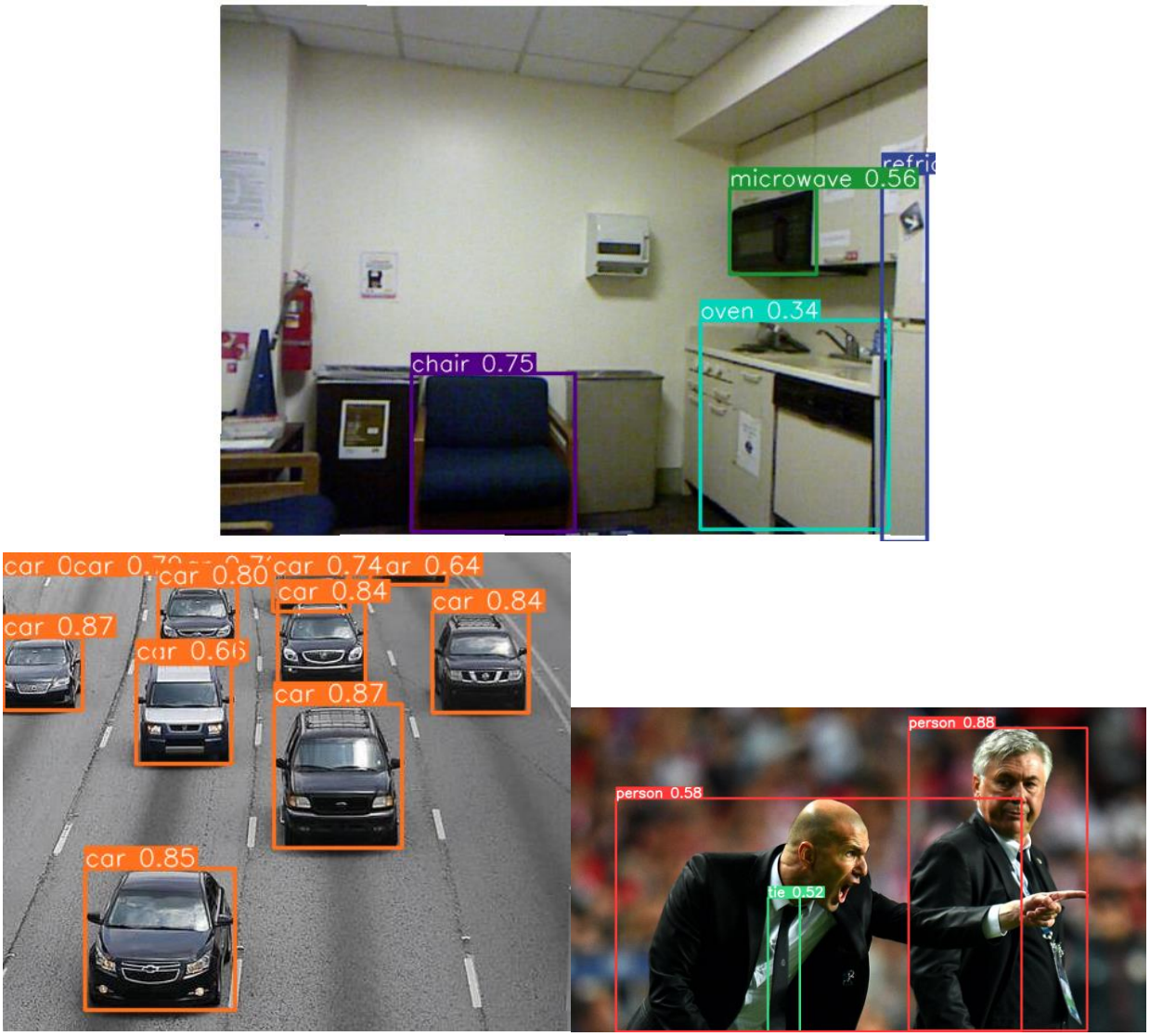
جدول ۳: مقایسه نتایج سیستم‌های تشخیص اشیای مختلف و جایگاه روش ترکیبی *Fast R-CNN + YOLO* در بین برترین‌ها [لینک مرجع]

در بخش‌های ابتدایی، یکی از ویژگی‌های مثبتی که برای YOLO برشمردیم، قابلیت تعمیم‌پذیری آن بود. ادعا شده بود که YOLO در داده‌های با توزیع متفاوت عملکرد خوبی دارد و می‌تواند تعمیم‌پذیری بالای خود را به رخ دیگر سیستم‌ها بکشد! شکل زیر نشان می‌دهد که چگونه دقت سایر سیستم‌های تشخیص به‌شدت افت می‌کند.

نتایج



شکل ۱۱: نمونه خروجی‌های سیستم تشخیص اشیای *YOLO* در تصاویر هنری و غیره [لینک مرجع]



۱۱: نمونه خروجی‌های سیستم تشخیص اشیای ***YOLO*** برای یک داده‌ی ***NYU Depth Dataset V2*** در شکل بالایی و داده‌های دیگر در شکل‌های پایینی.

تخمین عمق با شبکه‌های یادگیری عمیق

در تخمین عمق به کمک یادگیری ما به دنبال پیدا کردن مسافت هر اشیا (به‌طورکلی هر پیکسل تصویر) از دوربین آن هستیم. این امر از اهمیت زیادی برخورد دار است مخصوصاً در مباحث مهم ماشین‌های اتوماتیک و رباتیک که لازم دارند تا برای تصمیم‌گیری‌های خود فاصله هر اشیا داشته باشند. این کار به‌صورت چند دوربین با زاویه‌های متفاوت هم انجام می‌شود ولی ما به دنبال تشخیص عمق یک تصویر هستیم که اصطلاحاً به آن single-view depth estimation می‌گویند. الگوریتم‌ها و شبکه‌های متفاوتی تا الآن ارائه‌شده است ولی با توجه به مشکل سخت‌افزاری در فرایند یادگیری و نیاز به‌سرعت در فرایند تست، ما برای این قسمت از مقاله FastDepth [1] الهام گرفتیم. شبکه‌ی معرفی‌شده در این مقاله بدون در نظر گرفتن Network Pruning پیاده‌سازی شده است. برای داده‌های آموزشی از قسمت برچسب دهی شده داده‌های NYU Depth Dataset V2 استفاده‌شده است. تعداد این داده‌ها ۱۴۴۹ در ۴۶۴ صحنه که در ۳ شهر متفاوت است. فضای داده‌ها بسته است. ویژگی بسیار مهم این داده‌ها برچسب دهی آن‌ها است که برای انواع کارها مانند تشخیص اشیا، تخمین عمق، دسته‌بندی چند کلاسه و ... مناسب است، به همین خاطر دارای برچسب‌های متفاوت و ابزارهای مناسب است. با توجه به مبحث تخمین عمق ما فقط از داده‌های rgb (که تصویر هر صحنه است) و برچسب عمق مشخص‌شده هر پیکسل با توجه به تصویر اصلی آن استفاده می‌کنیم. تصویر اصلی rgb دارای ابعاد ۴۸۰ در ۶۴۰ است بدین منظور، برچسب عمق آن نیز همین سایز ولی بعد ۱ است تا فقط برای هر پیکسل عمق آن با واحد متر معرفی کرده باشد.

در ادامه به معرفی جزئیات روش پیشنهادی FastDepth [1] و پیاده‌سازی می‌پردازیم.

شبکه FastDepth به‌صورت انکودر و دیکدر است. ایشان نشان داده‌اند که عمده علت افزایش زمان اجرا عمق زیاد دیکدر است. درنتیجه، روشی را پیش نهاد دادند که عمق دیکدر کم شود ولی دقت به‌خوبی روش‌های قبلی باشد. اولین پیشنهاد این است که در فاز انکودر از شبکه mobilenet v2 پیش آموزش دید با imagenet استفاده شود. چراکه این شبکه نسبت به بقیه شبکه‌های معروف (vgg16 و resnet) بسیار سبک‌تر است و دقت خوبی دارد. پیشنهاد دوم این است که مانند شبکه U-net از skip connection استفاده شود بین انکودر و دیکدر و این skip connection جمع شونده باشد تا میزان پارامترها کاهش یابد (در مقابل concatenative). پیشنهاد آخر که در این پروژه استفاده‌شده است، استفاده از NNConv5 در دیکدر به‌عنوان Upsample Operation به‌جای DeConv، UpConv و UpProj که سرعت شبکه را زیاد می‌کند، تعداد عملیات جمع و ضرب را به‌شدت کاهش می‌دهد و دقت خوبی دارد (حتی بهتری نسبت به بقیه روش‌ها دارد. NNConv5 همان کانولوشن با فیلتر ۵ در ۵ است که با مقیاس ۲ و درون‌یابی nearestneighbor بسط پیدا می‌کند.

[1] : D. Wofk and F. Ma and T.Yang and S. Karaman and V. Sze,” FastDepth: Fast Monocular Depth Estimation on Embedded Systems”, arXiv, 2019.

برای کانولوشن های با سایز بالا از depthwise و pointwise استفاده شده است. که depthwise به این صورت عمل می کند که برای هر عمق ورودی یک کانولوشن جدا استفاده می کند و در pointwise کانولشن ۱*۱ است که فقط عمق را تغییر می دهد. قرار گرفتن depthwise و pointwise پشت سر هم، باعث خروجی کانولوشن معمولی می شود با این تفاوت که تعداد عملیات جمع و ضرب به شدت کاهش می یابد (در شکل های زیر عملکرد هر کدام به صورت جداگانه مشاهده می نمایید).

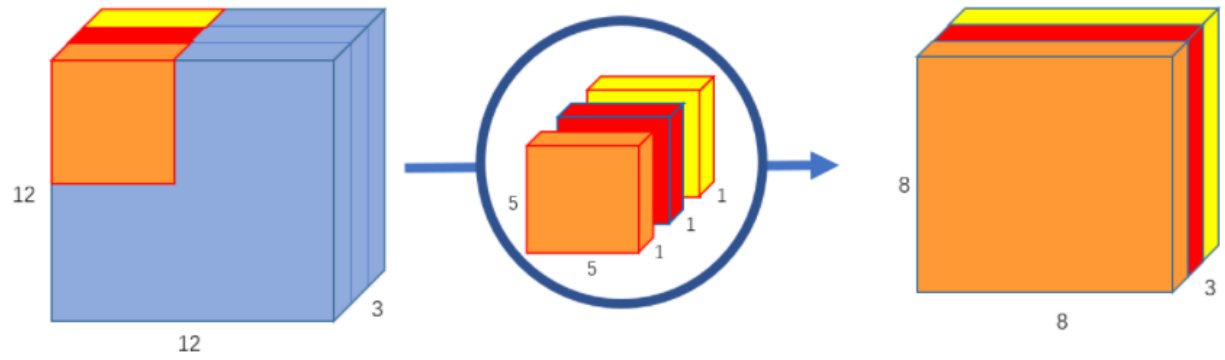


Image 6: Depthwise convolution, uses 3 kernels to transform a 12×12×3 image to a 8×8×3 image

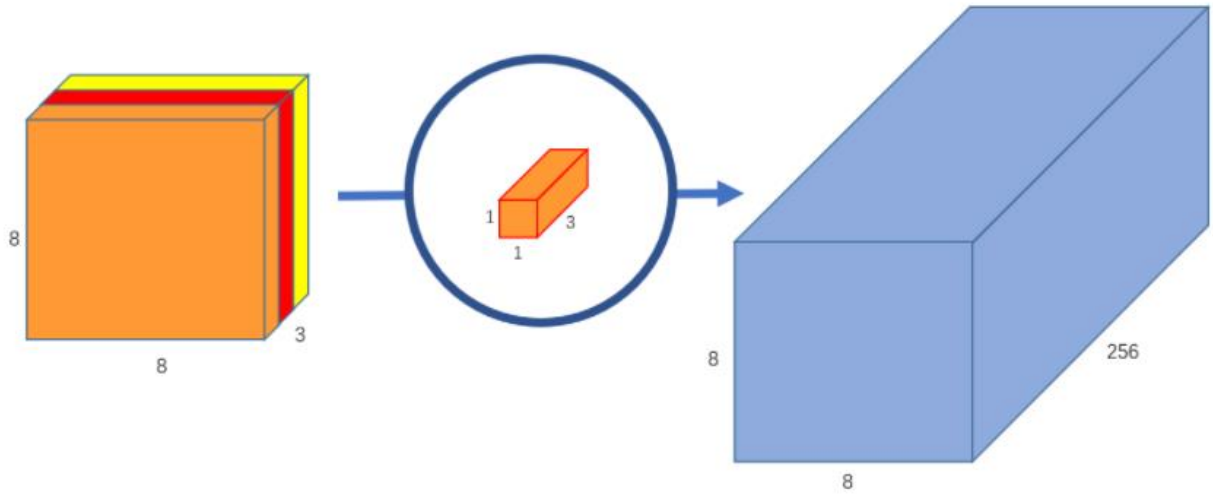


Image 8: Pointwise convolution with 256 kernels, outputting an image with 256 channels

شکل ۱۲: بالایی معرف *depth convolution* و پایینی معرف *pointwith convolution* .

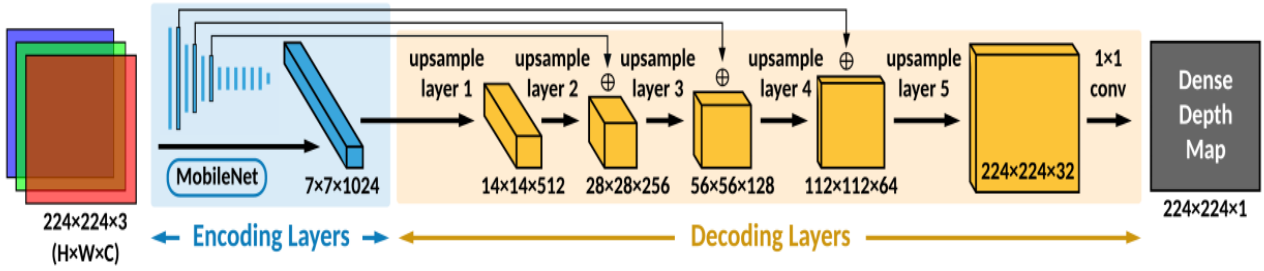


Fig. 2: Proposed network architecture. Dimensions of intermediate feature maps are given as height × width × # channels. Arrows from encoding layers to decoding layers denote additive (rather than concatenative) skip connections.

شکل ۱۳: نمای کلی شبکه عمق سنجی به همراه مشخصات هر لایه .

با توجه به اینکه در شبکه معرفی‌شده بالا ورودی ۲۲۴ در ۲۲۴ است ولی ورودی دیتاست اصلی ۴۸۰ در ۶۴۰ است، از ری سایز استفاده‌شده است. توجه شود قبل از ری سایز از سنتر کراپ به سایز ۳۳۶ در ۳۳۶ استفاده‌شده است تا جزئیات تصویر زیاد کوچک نشوند. برای آموزش بر روی‌داده‌های آموزش علاوه بر موارد بالا با توجه به مقاله [2] از چرخش تصویر به‌صورت ندوم بین (-0.5,0.5)، فلیپ تصویر به‌صورت افقی با احتمال ۰,۵ و Color Jitter به‌صورت تصادفی برای تغییرات brightness, contrast, and saturation به میزان [0.6,1.4] استفاده‌شده است. در پایان تمام تصاویر ورودی به شبکه با میانگین [۰,۴۸۵ ,۰,۴۵۶ ,۰,۴۰۶] و واریانس [۰,۲۲۹ ,۰,۲۲۴ ,۰,۲۲۵] نرمالیزه می‌شود. تمامی موارد از کتابخانه torchvision.transforms در کد می‌توانید مشاهده نمایید. توجه شود ۸۰ درصد داده‌های این دیتاست به داده‌های آموزش اختصاص یافته اند و ۱۵ درصد به داده‌های ارزیابی (validation) و ۵ درصد به داده‌های تست تخصیص دارند. البته چون، داده‌های تست و ارزیابی شبکه نمی بینن لازم نبود داده‌ی تست تعریف شود ولی از آنجا که سخت افزار من اجازه این حجم از داده نمی داد مجبور شده داده تست تعریف کنم تا اصلا استفاده نشود (یعنی کلا ۹۵ درصد داده‌ها شامل آموزش و ارزیابی کار شده است).

برای آموزش از بینه سازی SGD با نرخ یادگیری ۰,۰۱ که در هر ۵ ایپاک ۲۰ درصد باقی می‌ماند، ممنتوم ۰,۹ و ضریب کاهنده وزن ۰,۰۰۰۱ استفاده‌شده است. نکته مهم از تابع تلفات نرم ۱ به‌جای MSE استفاده شد. زیرا این نرم نسبت به هر میزان خطا یک میزان گرادیان دارد تا شبکه را مجبور نکند به نقاطی که خطا زیاد دارند بیشتر توجه کند که این امر باعث می‌شود شبکه سعی کند نقاط با خطا کم هم به یک‌میزان نگاه کند.

معیار های ارزیابی تشخیص عمق

برای سنجش از سه معیار، REL متوسط اندازه خطاها که همان تابع هزینه نرم یک ما است (چون در تابع هزینه از متوسط گیری استفاده می‌کنیم)، RMSE ریشه متوسط مجذور خطا است و δ_i که به‌طورکلی به‌صورت زیر تحریف می‌شود.

[2]: F. Ma and S. Karaman, “Sparse-to-dense: depth prediction from sparse depth samples and a single image,”*IEEE International Conference on Robotics and Automation (ICRA)*, 2018

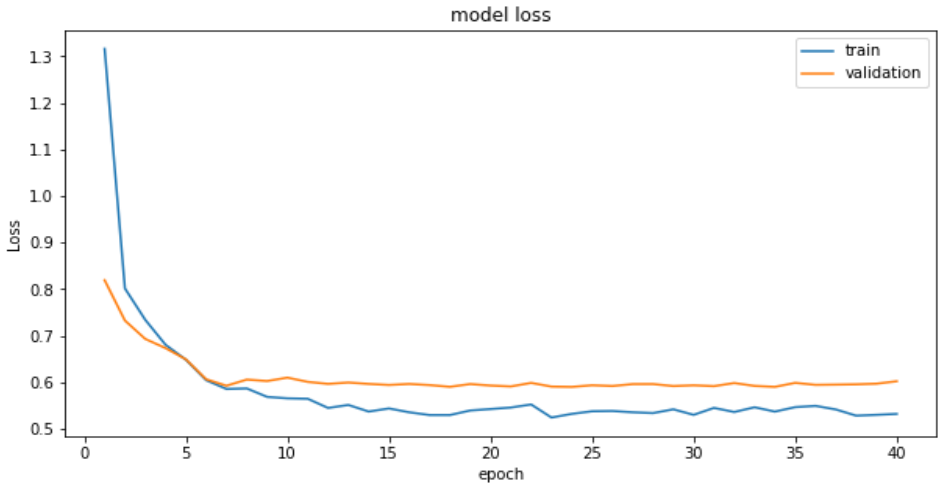
$$\delta_i = \frac{\mathbf{card}\left(\left\{\hat{y}_i : \max\left\{\frac{\hat{y}_i}{y_i}, \frac{y_i}{\hat{y}_i}\right\} < 1.25^i\right\}\right)}{\mathbf{card}\left(\{y_i\}\right)}$$

که بیانگر تعداد مجموعه صورت که بیشترین فاصله نسبی بین عمق واقعی بر پیش‌بینی‌شده و برعکس، کمتر از تروشولد ۱,۲۵ به توان i باشد، بر تعداد کل مجموعه ارزیابی (که شامل پیکسل‌ها و تعداد داده‌های ارزیابی می‌شود). درواقع می‌گوید چه تعداد از عمق پیکسل‌های پیش‌بینی‌شده از تروشولد موردنظر کمتر است و نسبت به‌کل پیکسل‌ها نرمالیزه می‌شود. هر چه i بیشتر شود تروشولد بیشتر می‌شود و دقت این معیار به یک نزدیک می‌شود. ولی با توجه به مقالات ما از δ_1 (i برابر ۱) و δ_2 (i برابر ۲) استفاده می‌کنیم که گویا این است که قدر مطلق اختلاف بین پیش‌بینی و واقعیت باید از ۲۵ درصد برای δ_1 و ۵۶ درصد برای δ_1 مقدار کوچک‌تر (پیش‌بینی یا واقعیت)، کمتر باشد تا به‌عنوان عملکرد درست شبکه ارزیابی شود.

از این معیارها، معیار δ_i عملی‌تر است زیرا نشان می‌دهد چه تعداد از تشخیصات در محدوده قابل‌قبول قرارگرفته است که این محدوده به‌طور وفقی با افزایش عمق پیکسل، افزایش می‌یابد و اجازه می‌دهد در عمق زیاد خطای بیشتری قابل‌قبول باشد که در دو معیار دیگر نیست. این معیار دقت شبکه را ارزیابی می‌کند (ماکسیمم آن‌یک است و هرچه نزدیک به یک باشد بهتر است) ولی درصورتی‌که معیار RMSE متوسط خطا را برای هر پیکسل نشان می‌دهد و ممکن است عمق چند پیکسل خیلی با واقعیت فاصله دارد و این امر تأثیر به سزایی در ارزیابی شبکه دارد که حتی در صورت عملکرد خوبِ شبکه در تشخیص عمق بقیه پیکسل‌ها ارزیابی شبکه به‌درستی انجام نمی‌شود.

نتایج

آموزش : در فرایند آموزش بهترین مدل که شامل کمترین تلفات L1 بود ذخیره‌شده است.



نتایج بهترین مدل درداده‌های ارزیابی :

REL : 0.589

RMSE: 0.861

δ_1 : 0.643

δ_2: 0.895

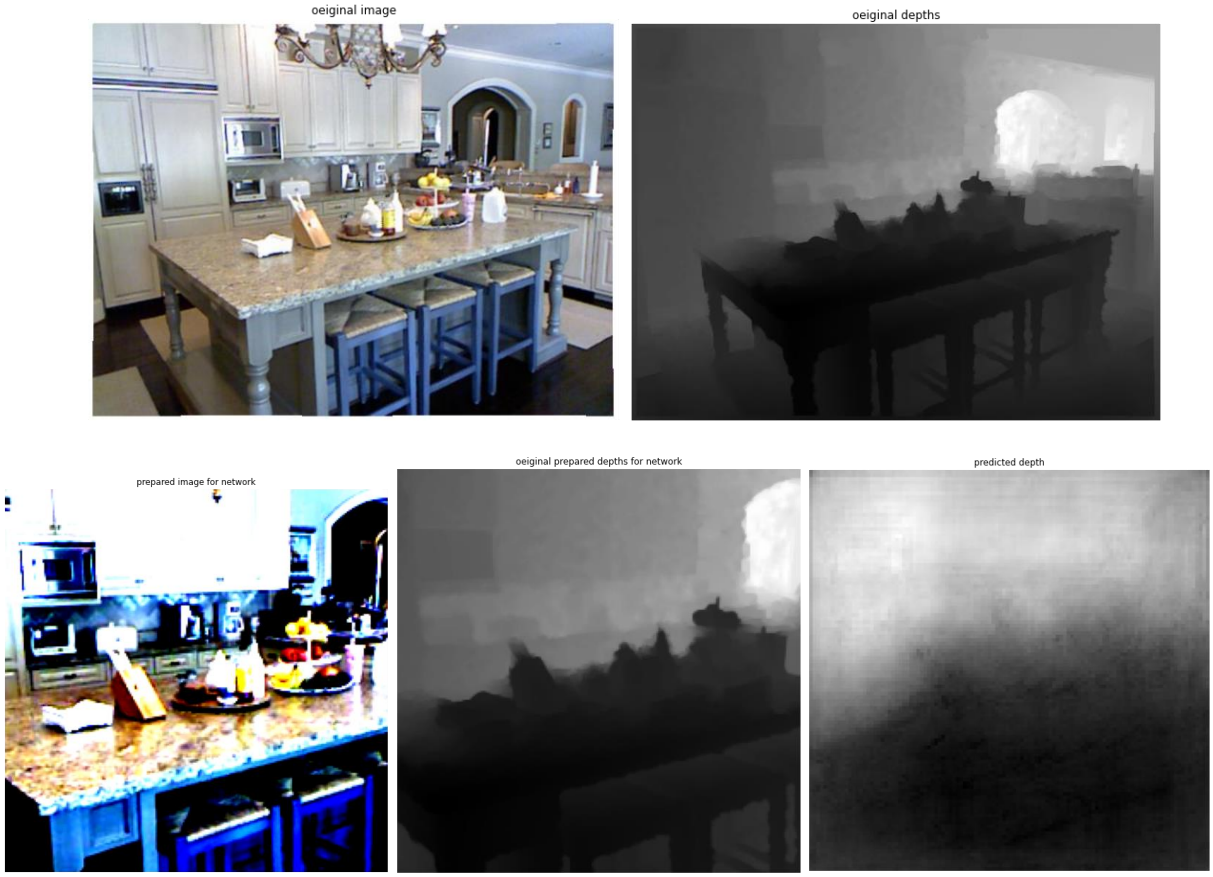
δ_3 : 0.971

مشاهده می شود هرچه i در δ_i بیشتر می شود دقت شبکه به یک نزدیک می شود ولی بااین‌حال برای ما دقت δ_1 و δ_2 اهمیت دارد که به ترتیب بیانگر اختلاف ۲۵ درصد و ۵۶ درصد است.

طبق معیارها شبکه عملکرد مناسبی دارد و با شبکه‌ی اصلی معرفی‌شده در مقاله [1] ازلحاظ معیارها (RMSE: 0.604 و δ_1 : 0.771)فاصله زیادی ندارد. از عوامل مهم نرسیدن به نتایج مشابه کمبود امکانات سخت‌افزاری که موجب شد انکدر علاوه بر این‌که سبک باشد، از شبکه پیش آموزش‌دیده استفاده شود و اگر انکدر هم آموزش می‌دید تعداد پارامترها ۲,۵ برابر می‌شد. دیگر استفاده نکردن از تمام داده‌ها (۹۵ درصد استفاده شد که در قسمت قبل توضیح داده شد). بااین‌حال نتایج خوب است.

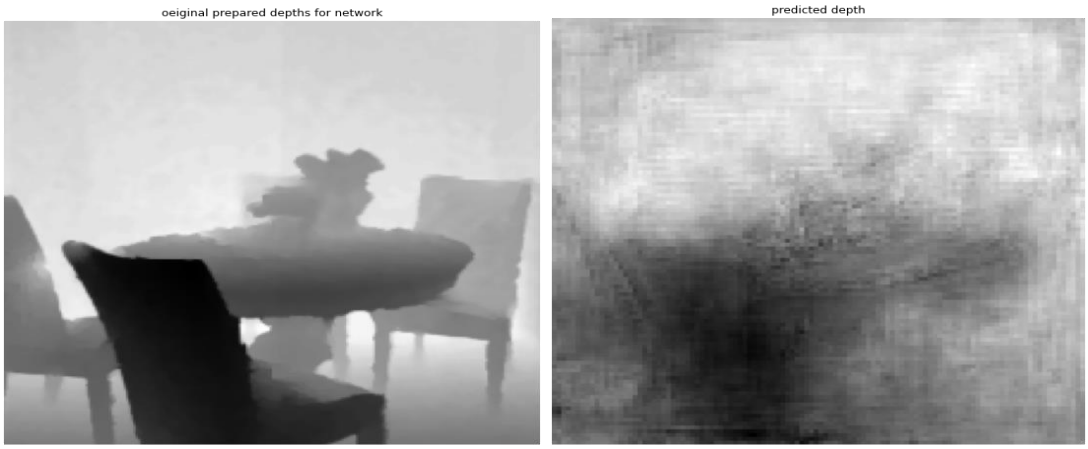
[1] : D. Wofk and F. Ma and T.Yang and S. Karaman and V. Sze,” FastDepth: Fast Monocular Depth Estimation on Embedded Systems”, arXiv, 2019.

خروجی شبکه بر روی‌داده‌های ارزیابی:



تصاویر ردیف بالا از چپ به ترتیب تصویر اصلی و عمق مرتبط به آن است. تصویر ردیف پایین از چپ به ترتیب تصویر اصلی نرمالیزه شده و کراپ و ری سایز شده به ۲۲۴ در ۲۲۴ و عمق واقعی آن و عمق تشخیص داده آن است.

چند مورد از داده‌های ارزیابی (از چپ به ترتیب عمق واقعی و عمق تخمینی)



مشاهده می شود نتایج خوب است. ازآنجاکه ما می‌خواهیم در این پروژه عمق مرکز اجسام بسنجیم نتایج بهتر هم می شود چراکه همان‌طور که مشاهده می‌کنید نتایج در مرکز اجسام دقیق‌تر است و علت آن این است که چالش عمق در لبه اجسام بیشتر است چراکه در لبه در همسایگی یک پیکسل عمق‌ها متفاوت بافاصله زیادی وجود دارد. برای بهبود شبکه در پروژه می‌توان از میانگین همسایگی مرکز جسم به‌عنوان تصمیم عمق استفاده کرد که به علت اینکه این روش در لبه‌ها تأثیر منفی می‌گذارد در کلیه پیکسل‌ها انجام نشد ولی در عمق سنجی اشیا استفاده می شود.

کد این قسمت در فایل کد ها در پوشه depth-estimation قراردارد. برای اجرا نیاز به پارامتر های پیش آموزش mobilenet است که به عنوان mobilenet_model.pth.tar در فایل کد قرار دارد. در انتها پارامتر های بهترین مدل به نام best_model ذخیره سازی شد تا در اپلیکیشن استفاده شود.

معرفی شبکه‌ی نهایی برای تشخیص اشیاء و عمق اشیاء موجود در تصویر

در نهایت نحوه‌ی استفاده از دو شبکه‌ای که به صورت مجزا برای تشخیص اشیا و تخمین عمق معرفی شده بود، برای تشخیص اشیا و به همرا تخمین عمق برای اشیاء که در فاصله‌ی معینی قرار دارند بیان می شود. توجه شود که در نحوه استفاده از این دو شبکه به اینکه دو شبکه‌ی معرفی شده دارای سرع بالایی دارند توجه شده است.

نحوه اتصال دو شبکه

با توجه به اینکه هردو شبکه ی عمق سنجی و تشخیص اشیا سرعت بسیار بالایی دارند (جز مزیت هر دو بود) تصویر یکبار به ۲۲۴ در ۲۲۴ ریسایز کرده و عمق سنجی انجام می شود و بعد تصویر اصلی به شبکه ی تشخیص اشیای Yolo داده تا فرایند تشخیص اشیا انجام شود. سپس با مختصات ماکسیسمم و مینیمم هر شی، مرکز آن را بدست آورده. همان طور که در قسمت عمق سنجی گفته شد، برای اینکار از پنجره ای برای میانگین گرفتن به طول ۵ به مرکز شی برای عمق سنجی دقیق تر استفاده شده است. حال از کار بر حداکثر عمق مورد نظر گرفته می شود و اشیایی که در عمق کمتر هستند در خروجی نمایش داده می شود.

ارزیابی شبکه نهایی به صورت تئوری

در بخش های قبل به صورت جداگانه برای هرشبکه معیارهای ارزیابی شبکه ها معرفی شد. معیارهای ارزیابی شبکه ی تخمین عمق REL، RMSE و δ_i با i های مختلف بود. از این معیار ها δ_1 و δ_2 اهمیت بیشتری داشتند (در قسمت خودش توضیح داده شد). ولی برای تولید معیار نهایی از δ_2 استفاده شده است. علت آن است که دقت بر حسب δ_2 بیشتر است (نزدیک به یک) و در پروژه نهایی (اتصال دو شبکه) ما نیاز داریم در تخمین عمق دقت چند پیکسل (مرکز اشیا و اطراف آن ها درون پنجره ای به طول ۵) بالا باشد و چون این پیکسل ها در مرکز بافت شی وجود دارد و دقت شبکه ی عمق سنج همان طور که در قسمت قبل گفته شد بالا تر است از لبه های اشیا. برای معیار ارزیابی شبکه تشخیص اشیا YOLO از همان mAP استفاده شده است. از آن جا که این دقت ها در دو داده‌ی متفاوت بدست آمده است و جنس دو شبکه فرق می کند پیشنهاد ما برای معیار نهایی برای شبکه نهایی پروژه میانگین دو معیار δ_2 و mAP است.

در نتیجه این دو معیار $\text{MmAP\&}\delta_2$ می نامیم و دقت نهایی پروژه برابر است با:

$$\text{MmAP\&}\delta_2 = \frac{\text{mAP} + \delta_2}{2} = \frac{0.579 + 0.895}{2} = 0.737$$

نتایج عملی

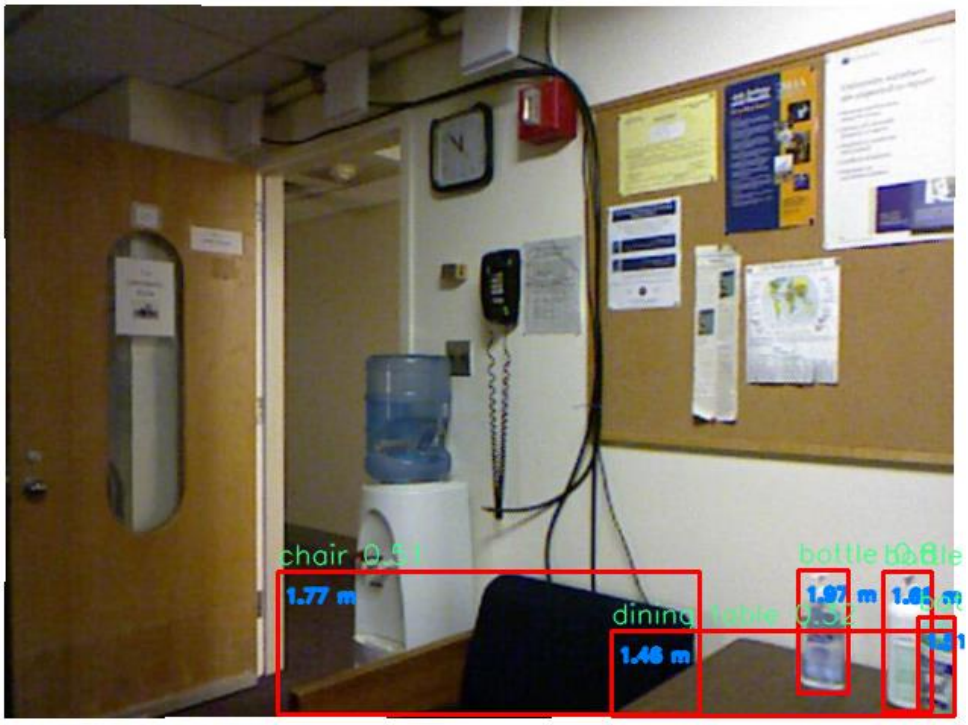
برای استفاده از کاربد تشخیص اشیا و عمق آن‌ها ما تابعی درست کردیم تحت عنوان detection که دوتا ورودی می گیرد یک تصویر مورد نظر با هر سایزی و دومی حداکثر فاصله (برحسب متر) که اشیا با فاصله بیشتر از آن لازم نیست شناسایی شود. کاربر می تواند تصویر مورد نظر خود را به همراه حداکثر فاصله را مشخص کند تا خروجی مورد نیاز خود را که تشخیص اشیا به همراه عمق سنجی است، دریافت کند. به این تابع می توان فریم های وبکم یا دوربین داد و از آن خروجی گرفت.

تصاویر دیتاست NYU Depth Dataset V2 :

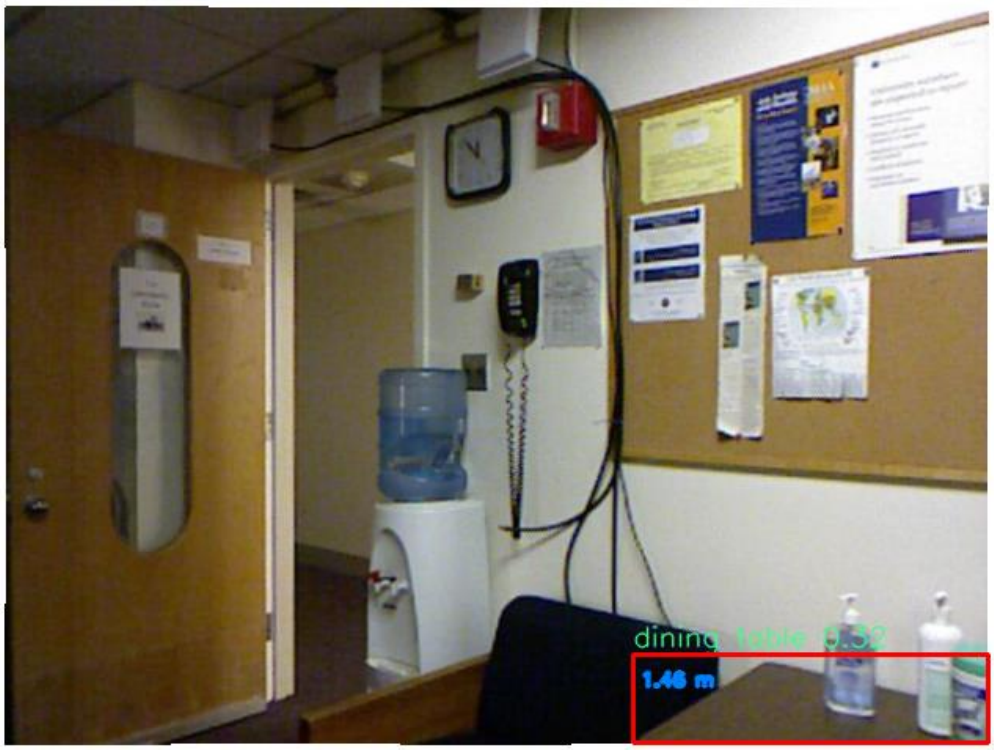
حداکثر فاصله ۳ متر:

حداکثر فاصله ۲ متر:

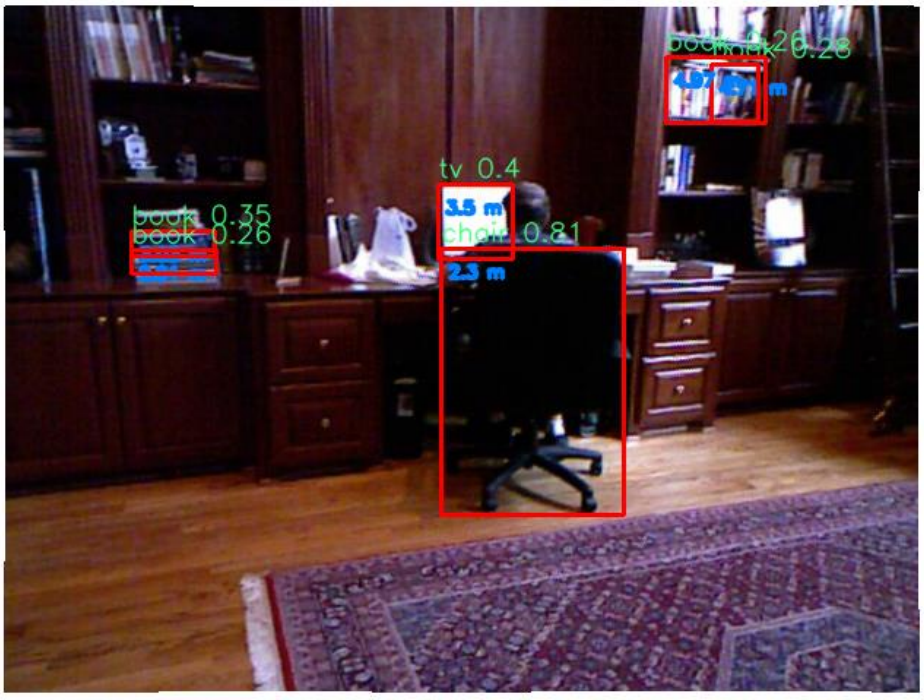
حداکثر فاصله ۴ متر:



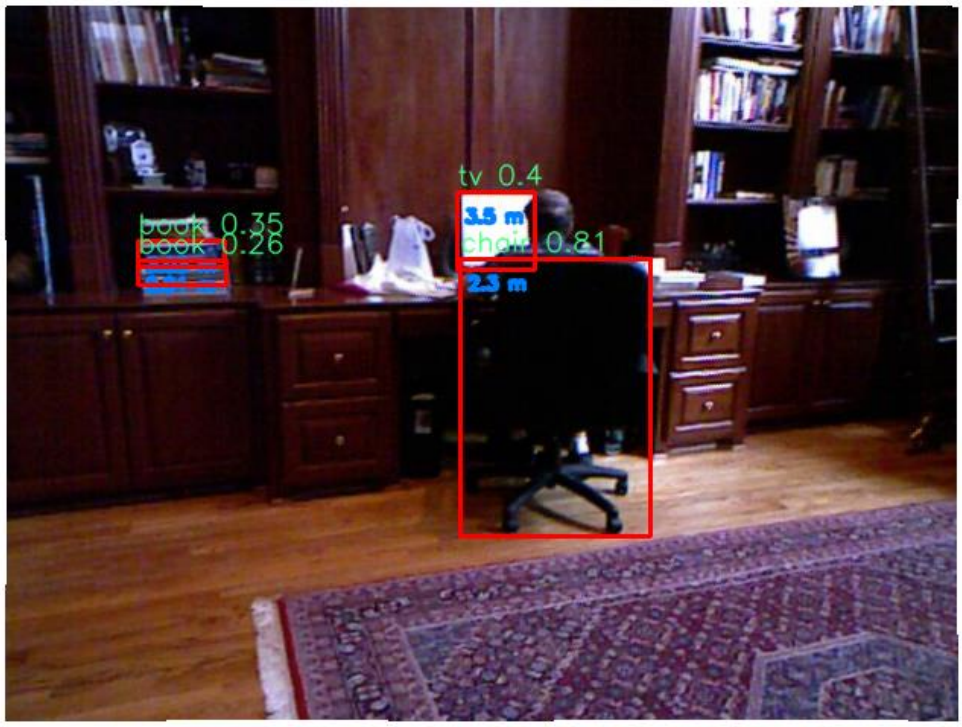
حداکثر فاصله ۱,۵ متر:



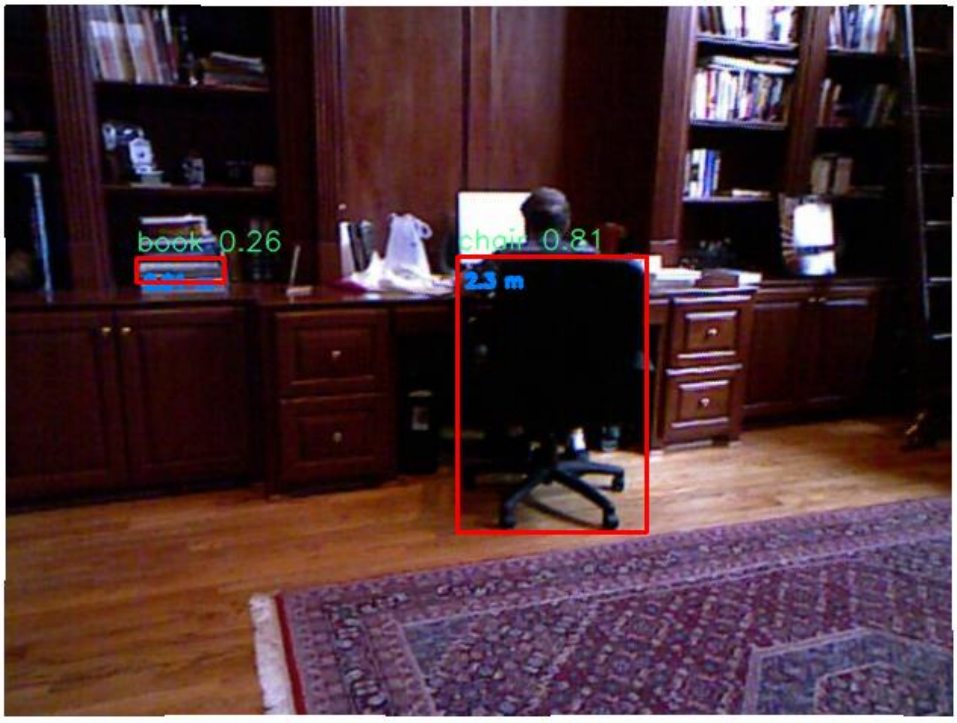
حداکثر فاصله ۵ متر:



حداکثر فاصله ۴ متر:



حداکثر فاصله ۳ متر:

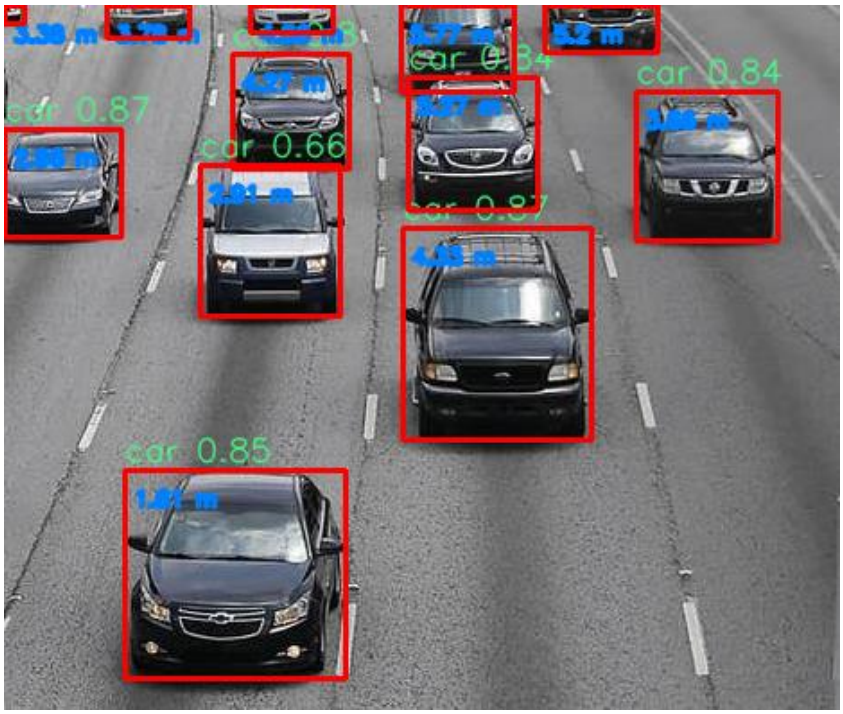


حداکثر فاصله ۴ متر:

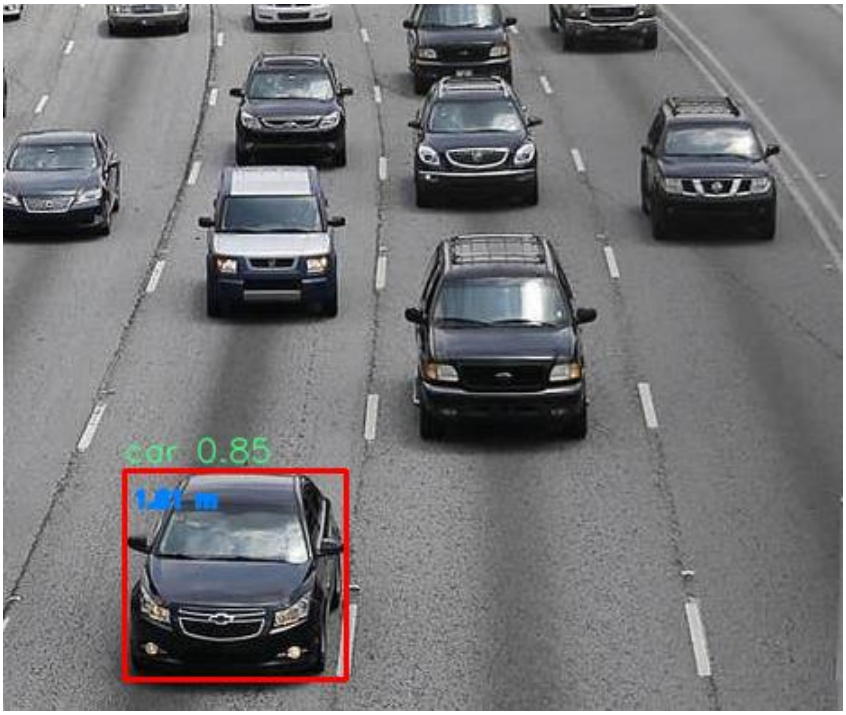
حداکثر فاصله ۳ متر:

تصاویر اینترنتی:

حداکثر فاصله ۶ متر:

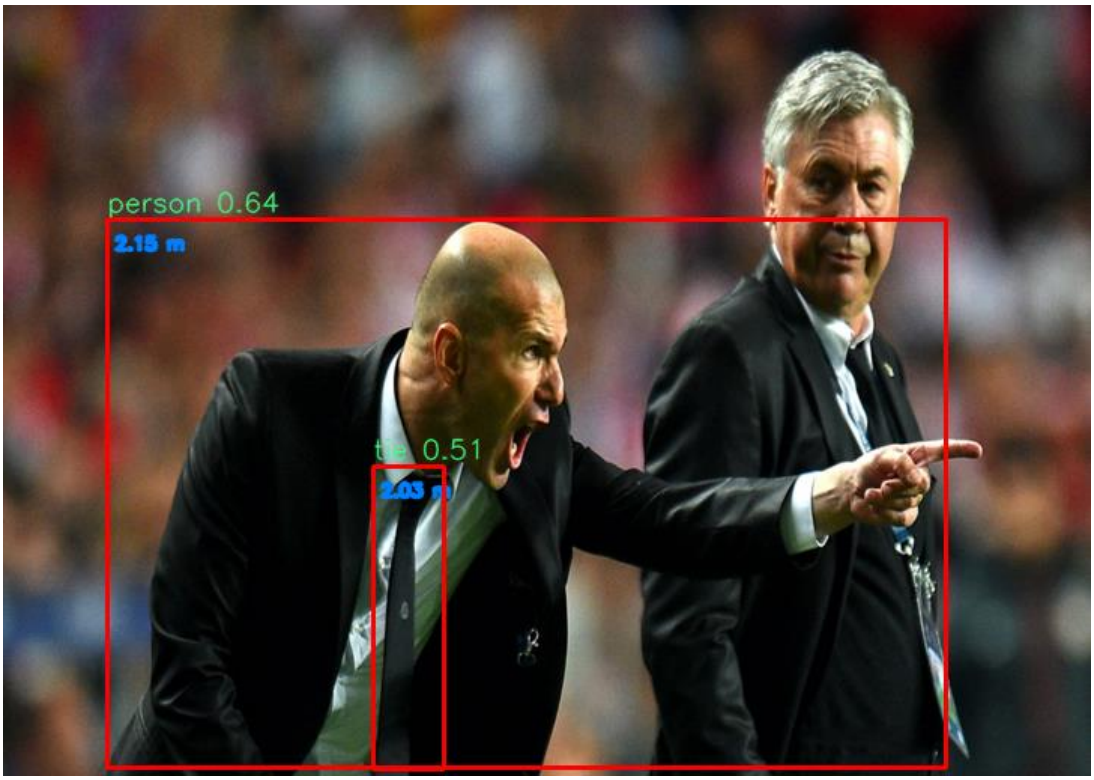


حداکثر فاصله ۲ متر:



حداکثر فاصله ۴ متر:

حداکثر فاصله ۳ متر:



همان طور که مشاهده می نمایید شبکه نهایی عملکرد خوبی دارد با اینکه آموزش هر دو شبکه از دادگان محدود استفاده شده است. این نشان می دهد دقت تئوری ۰,۷۳۷ خلاف واقع نبوده. البته باید توجه کرد که معیار $\mathrm{MmAP}\&\delta_2$ برایند دقت عمق سنجی طبقه بنده و باندینگ باکس درست پیدا کردن است که می تواند مسئولیت سنگینی بر دوش داشته باشد. می توانستیم به غیر از میانگین از توابع دیگر هم استفاده کرد ولی به نظر می رسد میانگین بهتر از بقیه است. زیرا این دو شبکه کاملا مستقل از هم هستند و حتی دادگان آموزش آن‌ها نیز با یکدیگر فرق دارد در نتیجه می توان برای دقت هر کدام وزن برابر در نظر گرفت.

نکات نهایی پروژه:

- این پروژه دارای دو فایل کد به نام depth-estimation.ipynb برای آموزش و ارزیابی شبکه معرفی شده تخمین عمق و DL_aplication_Project.ipynb به عنوان راه اندازی شبکه تشخیص اشیا YOLO و تشخیص عمق، استفاده از آن‌ها به صورت شبکه نهایی برای تشخیص اشیا و عمق آن‌ها در مسافت مشخص و درست کردن تابع detection برای کاربرد راحت تر، است.
- مشارکت در این پروژه به صورت برابر از طرف هر دو عضو بوده است.
- لینگ دادگان NYU Depth Dataset V2، ضرایب شبکه‌ی پیش آموزش mobilenet و ضرایب شبکه تخمین عمق (که در فایل depth-estimation.ipynb آموزش دیده است) در زیر به ترتیب آمده است. تا در صورت نیاز استفاده نمایید.

http://horatio.cs.nyu.edu/mit/silberman/nyu_depth_v2/nyu_depth_v2_labeled.mat

https://drive.google.com/file/d/1MUlAe5L_qm0P0f5KGmmgHnA4GZDyOqkB/view?usp=sharing

https://drive.google.com/file/d/1IzQ7tiAuDKLgoYG8-HCj8nnMVk_DBxOy/view?usp=sharing