با نام خدا

درس یادگیری عمیق

پروژه : تشخیص عمق اشیاء

سروش نورزاد (۹۹۲۰۵۳۷۲) - ارشد الکترونیک دیجیتال

ابوالفضل فلاح پاکدامن (۹۹۲۰۵۳۲۶) - ارشد الکترونیک دیجیتال

بهمن ماه ۱۴۰۰

دانشگاه صنعتی شریف

نکات عمومی در خصوص پروژه :

- پاسخ پیاده سازی عملی به سه شکلِ نوت بوکِ ipynb برای اجرا روی Google Colab ، یک فایل py برای اجرا به شکل یک برنامه ی مستقل پایتون و یک فایلِ py به شکل اجرای گرافیکی از طریقِ Qt آماده شده است.
- نسخه ی سوم پایتون و نسخه ی ۲.۷ از کتابخانه ی تنسورفلو مورد استفاده قرار گرفته است.
- برای اجرای برنامه های py، نیاز به موجود بودنِ ۴ فایلِ اولیه است. فایل وزن های شبکه ی YOLOv3 که برای Detection استفاده شده (حدود ۲۴۰ مگابایت) به همراه فایل های کانفیگ و کلاس های آن (مجموعا کمتر از ۱۰ کیلوبایت) و در نهایت یک فایل estimator_model.h5 که نتیجه ی آموزش مدل Depth Estimator است. این مدل را می توان با استفاده از فایل نوت بوک آموزش داده و به خاطر حجیم بودن، پس از انتقال روی Google Drive دانلود کرد. این کار از قبل انجام شده و فایل مدل به شکل ذخیره شده در فرمت zip روی سرور Picofile آپلود شده و به شکل یک لینک در دسترس است (با حجم ۳۹۴ مگابایت). برنامه های py برای اجرا، نیاز به این فایل ها دارند و باید این فایل ها را در پوشه ی resources خود داشته باشند. فرآیند دانلود این فایل ها به شکل خودکار پس از شروع شبیه سازی و از طریق دسترسی به اینترنت انجام خواهد شد.
- فایل نوت بوک از سه بخش اساسی تشکیل می شود. بخش ابتدایی که به بررسی ماهیت دیتابیس NYU V2 می پردازد و تصاویر مورد نیاز در بخش های Detection و Estimation را برای ما تولید می کند. بخش بعدی بخش Depth Estimation است که در آن به آموزش مدل اتوانکودر می پردازیم و خروجی آن را ذخیره می کنیم و بخش آخر که در آن به استفاده از مدل YOLO v3 برای Detection می پردازیم و همزمان از تصاویر تولید شده در بخش دیتابیس و مدل آموزش دیده شده در بخش Estimation نیز بهره می بریم تا خروجی نهایی شبکه را تولید کنیم. برای اجرای بخش Estimation بهتر است که پس از تولید تصاویر دیتابیس، Runtime را یک بار Restart کنیم تا به مشکل فضای رم نخوریم. همچنین در صورت تولید شدن فایل های لازم برای Detection می توانیم این بخش را پس از Restart کردن Runtime اجرا کنیم.
- برای اجرای فایل py با خروجی غیر گرافیکی، لازم است که در دایرکتوری pics پروژه یک فایل test1.jpg قرار داده شود.
- آدرس لینک گیت هاب (پس از پایان مهلت ارسال پروژه به عمومی تغییر خواهد یافت): لینک

توضیحات ابتدایی:

توضیحات عمومی دیتاست NYU-Depth-V2:

این دیتاست از تصاویری که از محیط های بسته و با استفاده از کینکت مایکروسافت که توانایی تصویر برداری سه بعدی را داراست تصویر برداری شده است تشکیل شده. کینکت می تواند هم تصویر RGB و هم تصویر نمایش دهنده ی عمق را ذخیره کند. این دیتاست ۱۴۴۹ جفت تصویر RGBD (تصویر RGB و تصویر نمایش دهنده ی عمق آن D) از ۴۶۴ صحنه ی متفاوت از سه شهر متفاوت را دارا می باشد. در این دیتاست، از روش لیبل گذاری بر پیکسل بهره گرفته شده است که با کمک سامانه ی خدماتی Amazon Mechanical Turk انجام گرفته است. هر کدام از اشیاء تصویر با یک کلاس و شماره ی نمونه (instance) آن لیبل گذاری شده است. نحوه ی شماره گذاری هر نمونه به این شکل است که اگر در هر تصویری، چند نمونه از یک کلاس موجود باشد، شماره نمونه های متفاوتی به نمونه ها تخصیص داده می شود.

دیتاست شامل ۳۶۰۶۴ شی مجزا از یکدیگر است که در ۸۹۴ دسته ی متفاوت (label) جای گرفته اند. برای تمام ۱۴۴۹ تصویر موجود در دیتابیس بردار اتصال (Support annotations) به طور دستی اضافه شده است. هر بردار اتصال شامل یک مجموعه ی سه تایی از Ri (شماره (ID) ناحیه (Region) ی شی حمایت شده) ، Rj (شماره (ID) ناحیه (Region) ی شی حمایت کننده) و نوع بردار است. بردار انواع متفاوتی را داراست که به طور مثال یکی حمایت از پایین به بالا (حمایت شده روی حمایت کننده قرار می گیرد. مانند فنجان روی میز) و یکی حمایت از پشت به جلو است (حمایت کننده از نقطه ای دورتر به حمایت شونده که نزدیک تر است وصل شده است، مانند قاب روی دیوار). در تصویر روبرو این موضوع قابل مشاهده است.

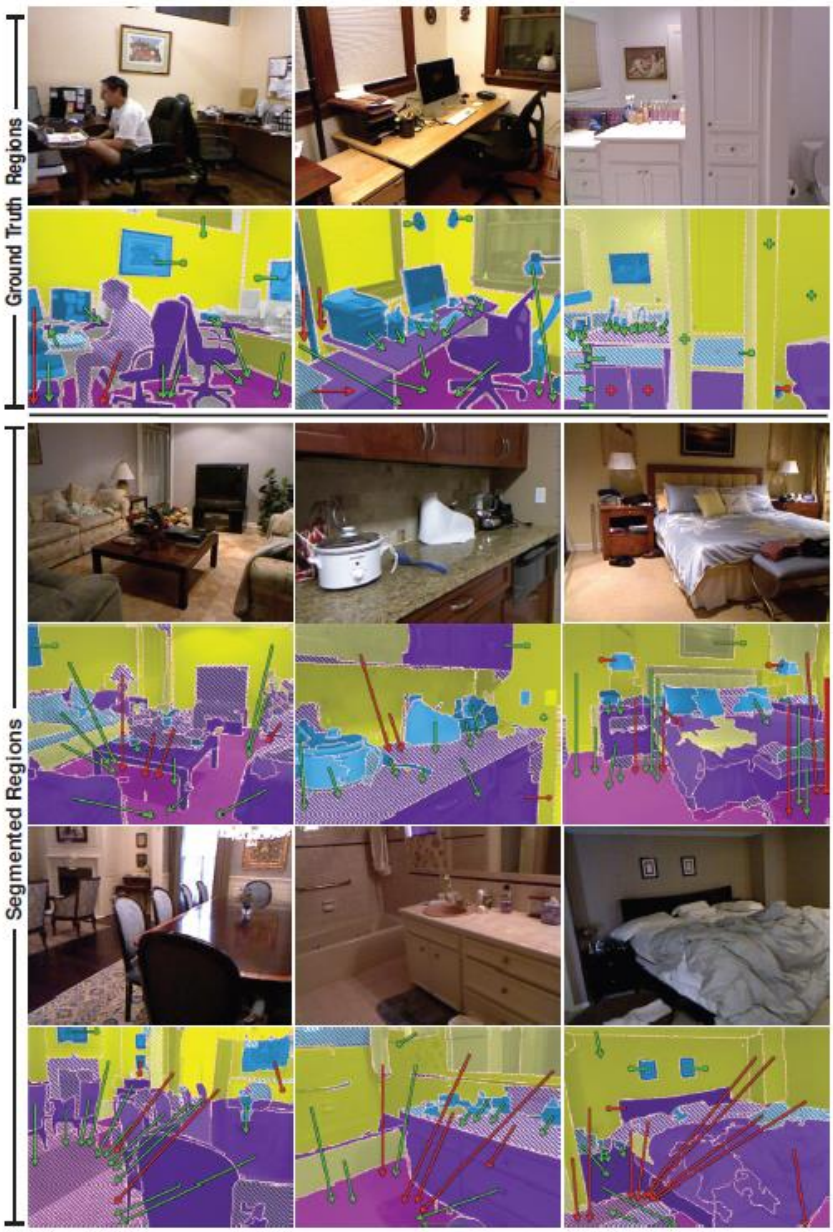
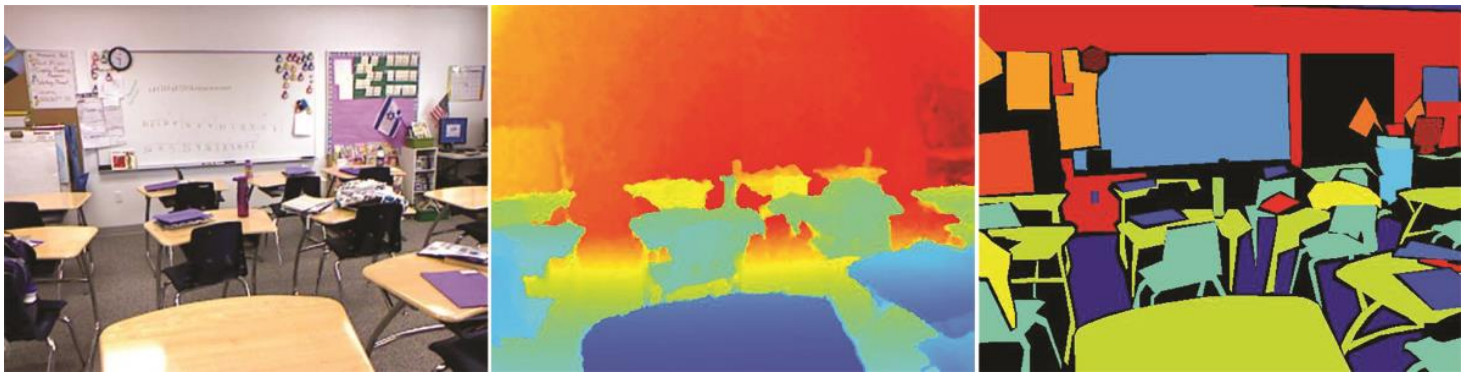


Fig. 7. Examples of support and structure class inference with the LP solution. → : support from below, —o : support from behind, + : support from hidden region. Correct support predictions in green, incorrect in red. Ground in pink, Furniture in Purple, Props in Blue, Structure in Yellow, Grey indicates missing structure class label. Incorrect structure predictions are striped.

مجموعه فایل های ارائه شده شامل چند بخش است:

- دیتاست Labeled: یک زیر مجموعه از تمامی داده های موجود است که با لیبل های چند کلاسه و سایر اطلاعاتش همراه شده است. در این داده ها پیش پردازش های لازم نیز به منظور امکان کار کردن با داده ها صورت پذیرفته است.
- فایل های Raw: تصاویر خام RGB و تصاویر Depth و Accelerometer متناظر با تصویر RGB که توسط Kinect مایکروسافت ضبط شده اند.
- فایل های Toolbox: ابزارهای لازم به جهت کار کردن با داده ها در متلب.

ما در این پروژه از دیتاست دارای لیبل بهره خواهیم گرفت. چرا که برای آموزش مدل ها نیاز به آموزش با ناظر داریم. در این دیتاست، تصاویر و اطلاعات آنها به صورت داده هایی ذخیره شده اند. تصاویر ذخیره شده، شامل خود تصاویر RGB ، تصاویر دارای مشخصه ی عمق تصویر و تصاویر دارای کلاسه بندی می باشند. نمونه ای از این تصاویر را می توان در شکل زیر مشاهده کرد:



دیتاست ذخیره شده، شامل موارد زیر است (N تعداد داده ها، H ارتفاع تصاویر و W عرض تصاویر محسوب می شوند):

- accelData: ماتریسی N*4 از مقادیر شتاب در زمان گرفته شدن تصاویر. هر کدام از N مقدار این داده ها شامل چهار پارامتر غلتش (roll) ، نوسان (yaw) ، گام (pitch) و زاویه ی کج شدن (tilt angle) دستگاه می باشند.
- depths: ماتریسی N*H*W که نمایش دهنده عمق N تصویر موجود در دیتاست می باشند.
- Images: ماتریسی N*3*H*W که همانطور که مشخص است، سه کانال RGB تصاویر اصلی را در بر دارد.
- instances: ماتریسی N*H*W که مختصه های نمونه های موجود در تصاویر را به همراه دارد. می توان از ابزار get_instance_masks.m نوشته شده در متلب برای بیرون کشیدن داده های موجود در این بخش استفاده کرد.
- labels: ماتریسی N*H*W که ماسک های لیبل ها را نمایش می دهند. لیبل ها از ۱ تا C هستند که C تعداد کل کلاس های موجود می باشند. لیبل صفر به منزله ی لیبل نخورده بودنِ پیکسل است.
- names: آرایه ای C تایی که نام انگلیسی تمام لیبل ها به ترتیب در آن قرار دارد.
- namesTolds: نگاشتی از نام های انگلیسی کلاس ها به ID کلاس ها (تعداد C جفتِ ۲ تایی key ، value).
- rawDepths: ماتریسی N*H*W که نمایشگر نگاشت "تصاویر خام نمایشگرِ عمق" هستند. این نگاشت ها عمق تصاویر را مانند بخش depths نشان می دهند، با این تفاوت که در اینجا، بخش های از دست رفته و تشخیص داده نشده، بهبود پیدا نکرده اند و پیش پردازش های لازم روی آنها انجام نگرفته است. به طور مثال هنوز غیرخطی بودن تصاویر دریافتی از کینکت در این داده ها، حذف نشده است.

- rawDepthFileNames: ماتریسی N*1 که نام تصاویر موجود در دیتاست Raw که به منظور تولید داده ی متناظر از آن بهره گرفته شده است را دارا می باشد.
- rawRgbFileNames: ماتریسی N*1 که نام تصاویر اصلی موجود در دیتاست Raw را دارا می باشد.
- scenes: ماتریسی N*1 که نام صحنه ای که تصویر از آن گرفته شده است را دارا می باشد.
- sceneTypes: ماترسی N*1 که نوع صحنه ای که تصویر از آن گرفته شده است را دارا می باشد.

در روش پیاده سازی شده ی مقاله ای که به همراه دیتاست ارائه شده، همانگونه که در تصویر زیر نیز قابل مشاهده است، ابتدا ساختار تصویر سه بعدی مورد نظر را پردازش کرده و سپس تصویر را به اشیا مجزایی که در ارتباط با یکدیگر هستند تقسیم کرده و ارتباط آنها با یکدیگر مشخص شده است. کشف بعضی از این ارتباطات ، مانند جهت و عمق قرارگیری سقف و دیوار ها، آسان و کشف بعضی مانند تشخیص هر شی و قطعه بندی تصویر می تواند کار سختی باشد. بنابراین از نگرش عمق محور (Depth cues) به منظور مرتفع کردن چالش های هندسی بهره گرفته شده است که باعث می شود تصویر تبدیل به ساختاری با جزئیات بیشتر شود. پس از انجام این عملیات و به دست آوردن ساختار عمق محور، می توان با سهولت بیشتری، اجزای تصویر را از یکدیگر جدا ساخته و رابطه ی اجزا با یکدیگر را نیز بهتر شناسایی کرد. یکی از ابتکارات موجود در مقاله، دسته بندی کلاس ها به کلاس های ساختاری (Structural Classes) است. با استفاده از این روش می توان نقش فیزیکی هر عنصر در تصویر را به خوبی شناسایی کرد. دسته های ساختاری را می توان شامل مواردی چون زمین (Ground)، ساختارهای دائمی (Permanent structures) مانند دیوار ها، سقف ها و ستون ها، وسایل بزرگ (large furniture)، مانند میزها، پیشخوان ها و کمدها و اثاثیه (props) که وسائل قابل حرکت دادن هستند، دانست.

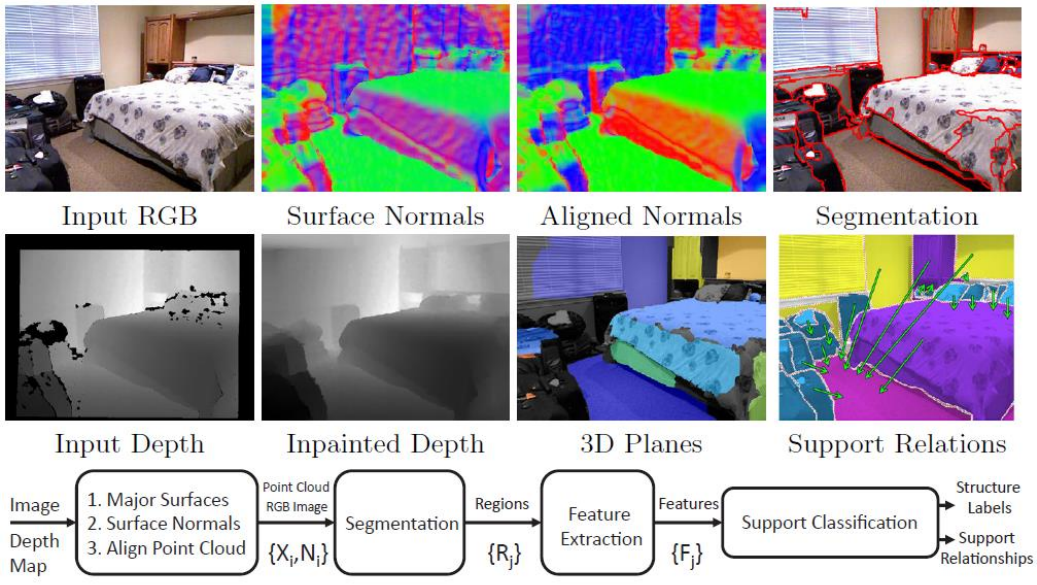


Fig. 1. Overview of algorithm. Our algorithm flows from left to right. Given an input image with raw and inpainted depth maps, we compute surface normals and align them to the room by finding three dominant orthogonal directions. We then fit planes to the points using RANSAC and segment them based on depth and color gradients. Given the 3D scene structure and initial estimates of physical support, we then create a hierarchical segmentation and infer the support structure. In the surface normal images, the absolute value of the three normal directions is stored in the R, G, and B channels. The 3D planes are indicated by separate colors. Segmentation is indicated by red boundaries. Arrows point from the supported object to the surface that supports it.

برای آنکه بتوان رابطه های بخش های متفاوت با یکدیگر را شناسایی کرد، در این مقاله روشی ارائه شده است که قیدهای فیزیکی را با مقدمات آماری ممکن (رابطه های غالب اشیا در جهان پیرامون) پیوند بزند. این روش روی تصاویر شلوغ و واقعی با المان های بسیار زیاد نیز جواب داده است. در این گونه ی تصاویر، رابطه های موجود بین بخش های متفاوت در تصویر به خوبی قابل مشاهده نیست و باید به گونه ای استنتاج شود. از دیگر مشکلات موجود در تصاویر واقعی، تغییرات قابل توجه در عمق های موجود در تصویر است. علاوه بر مشکلاتی که در تصاویری با موقعیت های مکانی گسترده دارند موجود است، در تصاویری که به بخشی از یک محیط کوچک نیز اشاره می کنند، ممکن است به چالش هایی نظیر عدم توانایی در تشخیص کف یا سقف محیط نیز بر بخوریم که با روش موجود در مقاله ارائه شده با دیتاست، این چالش نیز بر طرف شده است.

مدل سازی تصاویر داخل ساختمانی:

از آنجایی که فضای درون اتاق ها معمولا دارای خط کشی های عمودی-افقی زیادی هستند و به میزان زیادی، شیار های اینگونه در تصویر وجود دارد، یکی از اولین کارهایی که در عملیات انجام داده اند، تبدیل عملیات به یک مسئله ی alignment و segmentation بوده است. برای این کار ابتدا بردار نرمال های سطوح را با استفاده از مشاهدات موجود در تصویر عمق به دست آورده اند و سپس با استفاده از آنها و همچنین با توجه به خطوط مستیم موجود در تصویر ، شاخص ترین سه جهت اصلی تصویر را محاسبه کرده اند و پس از آن مختصات سه بعدی را به گونه ای در نظر گرفته اند که در تناسب با جهت های اصلی به دست آمده قرار گیرد. سپس صفحات سه بعدی را با استفاده از RANSAC به مختصات سه بعدی تبدیل کرده اند و نهایتا بخش های قابل مشاهده ی تصویر را که نمایانگر این صفحات اصلی می باشند را به عنوان یکی از آن صفحات سه بعدی یا پس زمینه ی تصویر در نظر گرفته اند (با استفاده از Graph cuts روی بردارهای نرمال، نقاط سه بعدی و گرادیان های RGB). در تصویر زیر می توان چند مثال از این عملیات را مشاهده کرد.

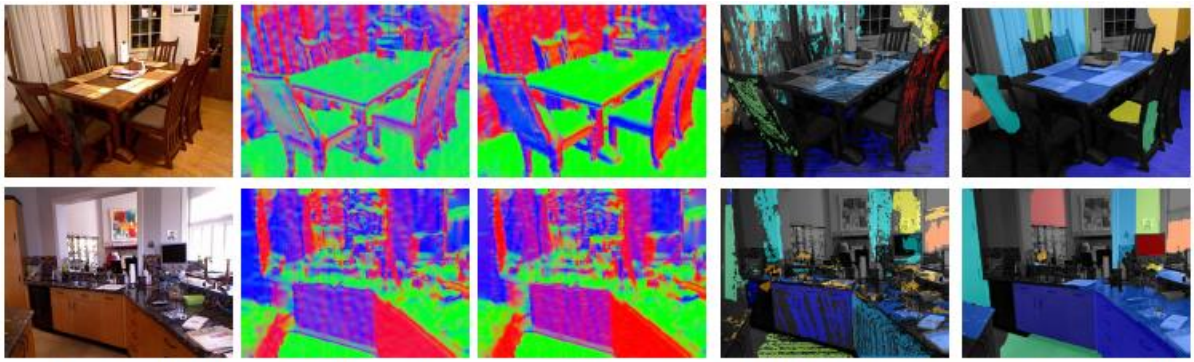


Fig. 2. Scene Structure Estimation. Given an input image (a), we compute surface normals (b) and align the normals (c) to the room. We then use RANSAC to generate several plane candidates which are sorted by number of inliers (d). Finally, we segment the visible portions of the planes using graph cuts (e). Top row: a typical indoor scene with a rectangular layout. Bottom row: an scene with many oblique angles; floor orientation is correctly recovered.

توضیحات در مورد بخش های متفاوت مدل پیاده سازی شده:

هدف در این پروژه، پیاده سازی روشی ست که بتوان با استفاده از آن تصویر ورودی را گرفته و نه تنها اشیا موجود در تصویر را تشخیص داد، بلکه، حتی فاصله ی آنها تا دوربین را نیز برآورد کرد. برای آنکه بتوانیم چنین هدفی را پیاده سازی کنیم به دو بخش متفاوت نیاز خواهیم داشت. یک بخش مدلی ست که بتواند اشیا متفاوت در تصویر را شناسایی کند (Object detection) و بخش دیگر مدلی ست که بتواند فاصله ی اجسام در تصویر را مشخص سازد (Depth Estimation). این دو بخش می توانند به طور پشت سر هم یا به طور همزمان در شبکه کار کنند و خروجی مورد نظر ما را تولید کنند. اگر بتوانیم چنین مدلی را طراحی کنیم، می توانیم از این مدل در کاربردهای متفاوتی بهره مند شویم که یکی از آنها می تواند توانمند سازی سامانه های خودروهای بدون سرنشین باشد. در این بخش، ابتدا به توضیحاتی در خصوص انواع مدل های مطروحه خواهیم پرداخت و سپس در خصوص نحوه ی پیاده سازی این مدل ها صحبت خواهیم کرد.

مدل های تشخیص دهنده ی اشیا:

در سیستم‌های تشخیص اشیا، هدف دسته‌بندی اشیاء موجود در تصویر، در مجموعه‌ای از کلاس‌های از پیش تعیین شده است. هر کدام از کلاس‌های تعریف شده، مجموعه‌ای از ویژگی‌های مختص به خود دارند که به سیستم در دسته‌بندی اشیاء متعلق به این کلاس‌ها کمک می‌کنند. به عنوان مثال، شکل هندسی دایره‌ها گرد است. بنابراین، هنگامی که سیستم به دنبال تشخیص دایره در ویدئو است، اشیائی که در فاصله خاصی از یک نقطه (مثلا مرکز) قرار دارند، جستجو می‌شوند. به طور مشابه، زمانی که سیستم به دنبال تشخیص مربع در تصویر است، اشیائی را جستجو می‌کند که در گوشه‌ها عمود هستند و اندازه اضلاع آن‌ها با یکدیگر برابر هستند.

روش مشابهی برای شناسایی چهره در کاربردهای مختلف بینایی کامپیوتر مورد استفاده قرار می‌گیرد؛ در شناسایی چهره، نقاط یا ویژگی‌های کلیدی سطح بالا در تصویر نظیر چشم، گوش، بینی و لب‌ها شناسایی می‌شوند. این دسته از ویژگی‌ها، Landmark یا ویژگی‌های شاخص نام دارند. در شناسایی چهره، برای تشخیص اشیا (نظیر چشم) در تصویر، ویژگی‌های دیگری نظیر رنگ پوست و فاصله میان چشم‌ها نیز شناسایی و استخراج می‌شوند.

در سالیان اخیر مهم‌ترین روش‌های تشخیص اشیا، معمولا رویکردهای مبتنی بر یادگیری دارند که خود به دو دسته تقسیم می شوند: یادگیری ماشین (شبکه های کم عمق) و یادگیری عمیق. در رویکردهای مبتنی بر یادگیری ماشین، بسیار حیاتی است که ابتدا ویژگی‌های مرتبط با اشیاء موجود در تصویر، با استفاده از روش‌های خاصی نظیر روش‌های زیر استخراج شوند:

- چارچوب تشخیص اشیا Viola–Jones مبتنی بر ویژگی‌های Haar
- روش‌های تبدیل ویژگی مستقل از ابعاد (Scale-Invariant Feature Transform | SIFT)
- ویژگی‌های هیستوگرام گرادیان های جهت دار (Histogram of Oriented Gradients)

سپس در مرحله بعد، از یکی روش‌های شناخته شده یادگیری ماشین، نظیر SVM استفاده می‌شود تا اشیاء در کلاس‌های از پیش تعیین شده دسته‌بندی شوند. در سویی دیگر، مدل‌های یادگیری عمیق امکان تشخیص اشیا را از روش های دیگری برای محققان فراهم می‌کنند. چنین روش‌هایی از این جهت حائز اهمیت هستند که قابلیت تشخیص اشیا، بدون تعریف صریح ویژگی‌های مرتبط

با هر کدام از کلاس‌های تعریف شده از اشیاء را برای سیستم پدید می آورند. این دسته از مدل‌های تشخیص اشیا، معمولا مبتنی بر شبکه های CNN هستند. مهم‌ترین سیستم‌های تشخیص اشیا مبتنی بر یادگیری عمیق عبارتند از:

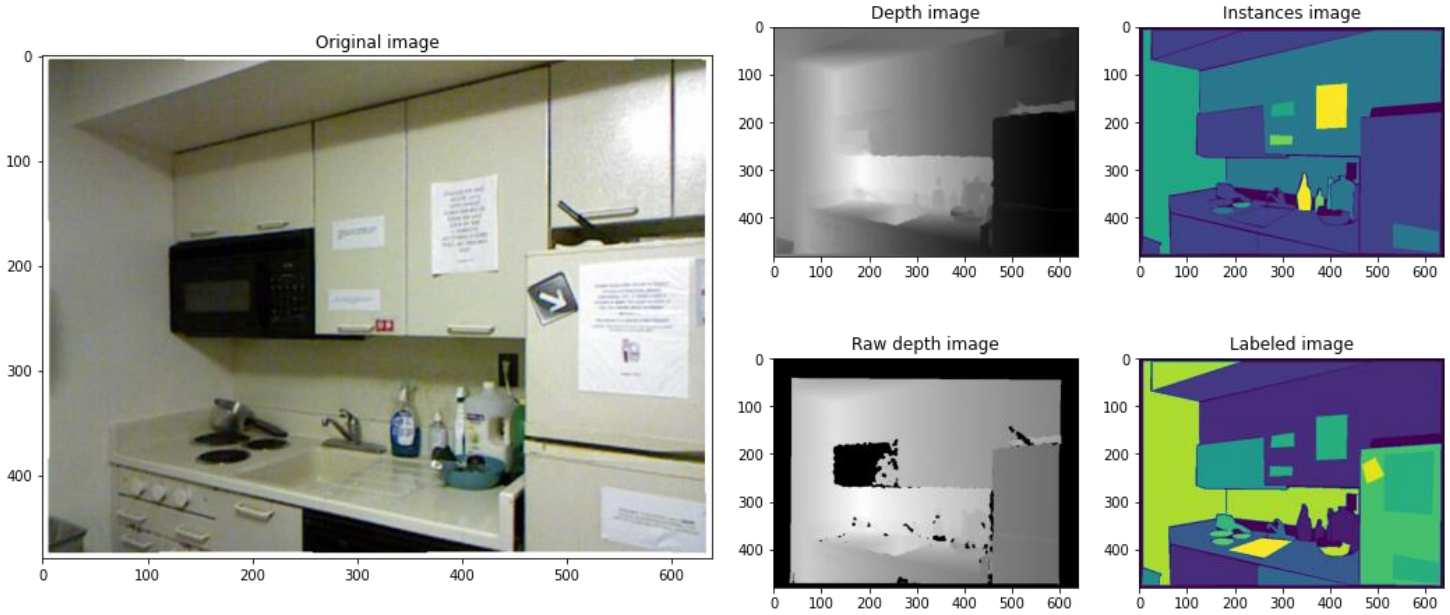
- روش‌هایRegion Proposals نظیرR-CNN ، Fast R-CNN وFaster R-CNN
- روش Single Shot MultiBox Detector یا SSD
- روش مطرح You Only Look Once و یا به اختصار YOLO

سیستم‌های تشخیص اشیا از مدل‌های دسته‌بندی تصاویر استفاده می‌کنند تا مشخص کنند چه چیزهایی در یک تصویر و در کجای آن قرار دارند. چنین کاربردهایی از سیستم‌های بینایی کامپیوتر، از طریق استفاده مدل‌های یادگیری عمیق نظیر شبکه‌های عصبی پیچشی امکان‌پذیر شده‌اند. استفاده از چنین مدل‌هایی برای تشخیص اشیا به سیستم‌های بینایی کامپیوتر اجازه می‌دهند تا یک تصویر را در چندین کلاس دسته‌بندی کنند در عین آنکه وجود چندین شیء را در تصویر تشخیص می دهند.

در این میان، معمولا الگوریتم **YOLO** به عنوان الگوریتم اصلی برای پیاده‌سازی این سیستم ها در نظر گرفته می شود. دلیل انتخاب الگوریتم **YOLO**، سرعت بالا و قدرت محاسباتی آن و همچنین، وجود منابع آموزشی زیاد برای راهنمایی کاربران هنگام پیاده‌سازی این الگوریتم است. معمولا در دو حالت از این مدل استفاده می شود. یک حالت زمانی ست که بخواهیم مدل YOLO را متناسب با دیتاست در اختیار خود بازآموزش دهیم که در این موارد از Transfer learning استفاده می‌شود و لایه کانولوشنی به انتهای مدل YOLO افزوده شده و دوباره آموزش داده می‌شود تا سیستم تشخیص اشیا بتواند اشیائی را که پیش از با آن‌ها برخورد نداشته است(Unseen Objects) ، نظیر گیتار، خانه، مرد، زن، پرنده و سایر موارد را نیز تشخیص دهد. روش دوم استفاده ی مستقیم از مدل YOLO است که با کلاس هایی که با آن آموزش داده شده مورد استفاده قرار گیرد. ما در این پروژه به علت محدود بودن امکانات از حالت دوم، یعنی استفاده ی مستقیم از مدل از پیش آموزش داده شده بهره می گیریم.

نحوه ی پیاده سازی مدل:

قبل از هر کاری در انجام این پروژه نسخه ی Labeled دیتاست NYUv2 دریافت می شود و در یک دیکشنری ریخته می شود تا دسترسی آسان تری به فایلهای موجود در آن داشته باشیم. سپس به منظور داشتن دیدی بهتر نسبت به این دیتابیس از داده های متفاوت موجود در این دیتاست، یک خروجی مناسب تهیه می کنیم. داده های موجود در این دیتاست به شکل زیر هستند:



همانگونه که گفته شد، دیتابیس شامل تمام داده های مورد نیاز به جهت تحلیل می باشد. حتی داده های تولید شده ی عمق تصویر که به شکل مطلوب پردازش نشده و خام محسوب می شود نیز در دیتاست گنجانیده شده است. از این میان، ما تنها به داده های تصویر اصلی و تصویر عمق نیازمندیم. چرا که برای پیاده سازی بخش تشخیص اشیا از مدل آماده استفاده خواهیم کرد. این داده ها را به شکل تصاویری png در پوشه های train_images و test_images ذخیره خواهیم کرد. پس از ذخیره کردن تصاویر در درایو، می توانیم Runtime را Restart کنیم تا از بروز مشکل کمبود حافظه جلوگیری کنیم. چرا که تمام تصاویر را برای پردازش وارد دیکشنری کرده ایم و در حافظه نگاه داشته ایم.

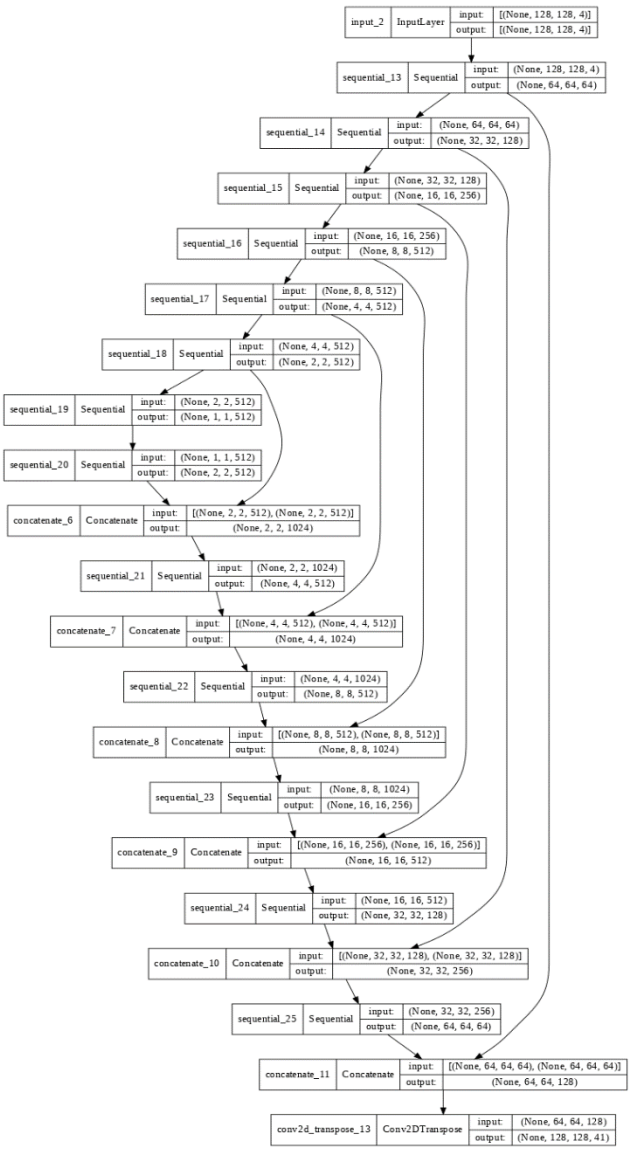
پس از Restart کردن Runtime می توانیم تصاویر را برای آموزش مدل مورد استفاده قرار دهیم. روی هر تصویر مراحل پیش‌پردازشی که شامل نورمالایز کردن و تغییر رزولوشن تصاویر به ۱۲۸ در ۱۲۸ پیکسل بود اعمال میشود. با انجام این کار می توانیم به افزایش سرعت یادگیری مدل کمک کنیم. سپس ۲۰ درصد تصاویر را به منظور Test و باقی را به منظور آموزش مدل مورد استفاده قرار می دهیم. از ۸۰ درصد باقی مانده نیز ۲۰ درصد را مجددا به عنوان داده های Validation جدا می کنیم. به عبارتی تنها از ۶۴ درصد تصاویر (حدود ۹۳۰ تصویر) به منظور آموزش مدل بهره می گیریم که این میزان ممکن است ما را با آموزش ضعیفی مواجه کند. در فرآیند آموزش از Batch size های ۶۴ تایی بهره خواهیم گرفت و از بهینه ساز Adam با نرخ یادگیری 0.0009 به منظور Optimization استفاده خواهیم کرد.

برای تابع تابع Loss از Huber استفاده می کنیم. تابع تلف از نوع MSE به منظور یادگیری هرچه بهتر نمونه های دور افتاده کاربردی است. در طرف دیگر، تابع MAE برای کم اهمیت کردنِ نمونه های دورافتاده بیشتر مورد استفاده است. اما در تصاویری که مورد پردازش ما هستند، ما با ترکیبی از این موارد مواجه هستیم. ممکن است، فاصله در بسیاری از تصاویر، به ناگهان کم و یا به ناگهان زیاد شود و با اجسامی مواجه باشیم که نسبت به سایر المان های تصاویر در فاصله ی متفاوت تری قرار گرفته باشد. بنابراین تابع MAE می تواند کاربرد خود را از دست بدهد. از طرف دیگر ممکن است با تصاویری مواجه شویم که تمام اجسام در فاصله ی نسبتا یکسانی تا دوربین قرار گرفته باشند. در چنین موردی نیز ممکن است تابع MSE توانایی ایجاد تلف مورد نیاز را از

دست بدهد. پس ما نیاز به تابع تلفی داریم که در حالت منعطف تری نسبت به این حالات عملکرد داشته باشد. یک گزینه در این شرایط، تابع Huber است. تابع هابر در حدودی بین این دو نوع تابع تلف نیز می تواند کار کند و برای دادگان ما می تواند گزینه ی مناسب تری باشد. تعریف ریاضی این تابع به شکل زیر است:

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for} |y - f(x)| \le \delta, \\ \delta\,|y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$$

برای مدل Estimation از یک شبکه ی اتوانکودر بهره خواهیم گرفت که مشابهت زیادی با شبکه U-Net دارد. از معماری U-Net به منظور آموزش مدل های مبتنی بر تصویر استفاده می شود که از عملکرد بسیار خوبی برخوردار هستند و آموزش و ارزیابی بسیار سریعی را نیز به ارمغان می آورند. در این معماری از لایه‌های کانولوشن دو بعدی استفاده می شود و می توان این مدل را تا حد زیادی شبکه ی یک مدل Fully Connected دو بعدی در نظر گرفت. یک شبکه‌ی U-net از دو بخش اصلی تشکیل شده است. یک بخش انکودر و بخش دیگر دیکودر نام دارد. گاهی این دو بخش را Up-sampler یا Down-sampler نیز نامگذاری می کنند. گرچه واژگان Up و Down می توانند توصیف گر خوبی برای بخش های یک اتوانکودر محسوب نشوند. چرا که یک اتوانکودر می تواند Over-Complete یا Under-Complete باشد و انکودر یا دیکودر آن به گونه ای عکس عمل کند.

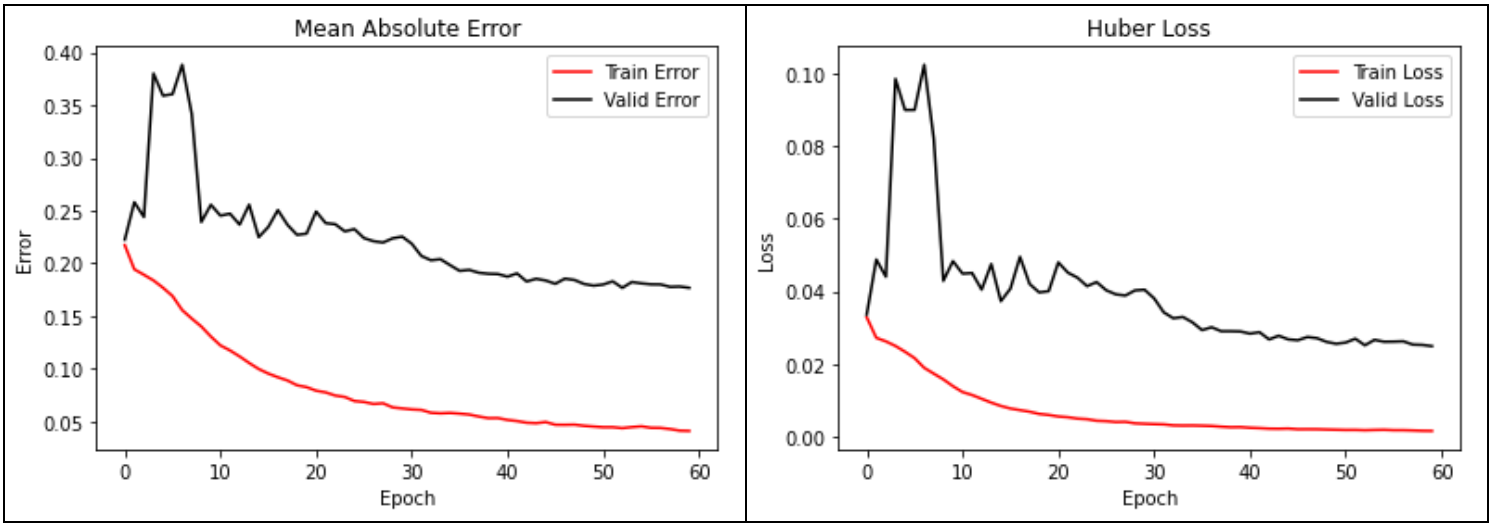


هر ماژول Up-sampler از لایه های کانولوشن، Batch normalizer و در نهایت تابع فعالساز ReLU تشکیل می شود. در ماژول های Down-sampler نیز از لایه های کانولوشن ترانهاده شده، Batch normalizer، لایه‌های Drop out و تابع فعال‌ساز Leaky-ReLU تشکیل میشود. همچنین بین هر انکودر و دیکودر اتصال‌های skip برقرار شده تا شبکه‌ی عصبی بتواند با یادگرفتن روابط جدید بین لایه‌های مختلف از overfit کردن جلوگیری کند. به طور کلی در معماری از ۷ بلاک Down-sampler و ۶ بلاک Up-sampler به علاوه یک لایه‌ی کانولوشن به عنوان لایه‌ی خروجی استفاده شده است. معماری پیاده سازی شده در شکل روبرو قابل مشاهده است.

همچنین برای آموزش مدل از روند Early Stopping با آستانه ی انتظار ۷ ایپاک استفاده شد، که در صورت overfit کردن روند آموزش مدل را به طور خودکار قطع کرده و به نقطه‌ی قبل از overfit باز میگردد. اگرچه این میزان را معمولا روی عددی بین ۱۰ تا ۱۰۰ تنظیم می کنند. لازم به ذکر است که ما آموزش مدل را قبل از بهره مندی از این خاصیت متوقف می کنیم و معمولا به ایپاک های کم اکتفا می کنیم. چرا که هدف بررسی تقریبی توانایی مدل است و نه انجام یک شبیه سازی رقابتی با نمونه های قدرتمند موجود در جهان. به منظور بررسی نهایی آموزش مدل متریک تلفِ MAE را نیز در هنگام آموزش اندازه گیری می کنیم که می توان این مقادیر را در تصویر زیر برای ایپاک ۵۰ تا ۶۰ مشاهده کرد:

```
Epoch 50/60
15/15 [==============================] - 5s 354ms/step - loss: 0.0019 - mean_absolute_error: 0.0453 - val_loss: 0.0256 - val_mean_absolute_error: 0.1789
Epoch 51/60
15/15 [==============================] - 5s 356ms/step - loss: 0.0019 - mean_absolute_error: 0.0447 - val_loss: 0.0259 - val_mean_absolute_error: 0.1799
Epoch 52/60
15/15 [==============================] - 5s 356ms/step - loss: 0.0019 - mean_absolute_error: 0.0447 - val_loss: 0.0270 - val_mean_absolute_error: 0.1830
Epoch 53/60
15/15 [==============================] - 5s 355ms/step - loss: 0.0018 - mean_absolute_error: 0.0438 - val_loss: 0.0252 - val_mean_absolute_error: 0.1768
Epoch 54/60
15/15 [==============================] - 5s 355ms/step - loss: 0.0019 - mean_absolute_error: 0.0448 - val_loss: 0.0267 - val_mean_absolute_error: 0.1823
Epoch 55/60
15/15 [==============================] - 5s 355ms/step - loss: 0.0019 - mean_absolute_error: 0.0455 - val_loss: 0.0262 - val_mean_absolute_error: 0.1810
Epoch 56/60
15/15 [==============================] - 5s 355ms/step - loss: 0.0018 - mean_absolute_error: 0.0441 - val_loss: 0.0262 - val_mean_absolute_error: 0.1801
Epoch 57/60
15/15 [==============================] - 5s 354ms/step - loss: 0.0018 - mean_absolute_error: 0.0440 - val_loss: 0.0263 - val_mean_absolute_error: 0.1800
Epoch 58/60
15/15 [==============================] - 5s 355ms/step - loss: 0.0017 - mean_absolute_error: 0.0429 - val_loss: 0.0254 - val_mean_absolute_error: 0.1777
Epoch 59/60
15/15 [==============================] - 5s 354ms/step - loss: 0.0016 - mean_absolute_error: 0.0414 - val_loss: 0.0253 - val_mean_absolute_error: 0.1780
Epoch 60/60
15/15 [==============================] - 5s 353ms/step - loss: 0.0016 - mean_absolute_error: 0.0411 - val_loss: 0.0249 - val_mean_absolute_error: 0.1769
Learning time: 325.48392605781555
```

قابل مشاهده است که متریک هابر خطا را تا حد بسیار مطلوبی به نسبت متریک MAE کاهش داده است. نمودار این دو تابع تلف را می توان در تصاویر زیر مشاهده و مقایسه کرد:



نتیجه ی بررسی چند مورد از تصاویر تست را می توان در شکل های زیر مشاهده کرد که در اکثر موارد عمق تصاویر به طور نسبی تا حد خوبی درست پیش بینی شده است.

Original image | Depth image | Estimated

Original image | Depth image | Estimated

Original image | Depth image | Estimated

Original image | Depth image | Estimated

پس از آموزش مدل و ذخیره سازی آن نوبت به تشخیص اشیا در تصویر می رسد. برای این کار از مدل از پیش آموزش داده شده ی YOLOv3 بهره خواهیم گرفت. این مدل که روی دیتاست COCO با بیش از ۳۸ هزار تصویر آموزش دیده است، توانایی بالایی در تشخیص اشیا در تصاویر را برای دسته بندی در ۸۰ کلاس داراست. در استفاده از این مدل، سعی خواهیم کرد که از تنظیمات پیش فرضِ موجود در فایل yolov3.cfg بهره ببریم. به منظور وارد کردن تصاویر به این مدل ابتدا آنها را تبدیل به فرمت blob و با سایز 416 در 416 می کنیم که متناسب با استاندارد شبکه yolov3 شود. فرمت blob فرمتی ست که در آن داده ها به شکل باینری ذخیره می شوند و معمولا از این فرمت برای ذخیره ی مالتی مدیا بهره گرفته می شود. پس از دریافت خروجی ها، امتیاز Confidency آنها را با آستانه ی اطمینان 0.3 مقایسه می کنیم تا مواردی که زیر این میزان باشند را در نظر نگیریم و تنها اشیایی که در خصوص آنها اطمینان بالاتری داریم را انتخاب کنیم (چرا که معمولا تعداد اشیا انتخاب شده در تصویر ممکن است بسیار زیاد شود و تنها بعضی از آنها از Confidency مطلوب برخوردارند).

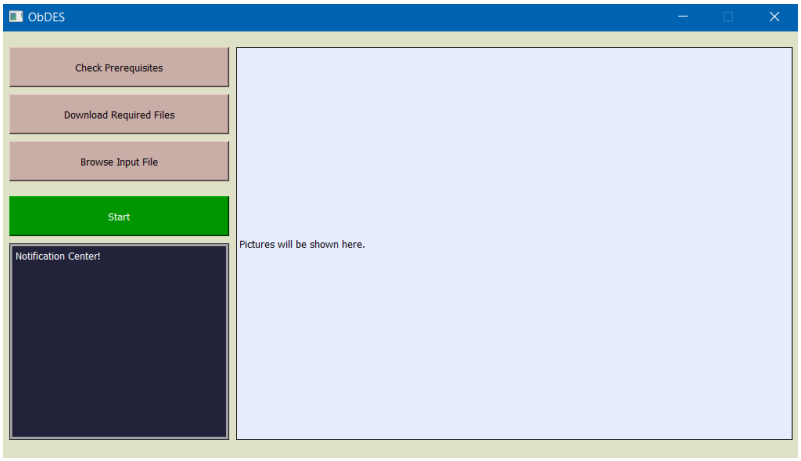
همچنین در این عمل از تکنیک Non-Maximal Suppression یا NMS نیز با آستانه ی ۰.۳ بهره خواهیم گرفت. این تکنیک به منظور کاهش تعداد باندینگ باکس هایی است که روی یک شی تشخیص داده شده جمع شده اند. نمونه ای از عملکرد این تکنیک را در تصویر زیر می توان مشاهده کرد.

نهایتا مختصه های باندینگ باکس های مهم را به منظور تخمین عمق استفاده می کنیم. به این شکل که تصویر تخمین زده شده را با تصویر دارای باندینگ باکس ها هم اندازه می کنیم و از پیکسل های درون باندینگ باکس، متوسط می گیریم و این متوسط را به عنوان متوسط فاصله ی شی از دوربین در نظر میگیریم. البته که این فاصله یک عدد نسبی بین صفر تا یک است و در آن صفر، نزدیک ترین فاصله و یک دورترین فاصله محسوب می شود. برای محاسبه ی دقیق فاصله ی هر شی تا دوربین می توان از روش های متفاوتی چون سونار بهره گرفت و از فاصله ی محاسبه شده در کنار این مقیاس بندی، تمام فاصله های موجود در تصویر را مقیاس زد. یک خروجی نمونه از عملیات انجام شده توسط ما، به شکل زیر حاصل شده است:

در تصویر فوق، تقریبا تمام اشیا با فواصل نسبتا مطلوبی تخمین زده شده اند، به جز گلدان که دورتر از مبلمان تخمین زده شده است. اما در حالت کلی، نتایج نسبتا مطلوب هستند. به طبع، اگر برای آموزش از دیتاست بزرگتری بهره می گرفتیم، مدل تخمین

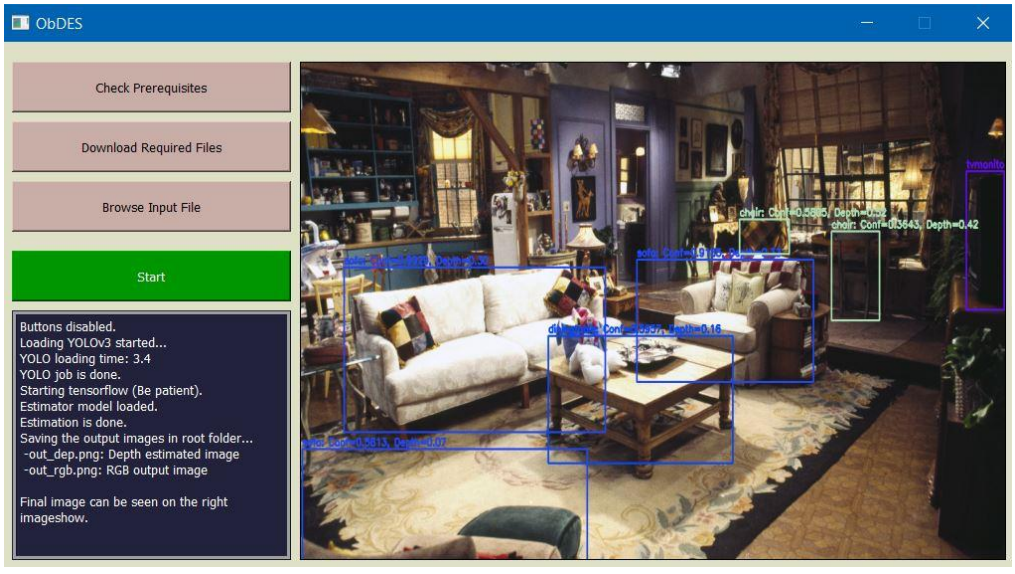
زننده ی عمق نیز با دقت بهتری می توانست عمق اشیا را تشخیص دهد. اما با توجه به زمان محدود پروژه به همین میزان از دقت قناعت می کنیم.

پس از این مرحله به ساخت نرم افزاری بر مبنای PyQt ورژن ۵ می پردازیم که بتواند پس از دانلود محتوای مورد نیاز پروژه و دریافت یک تصویر نمونه، خروجی های نهایی را برای ما تولید کند. ما اسم این نرم افزار خود را ObDES گذاشتیم. خلاصه شده ای از عبارت Object Detection and depth EStimation . ظاهر این نرم افزار پس از اجرای برنامه ObDES_GUI به صورت زیر است:



کاربرد کلید های نرم افزار به شرح زیر است:

- Check prerequisites: بررسی آنکه آیا هر چهار فایل مورد نیاز نرم افزار، دانلود شده و در پوشه ی resources موجود هست یا خیر؟ دقت کنید که این عملیات، سلامت فایل های دریافتی را بررسی نمی کند و در صورت خراب بودن فایل ها، خود فرد ناظر باید متوجه خرابی فایل ها شود.
- Download required files: در صورتی که فایلی در پوشه ی resources موجود نباشد، با این کلید می توانید آن را به طور خودکار دانلود کرده و در پوشه ی resources قرار دهید. توجه داشته باشید که این عملیات، فایل zip شده ی مدل آموزش داده شده ی Depth estimation را نیز دانلود کرده و به طور خودکار در همان مسیر extract می کند. منتها فایل zip باید به طور دستی پاک گردد.
- Browse Input File: پس از دانلود فایل های مورد نیاز و قرار گرفتن آنها در مسیر مورد نظر، می توانیم تصویر ورودی خود را با این کلید Browse کرده و به مدل تزریق کنیم.
- با کلید start فرآیند پردازش انجام می شود و در خروجی ، تصویری مانند تصویر زیر را مشاهده خواهیم کرد:



در پایان چند نمونه از تصاویر ورودی خروجی با استفاده از این برنامه را می آوریم:

نمونه ی اول:

نمونه ی دوم:

نمونه ی سوم:

با تشکر از حوصله شما

پایان

زمستان ۱۴۰۰