گزارش پروژه نهایی

محمد حسین شکوهی - ۹۹۲۰۲۵۶۷

یادگیری عمیق - دکتر عمادالدین فاطمی زاده

دانشگاه صنعتی شریف

فهرست مطالب

شبکهٔ تشخیص عمق:

به منظور یافتن شبکه و مدل مناسب برای تشخیص عمق روی دیتاست NYUv2، تحقیقات گسترده ای توسط اینجانب انجام شده است. در یکی از مقاله[1] ها، تحلیل ها و بررسی های مختلفی بر روی نحوهٔ کارکرد مدل MonoDepth انجام شده است و از آن به عنوان یکی از برترین مدل های موجود در حوزهٔ تشخیص عمق یاد شده است. در یکی دیگر از مقاله[2] ها، از مدل های MonoDepth و PSMNet به عنوان مطرح ترین مدل های تشخیص عمق یاد شده است. در یک سایت[3] معتبر، مدل های تشخیص عمق مختلف بر اساس عملکردشان روی دیتاست NYUv2 رتبه بندی شده اند. در نهایت، پس از بررسی های متعدد و سنجش میزان پیچیدگی و عملکرد مدل های مختلف، برای پیاده سازی شبکهٔ تشخیص عمق از مدل DenseDepth[4] استفاده شده است. لازم به ذکر است که در سایت ذکر شده، همانطور که در شکل ۱ مشاهده می‌شود، عملکرد مدل DenseDepth روی دیتاست NYUv2، از بین ۳۰ شبکهٔ موجود رتبهٔ ۱۱ را به خود اختصاص داده است که نوید از یک مدل قوی می‌دهد. معماری و جزئیات پیاده سازی این مدل در ادامه شرح داده خواهد شد.

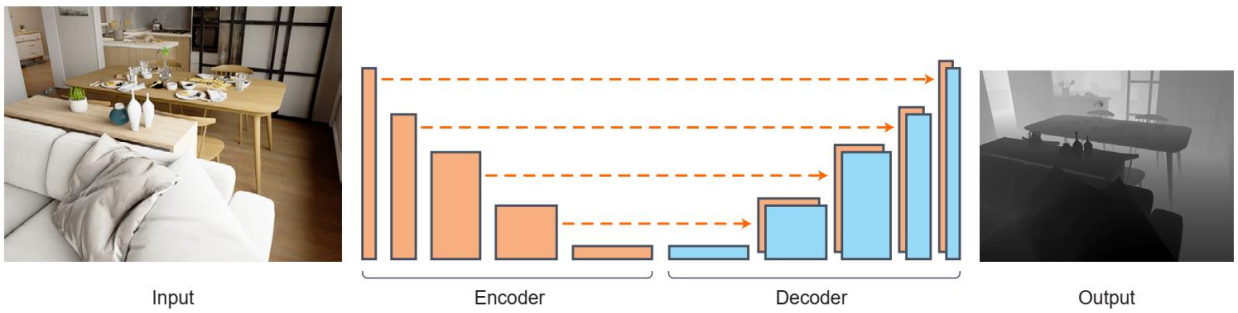
Rank	Model	absolute relative error	RMSE	log 10	Delta < 1.25	Delta < 1.25^2	Delta < 1.25^3	Extra Training Data	Paper	Code	Result	Year	Tags
1	**LeReS**	0.09			0.916			✓	Learning to Recover 3D Scene Shape from a Single Image			2020	
2	**GLPDepth**	0.098	0.344	0.042	0.915	0.988	0.997	×	Global-Local Path Networks for Monocular Depth Estimation with Vertical CutDepth			2022	
3	**AdaBins**	0.103	0.364	0.044	0.903	0.984	0.997	×	AdaBins: Depth Estimation using Adaptive Bins			2020	
4	**DPT-Hybrid**	0.110	0.357	0.045	0.904	0.988	0.998	✓	Vision Transformers for Dense Prediction			2021	
5	**BTS**	0.110	0.392	0.047	0.885	0.978	0.994	×	From Big to Small: Multi-Scale Local Planar Guidance for Monocular Depth Estimation			2019	
6	**LapDepth**	0.110	0.393	0.047	0.885	0.979	0.995	×	Monocular Depth Estimation Using Laplacian Pyramid-Based Depth Residuals			2021	
7	**VNL**	0.111	0.416	0.048	0.875	0.976	0.994	×	Enforcing geometric constraints of virtual normal for depth prediction			2019	
8	**SC-DepthV2**	0.138	0.532	0.059	0.820	0.956	0.989	×	Auto-Rectify Network for Unsupervised Indoor Depth Estimation			2021	
9	**U-Net**		0.382					×	PhaseCam3D — Learning Phase Masks for Passive Single View Depth Estimation			2019	
10	**DSN**		0.429					✓	On Deep Learning Techniques to Boost Monocular Depth Estimation for Autonomous Navigation			2020	
11	**DenseDepth**		0.465					×	High Quality Monocular Depth Estimation via Transfer Learning			2018	

شکل ۱ - رتبه بندی عملکرد شبکه های مختلف روی دیتاست NYUv2

[1] How do neural networks see depth in single images?
[2] Joint Object Detection and Depth Estimation in Multiplexed Image
[3] Monocular Depth Estimation on NYU-Depth V2
[4] High Quality Monocular Depth Estimation via Transfer Learning

مقدمه: استخراج دقیق عمق از تصاویر دارای کاربرد های متعددی در حوزه های مختلف می‌باشد که مدل های متعددی برای آن پیشنهاد شده است. مدل DenseDepth از یک شبکهٔ CNN و ایدهٔ Transfer Learning برای تولید نقشهٔ عمق تصاویر RGB با رزولوشن بالا استفاده می‌کند. در واقع، این مدل از یک معماری استاندارد Encoder-Decoder به همراه Augmentation های مختلف استفاده می‌کند تا به دقت بالایی در تشخیص عمق دست یابد. یکی از اهداف مهم این مدل، استفاده از شبکه های نسبتاً ساده برای دستیابی به دقت بالا است که سرعت یادگیری را به شدت افزایش می‌دهد. به منظور دستیابی به این هدف، مدل DenseDepth از ایدهٔ Transfer Learning استفاده می‌کند و برای بخش Encoder از شبکه های از پیش آموزش دیده با دقت بالا که در اصل برای دسته بندی تصاویر آموزش دیده اند استفاده می‌کند. نتایج اولیّه نشان می‌دهد که ایدهٔ استفاده از شبکه های Encoder ساده که در اصل برای اهداف دیگری از جمله دسته بندی تصاویر آموزش دیده اند، منجر به دست یابی به دقت بالایی می‌شود. در ادامه با ارائه Decoder مناسب و آموزش آن بر روی دیتاست NYUv2، شبکه تکمیل شده و نقشهٔ عمق تصاویر در خروجی تولید خواهد شد. در این مدل، همانطور که پیش تر اشاره شد، از یک شبکهٔ Encoder-Decoder ساده به همراه Skip Connection استفاده می‌شود.

معماری شبکه: شکل ۲ یک نمای کلی از معماری شبکهٔ مورد استفاده را نشان می‌دهد. برای بخش Encoder، از شبکهٔ DenseNet-169 که بر روی دیتاست ImageNet از پیش آموزش دیده است استفاده می‌شود تا تصویر RGB ورودی را به یک بردار ویژگی ها انکود کند. در ادامه، این بردار از لایه های Up-sampling متعدد عبور می‌کند تا نقشهٔ عمق در خروجی تولید شود. این لایهٔ های Up-sampling به همراه Skip Connection های متناظر، Decoder ما را تشکیل می‌دهند که توسط خودمان بر روی دیتاست NYUv2 آموزش خواهد دید. لازم به ذکر است که در بخش Decoder از Batch Normalization استفاده نشده است.



شکل ۲ - ما از یک شبکه‌ی ساده‌ی Encoder-Decoder به همراه Skip Connection استفاده می‌کنیم. برای بخش Encoder از شبکه‌ی از پیش آموزش دیده‌ی DenseNet-169 استفاده می‌شود. بخش Decoder از لایه های کانولوشنی متعدد استفاده می‌کند که ورودی آن ها ترکیب Up-sample شده‌ی خروجی لایه‌ی قبل و لایهٔ متناظر در Encoder می‌باشد. شبکهٔ Decoder در ادامه بر روی دیتاست NYUv2 آموزش خواهد دید.

حال می‌خواهیم کمی بیشتر وارد جزئیات این معماری شده و لایه های مختلف آن را بررسی کنیم. جدول ۱ ساختار لایه به لایه‌ی شبکه را نشان می‌دهد. بخش Encoder مبتنی بر شبکهٔ DenseNet-169 است که بر روی ImageNet آموزش دیده است. برای بخش Decoder، با یک لایه‌ی کانولوشنی 1 X 1 با همان تعداد کانال های خروجی Encoder شروع می‌کنیم. سپس به طور متوالی از بلاک های Up-sampling استفاده می‌کنیم. هر بلاک از یک لایه‌ی Bilinear up-sampling و در ادامه دو لایه‌ی کانولوشنی تشکیل شده است. در هر بلاک، ورودی اولین لایه‌ی کانولوشنی، حاصل ادغام خروجی لایه‌ی قبلی و خروجی لایه‌ی Pooling متناظر Encoder (Skip-connection) است. خروجی هر بلاک Up-sampling، به جز بلاک آخر، از تابع فعالساز

Leaky ReLU با پارامتر $\alpha = 0.2$ عبور می‌کند. ماتریس تصاویر ورودی تقسیم بر ۲۵۵ می‌شود تا محدوده ورودی بین ۰ و ۱ باشد و هیچ روش Normalization دیگری بر روی لایه‌ی ورودی استفاده نمی‌شود.

LAYER	OUTPUT	FUNCTION
INPUT	480 × 640 × 3	
CONV1	240 × 320 × 64	DenseNet CONV1
POOL1	120 × 160 × 64	DenseNet POOL1
POOL2	60 × 80 × 128	DenseNet POOL2
POOL3	30 × 40 × 256	DenseNet POOL3
…	…	…
CONV2	15 × 20 × 1664	Convolution 1 × 1 of DenseNet BLOCK4
UP1	30 × 40 × 1664	Upsample 2 × 2
CONCAT1	30 × 40 × 1920	Concatenate POOL3
UP1-CONVA	30 × 40 × 832	Convolution 3 × 3
UP1-CONVB	30 × 40 × 832	Convolution 3 × 3
UP2	60 × 80 × 832	Upsample 2 × 2
CONCAT2	60 × 80 × 960	Concatenate POOL2
UP2-CONVA	60 × 80 × 416	Convolution 3 × 3
UP2-CONVB	60 × 80 × 416	Convolution 3 × 3
UP3	120 × 160 × 416	Upsample 2 × 2
CONCAT3	120 × 160 × 480	Concatenate POOL1
UP3-CONVA	120 × 160 × 208	Convolution 3 × 3
UP3-CONVB	120 × 160 × 208	Convolution 3 × 3
UP4	240 × 320 × 208	Upsample 2 × 2
CONCAT3	240 × 320 × 272	Concatenate CONV1
UP2-CONVA	240 × 320 × 104	Convolution 3 × 3
UP2-CONVB	240 × 320 × 104	Convolution 3 × 3
CONV3	240 × 320 × 1	Convolution 3 × 3

جدول ۱ – سطر های نارنجی رنگ مربوط به Encoder (DenseNet-169) و سطر های آبی رنگ مربوط به Decoder می‌باشند.

تابع Loss: یک تابع هدف استاندارد که در مسائل تشخیص عمق استفاده می‌شود، اختلاف بین نقشه‌ی عمق اصلی و نقشه‌ی عمق خروجی شبکه می‌باشد. استفاده از تابع هدف مناسب نقش بسزایی در سرعت و عملکرد شبکه خواهد داشت. در این مدل، ما یک تابع هدف جدید تعریف می‌کنیم که علاوه بر محاسبهٔ اختلاف نقشه‌ی اصلی و نقشه‌ی خروجی شبکه، وجود ناهنجاری های فرکانس بالا در نقشه‌ی خروجی شبکه را نیز مجازات می‌کند. این تابع هدف شامل جمع وزن دار سه جمله است:

$$L(y, \hat{y}) = \lambda L_{depth}(y, \hat{y}) + L_{grad}(y, \hat{y}) + L_{SSIM}(y, \hat{y}).$$

جمله‌ی اول همان L1 loss است که روی مقادیر عمق تعریف می‌شود:

$$L_{depth}(y, \hat{y}) = \frac{1}{n}\sum_{p}^{n}|y_p - \hat{y}_p|.$$

جمله‌ی دوم L1 loss است که روی گرادیان نقشه‌ی عمق تعریف می‌شود:

$$L_{grad}(y, \hat{y}) = \frac{1}{n}\sum_{p}^{n}|\boldsymbol{g}_{\mathbf{x}}(y_p, \hat{y}_p)| + |\boldsymbol{g}_{\mathbf{y}}(y_p, \hat{y}_p)|$$

جملات g_x و g_y به ترتیب اختلاف مولفه های x و y گرادیان نقشه‌ی عمق اصلی و خروجی شبکه را نشان می‌دهند.

در نهایت، جمله‌ی سوم از SSIM (Structural Similarity) استفاده می‌کند. اخیرا نشان داده شده است که وجود این جمله در تابع هدف می‌تواند به بهبود کیفیت آموزش شبکه های تشخیص عمق کمک کند.

$$L_{SSIM}(y, \hat{y}) = \frac{1 - SSIM(y, \hat{y})}{2}.$$

Data Augmentation: همانطور که می‌دانیم Data Augmentation یکی از روش های مناسب برای جلوگیری از Over-fitting در فرآیند آموزش شبکه می‌باشد. در این مدل، دو نوع Data Augmentation متفاوت مورد استفاده قرار گرفته است:

۱- Horizontal Flipping با احتمال ۰.۵

۲- جابجا کردن کانال های رنگ تصویر ورودی (برای مثال جابجا کردن کانال قرمز و سبز) با احتمال ۰.۲۵. ثابت شده است که این روش منجر به بهبود عملکرد شبکه می‌شود.

دیتاست مورد استفاده: همانطور که پیش تر اشاره شد و مطابق دستورالعمل های دستیاران آموزشی درس، برای آموزش شبکه از دیتاست NYUv2 با ۱۴۴۹ تصویر استفاده کردم. از این عدد، ۱۳۰۴ تصویر به آموزش و ۱۴۵ تصویر به تست اختصاص داده شده است. با توجه به معماری شبکه، نقشه‌ی عمق خروجی دارای ابعاد 240 × 320 می‌باشد. به هنگام آموزش، نقشه‌ی عمق واقعی را با ضریب ۲ کوچک می‌کنیم تا با ابعاد خروجی شبکه همخوانی داشته باشد. امّا به هنگام تست، خروجی شبکه را با ضریب ۲ بزرگ می‌کنیم تا با نقشه‌ی عمق واقعی همخوانی داشته باشد. لازم به ذکر است که به هنگام تست، خروجی نهایی برابر است با میانگین خروجی شبکه برای تصویر اصلی و خروجی شبکه برای آینه شده‌ی تصویر اصلی.

جزئیات پیاده سازی: برای پیاده سازی شبکه‌ی DenseDepth از کد موجود در گیت هاب[5] که با کتابخانه‌ی TensorFlow نوشته شده است استفاده کردیم. البته از کد ذکر شده تنها به عنوان شروع و پایه ی کار استفاده شده و تغییرات فراوان و متعددی در آن اعمال شده تا برای استفاده در این پروژه مناسب گردد. برای آموزش شبکه از سرویس آنلاین Google Colab با کارت گرافیک Tesla T4 و ۱۶ گیگابایت VRAM استفاده شده است. همانطور که پیش تر اشاره شد، بخش Encoder یک شبکهٔ

[5] GitHub DenseDepth

DenseNet-169 است که از قبل روی دیتاست ImageNet آموزش دیده است. وزن های بخش Decoder به صورت رندوم مقدار دهی اولیه شده اند. برای آموزش شبکه از بهینه ساز Adam و learning rate = 0.0001 و مقادیر پارامتر $\beta_1 = 0.9$ و $\beta_2 = 0.999$ و سایز بچ برابر با ۴ استفاده شده است. فرآیند آموزش برای تعداد epoch برابر با ۲۰ انجام شد و حدود ۳۰ دقیقه به طول انجامید.

ارزیابی: برای ارزیابی دقت شبکه روی داده های تست، از ۶ معیار معتبر برای ارزیابی دقت شبکه ها استفاده شده است. این معیار ها را در شکل ۳ مشاهده می‌فرمایید:

average relative error (rel): $\frac{1}{n}\sum_p^n \frac{|y_p - \hat{y}_p|}{y}$;

root mean squared error (rms): $\sqrt{\frac{1}{n}\sum_p^n (y_p - \hat{y}_p)^2)}$;

average ($\log_{10}$) error: $\frac{1}{n}\sum_p^n |\log_{10}(y_p) - \log_{10}(\hat{y}_p)|$;

threshold accuracy (δ_i): % of y_p s.t. $\max(\frac{y_p}{\hat{y}_p}, \frac{\hat{y}_p}{y_p}) = \delta < thr$ for $thr = 1.25, 1.25^2, 1.25^3$;

شکل ۳ – معیار های مورد استفاده برای سنجش دقت شبکه

همانطور که پیش تر اشاره شد، از یک زیر مجموعه‌ی ۱۴۵ تایی از دیتاست برای انجام تست استفاده شده است. نتایج در جدول ۲ آورده شده است.

δ_1	δ_2	δ_3	rel	rms	log_{10}
0.7960	0.9606	0.9931	0.1421	0.6948	0.0616

جدول ۲ – معیار های سنجش عملکرد شبکه

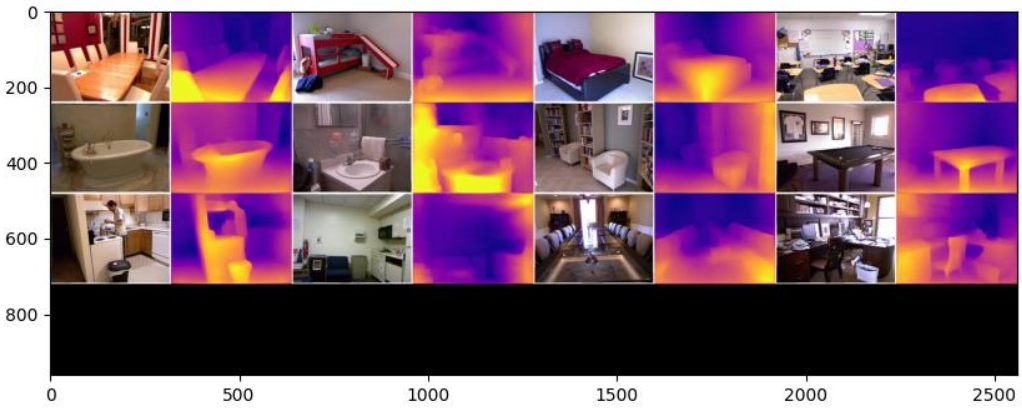
علیرغم اینکه نتایج سنجش عملکرد این شبکه که توسط اینجانب آموزش دیده است، نشان دهنده‌ی عملکرد بسیار خوب شبکه هستند، اما کمی نسبت به نتایج آورده شده در خود مقالهٔ اصلی این مدل ضعیف تر هستند. این اتفاق می‌تواند ۲ دلیل داشته باشد:

۱- در مقالهٔ اصلی از دیتاست NYUv2 با 50k تصویر استفاده شده است، حال آنکه ما برای آموزش شبکه تنها از ۱۳۰۴ عدد تصویر از این دیتاست استفاده کردیم. بنابراین منطقی است که عملکرد شبکه ی آموزش دیده توسط اینجانب کمی ضعیفتر باشد.

۲- برخی از روش های استفاده شده در مقالهٔ اصلی برای Data Augmentation را به دلیل پیچیدگی زیاد حذف کردیم.

گراف کامل شبکه در فایل model_plot.png قابل مشاهده می‌باشد.

شکل ۴ خروجی شبکه را برای برخی از تصاویر تست نشان می‌دهد.



شکل ۴ – خروجی شبکه‌ی تشخیص عمق برای برخی از داده های تست

شبکهٔ تشخیص اشیاء:

برای بخش تشخیص اشیاء مطابق پیشنهاد دستیاران آموزشی تصمیم گرفتیم از شبکه های از پیش آموزش دیده استفاده کنیم. برای مساله‌ی تشخیص اشیاء، کتابخانه های متعددی برای زبان پایتون نوشته شده اند که مدل های آماده و از پیش آموزش دیده‌ای با استفاده از انواع معماری های مشهور را در اختیار کاربر قرار می‌دهند تا بتواند بنا بر نیاز خود، یکی از آنها را انتخاب کرده و عملیات تشخیص اشیاء را انجام دهد. بنابراین جستجو های بسیار فراوانی انجام دادیم تا مدل مناسب برای تشخیص اشیاء انتخاب شود. بر این اساس، در یک سایت معتبر[6]، انواع الگوریتم ها و کتابخانه های مشهور برای تشخیص اشیاء ذکر شده و مزایا و معایب هرکدام شرح داده شده است. بر اساس کتابخانه های ذکر شده در این سایت، بر روی چندین کتابخانه‌ی که در زمینه‌ی تشخیص اشیاء وجود دارد تست های مختلفی انجام دادیم تا کتابخانهٔ مناسب برای استفاده در پروژه را انتخاب کنیم. چند نمونه از این کتابخانه ها در ذیل آورده شده است:

۱- ImageAI[7]: این کتابخانه که با استفاده از TensorFlow نوشته شده است، مجموعه‌ی بزرگی از مدل ها و معماری های از پیش آموزش دیده را در اختیار کاربر می‌گذارد تا بتواند بر اساس نیاز های خود یکی را انتخاب کرده و عملیات تشخیص اشیاء را انجام دهد. ما این کتابخانه را نصب کردیم و تست های متعددی روی آن انجام دادیم. بر اساس تست هایی که انجام دادیم، مدل های موجود در این کتابخانه، از جمله مدل های YOLO، SSD و RetinaNet از دقت نسبتاً خوبی برخوردار بودند امّا به دلیل استفاده‌ی این کتابخانه از پکیج های بسیار قدیمی که در ورژن های جدید پایتون دیگر در دسترس نیستند، این کتابخانه را انتخاب نکردیم.

۲- GluonCV[8]: این کتابخانه که با استفاده از MXNet نوشته شده است، یکی دیگر از کتابخانه های معتبر در حوزه‌ی تشخیص اشیاء می‌باشد. این کتابخانه، مدل های از پیش آموزش دیده‌ی متعددی را برای Image Classification، Object Detection و Semantic Segmentation در اختیار کاربر قرار می‌دهد. به طور خاص، این کتابخانه برای انجام Object Detection مدل های معروف از پیش آموزش دیده همچون Faster R-CNN، SSD و Yolo-v3 را در اختیار کاربر قرار می‌دهد. ما با استفاده از تک تک این مدل ها بر روی دیتاست NYUv2 تست های متعددی انجام دادیم. به دلیل آموزش دیدن مدل های این کتابخانه روی دیتاست های نه چندان مناسب، تشخیص اشیاء با استفاده از مدل های این کتابخانه بر روی دیتاست NYUv2 از عملکرد بسیار ضعیف و دقت بسیار کمی برخوردار بود. به دلیل عملکرد غیر قابل قبول این کتابخانه بر روی دیتاست NYUv2، این مدل را نیز انتخاب نکردیم.

۳- YOLOv3_TensorFlow[9]: این کتابخانه با استفاده از مدل YOLOv3 عملیات تشخیص اشیاء را انجام می‌دهد و از دقت نسبتاً خوبی برخوردار است، امّا به دلیل استفاده‌ی از ورژن های بسیار قدیمی و منسوخ شده‌ی TensorFlow، این کتابخانه را نیز انتخاب نکردیم.

۴- Detectron2[10]: این کتابخانه که توسط تیم Facebook AI Research (FAIR) توسعه داده شده است، یکی از بهترین و قوی ترین کتابخانه های موجود برای انجام عملیات تشخیص اشیاء می‌باشد که مجموعه‌ی بسیار عظیمی از مدل

[6] Object Detection Algorithms and Libraries
[7] GitHub ImageAI
[8] GitHub GluonCV
[9] GitHub YOLOv3_TensorFlow
[10] GitHub Detectron2

های از پیش آموزش دیده را برای انواع عملیات Object Detection و Semantic Segmentation در اختیار کاربر قرار می‌دهد. کار کردن با این کتابخانه که با استفاده از PyTorch نوشته شده است، به دلیل منظم بودن و Documentation های بسیار خوب، برای کاربران ساده است. مدل های این کتابخانه به دلیل استفاده از دیتاست های معتبر و قوی برای آموزش مدل ها، از جمله دیتاست های COCO و ImageNet از دقت بسیار بالایی برخوردار است. بر اساس تست هایی که با استفاده از این کتابخانه روی دیتاست NYUv2 انجام دادیم، مدل های آموزش دیده‌ی این کتابخانه قادر بودند تا با دقت بسیار بالایی عملیات Object Detection را انجام دهند. به دلیل قوی بودن تیم توسعه‌ی این کتابخانه، سادگی کار با آن، و دقت بسیار بالای آن روی دیتاست NYUv2، در نهایت این کتابخانه برای استفاده در پروژه انتخاب شد.

استفاده از کتابخانه‌ی Detectron2: کار با کتابخانه‌ی Detectron2 بسیار ساده می‌باشد. کافیست مطابق دستورالعمل های ذکر شده در صفحه‌ی گیت هاب این کتابخانه، ابتدا پکیج های مورد نیاز نصب گردند و سپس خود کتابخانه نصب شود. در ادامه، با توجه به Model Zoo[11] ارائه شده توسط توسعه دهندگان، کاربر مدل مورد نظر خود را از بین چندین مدل از پیش آموزش دیده انتخاب می‌کند، آن را دانلود کرده و در کتابخانه‌ی Detectron2 وارد می‌کند و عملیات تشخیص اشیاء را به سادگی انجام می‌دهد.

مدل مورد استفاده: تا کنون در مورد کتابخانه‌ی مورد استفاده صحبت کردیم. هر کتابخانه، شامل مدل های از پیش آموزش دیده‌ی متعددی می‌شود. بنابراین، حال نوبت به آن رسیده که مدل مناسب برای تشخیص اشیاء در این پروژه را مشخص کنیم. کتابخانه‌ی Detectron2 مدل های از پیش آموزش دیده‌ی متعددی را برای Object Detection و Instance Segmentation در اختیار کاربران قرار می‌دهد که از جمله معروف ترین آن ها می‌توان به Faster R-CNN و Mask R-CNN اشاره کرد. به دلیل برخی ملاحظات که در بخش های بعدی به آن اشاره خواهد شد، تصمیم گرفتیم تا از بین مدل های موجود در Model Zoo کتابخانه‌ی Detectron2، از مدل Mask R-CNN استفاده کنیم که عملیات Instance Segmentation انجام می‌دهد، یعنی هم عملیات Object Detection (Box Prediction) و هم عملیات Semantic Segmentation (Mask Prediction) را به صورت توامان انجام می‌دهد. این مدل بر روی دیتاست COCO آموزش دیده است و با توجه به تست های انجام شده، برای استفاده در این پروژه بسیار مناسب می‌باشد. حال می‌رسیم به قدم بعدی! کتابخانه‌ی Detectron2 برای مدل Mask R-CNN، Backbone های مختلفی را پیشنهاد داده است که در ادامه به آنها اشاره می‌شود:

۱- FPN: از یک زیرساخت ResNet+Feature Pyramid Network (FPN) به همراه لایه های conv و FC استاندارد برای تشخیص ماسک و تشخیص اشیاء استفاده می‌شود. این زیر ساخت بهترین Tradeoff را بین سرعت و دقت برقرار می‌کند بنابراین برای این پروژه این زیرساخت را انتخاب کردیم.
۲- C4: از زیرساخت ResNet conv4 به همراه لایه conv5 استفاده می‌کند.
۳- DC5: از زیرساخت ResNet conv5 به همراه لایه های conv و FC برای تشخیص ماسک و اشیاء استفاده می‌کند.

[11] Detectron2 Model Zoo and Baselines

در شکل ۵ که از Model Zoo کتابخانه برداشته شده است، جدول انواع مدل های از پیش آموزش دیده‌ی مبتنی بر Mask R-CNN را مشاهده می‌فرمایید:

COCO Instance Segmentation Baselines with Mask R-CNN

Name	lr sched	train time (s/iter)	inference time (s/im)	train mem (GB)	box AP	mask AP	model id	download
R50-C4	1x	0.584	0.110	5.2	36.8	32.2	137259246	model | metrics
R50-DC5	1x	0.471	0.076	6.5	38.3	34.2	137260150	model | metrics
R50-FPN	1x	0.261	0.043	3.4	38.6	35.2	137260431	model | metrics
R50-C4	3x	0.575	0.111	5.2	39.8	34.4	137849525	model | metrics
R50-DC5	3x	0.470	0.076	6.5	40.0	35.9	137849551	model | metrics
R50-FPN	3x	0.261	0.043	3.4	41.0	37.2	137849600	model | metrics
R101-C4	3x	0.652	0.145	6.3	42.6	36.7	138363239	model | metrics
R101-DC5	3x	0.545	0.092	7.6	41.9	37.3	138363294	model | metrics
R101-FPN	3x	0.340	0.056	4.6	42.9	38.6	138205316	model | metrics
X101-FPN	3x	0.690	0.103	7.2	44.3	39.5	139653917	model | metrics

شکل ۵ – مدل های از پیش آموزش دیده مبتنی بر Mask R-CNN برای عملیات Object Detection and Instance Segmentation

در ادامه، بر اساس مقاله‌ی[۱۲] اصلی مدل Mask R-CNN، معماری و جزئیات این مدل را بیشتر بررسی خواهیم کرد.

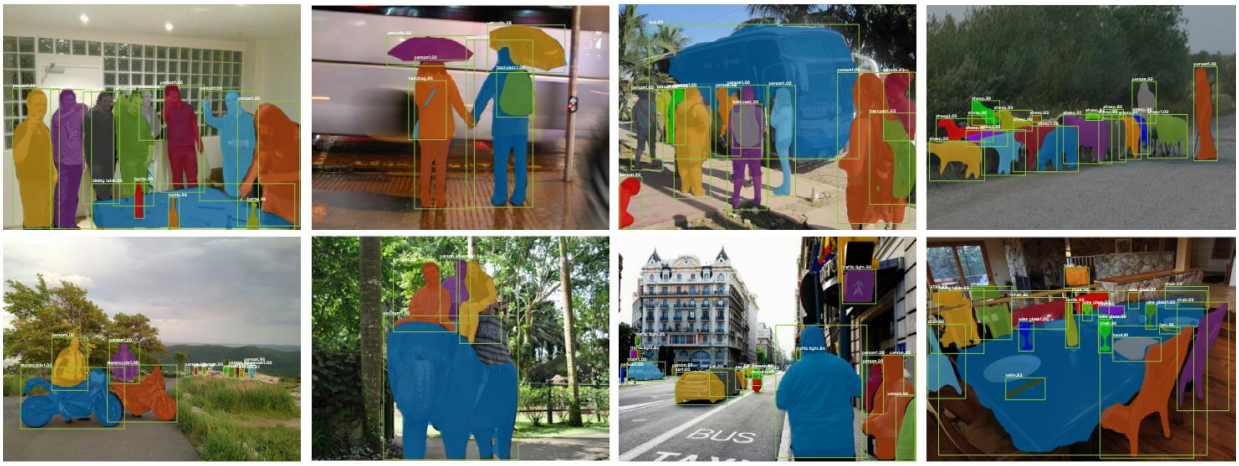
مقدمه: Mask R-CNN یک شبکه‌ی سریع، انعطاف پذیر و تعمیم پذیر می‌باشد که برای انجام Instance Segmentation طراحی شده است. ما در این گزارش قصد نداریم وارد جزئیات Mask R-CNN شویم و تنها مفاهیم کلی آن را بررسی می‌کنیم. این شبکه نه تنها به طور بهینه اشیاء موجود در تصاویر را تشخیص می‌دهد، بلکه به صورت همزمان برای هر شیء تشخیص داده شده یک Segmentation Mask با کیفیت بالا نیز تولید می‌کند. این شبکه به نوعی شبکه‌ی Faster R-CNN را با اضافه کردن یک شاخه‌ی جدید گسترش می‌دهد که از آن برای تولید Segmentation Mask به طور موازی با شاخه‌ی موجود برای تولید Bounding Box استفاده می‌شود. این شاخه‌ی جدید در واقع یک FCN است که به هر Region of Interest (RoI) اعمال می‌شود و به صورت پیکسل به پیکسل Segmentation Mask تولید می‌کند. در شکل ۶ نمونه خروجی شبکه‌ی Mask R-CNN را مشاهده می‌فرمایید.

Mask R-CNN از نظر مفهومی بسیار ساده است: شبکه‌ی Faster R-CNN برای هر شیء تشخیص داده شده دو خروجی دارد، یکی برچسب کلاس و یکی آفست های Bounding Box. کافیست به این ساختار یک شاخه‌ی سوم اضافه کنیم که Object Mask را در خروجی تولید می‌کند.

معماری شبکه: برای نشان دادن انعطاف پذیری Mask R-CNN، نویسندگان مقاله معماری های مختلفی را برای این مدل پیاده سازی کرده اند. معماری این شبکه به طور کلی دو بخش دارد: ۱) معماری بخش Backbone که یک شبکه کانولوشنی بوده و بردار

[12] Mask R-CNN

ویژگی ها را از تصاویر استخراج می‌کند. ۲) بخش Head که برای تولید مشخصات Bounding Box و Mask Prediction از آن استفاده می‌شود. در ادامه، در مورد هر کدام از این بخش ها به طور جداگانه صحبت خواهد شد و معماری های مورد استفاده برای هر بخش ذکر خواهد شد.



شکل ۶ – نمونه خروجی شبکه‌ی Mask R-CNN که بر روی دیتاست COCO آموزش دیده است.

برای بخش Backbone، از معماری های ResNet و ResNeXt با ۵۰ یا ۱۰۱ لایه استفاده شده است. پیاده سازی اصلی Faster R-CNN با استفاده از ResNet، ویژگی ها را از آخرین لایه‌ی کانولوشنی مرحله‌ی چهارم استخراج می‌کرد که به آن C4 می‌گوییم. بنابراین، استفاده زیرساخت ResNet-50 . استخراج ویژگی ها از C4 را ResNet-50-C4 نام‌گذاری می‌کنیم. نویسندگان مقاله همچنین از یک معماری مشهور دیگر نیز برای Backbone استفاده کرده اند که به آن Feature Pyramid Network (FPN) گفته می‌شود. شبکه‌ی Faster R-CNN با استفاده از FPN، ویژگی های RoI ها را از لول های مختلفی از Feature Pyramid با توجه به ابعاد آنها استخراج می‌کند، امّا بقیه‌ی ساختار دقیقا همانند ResNet است. استفاده از زیرساخت ResNet-FPN در شبکه‌ی Mask R-CNN منجر به بهترین نتایج چه از نظر سرعت و چه از نظر دقت شد. به همین دلیل بود که ما تصمیم گرفتیم تا از این زیرساخت در کتابخانه‌ی Detectron2 استفاده کنیم.

ارزیابی: بر اساس جدول های موجود در Model Zoo کتابخانه‌ی Detectron2، عملکرد شبکه‌ی Mask R-CNN با استفاده از زیرساخت R50-FPN و روی دیتاست COCO در جدول ۳ آورده شده است. لازم به ذکر است که عملکرد شبکه با استفاده از معیارهای AP جداگانه برای Box Prediction و Mask Prediction سنجیده شده است.

box AP	mask AP
41.0	37.2

جدول ۳ – عملکرد شبکه‌ی Mask R-CNN با استفاده از زیرساخت R50-FPN

اتصال دو شبکه به یکدیگر:

برای اتصال دو شبکه به یکدیگر، حالت های مختلفی را بررسی کردیم تا بهینه ترین حالت را پیدا کنیم. بر این اساس، می‌توان برای اتصال دو شبکه، دو استراتژی مختلف را دنبال نمود:

۱- ابتدا شبکه ی تشخیص اشیاء، تمامی اشیاء موجود در تصویر را تشخیص می‌دهد، و سپس با استفاده از نقشه‌ی عمق تولید شده توسط شبکه‌ی تشخیص عمق، عدد عمق هر شیء به دست می‌آید.

۲- ابتدا شبکه‌ی تشخیص عمق، نقشه‌ی عمق تصاویر را تولید می‌کند، سپس بر اساس محدودیت هایی که کاربر روی میزان عمق قابل تشخیص اعمال نموده، بخش هایی از تصویر اصلی را Mask می‌کنیم و آن را به شبکه‌ی تشخیص اشیاء می‌دهیم. سپس نتایج و Bounding Box های خروجی این شبکه بر روی تصویر اصلی اعمال خواهد شد.

پس از بررسی های انجام شده، تصمیم گرفتیم تا در این پروژه از استراتژی ۱ برای ترکیب دو شبکه استفاده کنیم زیرا به نظر می‌رسد این استراتژی منجر به نتایج بهتری می‌شود. همانطور که پیش تر اشاره شد، در این استراتژی ابتدا شبکه‌ی تشخیص اشیاء، تمامی اشیاء موجود در تصویر را تشخیص می‌دهد و سپس، عمق هر تصویر با توجه به نقشه‌ی عمق تولید شده توسط شبکه‌ی تشخیص عمق قابل محاسبه است. در ادامه، بر اساس محدودیت هایی که کاربر بر روی عمق قابل تشخیص اعمال کرده، آن Bounding Box هایی که شرایط مورد نظر کاربر را ندارند حذف می‌شوند و Bounding Box های باقیمانده به همراه عمق بر روی تصویر اصلی رسم می‌شوند و خروجی نهایی تولید می‌شود. امّا چگونه عمق هر شیء را از روی نقشه‌ی عمق محاسبه کنیم؟ اگر به خاطر داشته باشید، پیش تر اشاره شد که برای شبکه‌ی تشخیص اشیاء از Instance Segmentation استفاده کرده ایم که هم Box Prediction و هم Mask Prediction انجام می‌دهد. در اینجا دلیل اینکار مشخص می‌شود! Bounding Box ها به تنهایی اطلاعات کافی برای محاسبه‌ی عمق اشیاء را در اختیار ما نمی‌گذراند زیرا در هر Bounding Box، تنها درصدی از پیکسل ها مربوط به شیء تشخیص داده شده است و درصدی دیگر مربوط به سایر اشیاء موجود در پس زمینه هستند، بنابراین اطلاعات دقیقی از مختصات پیکسلی هر شیء در اختیار نداریم و تخمین عمق شیء از روی نقشه‌ی عمق بسیار دشوار و بی کیفیت می‌شود. امّا با استفاده از Instance Segmentation، علاوه بر Bounding Box ها، Segmentation Mask ها را نیز در اختیار داریم که مختصات پیکسلی هر شیء تشخیص داده شده را به طور دقیق مشخص می‌کنند. به عبارتی، Segmentation Mask به ما اجازه می‌دهد که به طور دقیق مرز های هر شیء تشخیص داده شده را مشخص کنیم. در واقع، با استفاده از Mask R-CNN، همانطور که در شکل ۶ مشاهده می‌شود، علاوه بر مختصات Bounding Box ها، برای هر شیء تشخیص داده شده یک آرایه‌ی True و False نیز در خروجی داریم که دارای ابعاد تصویر اصلی است و مکان پیکسل های متعلق به آن شیء را در اختیار ما قرار می‌دهد. در شکل ۷ نمونه یک تصویر و Segmentation Mask تولید شده برای یکی از اشیاء را مشاهده می‌فرمایید.

با داشتن پیکسل های هر شیء، می‌توانیم به سادگی پیکسل های متناظر را در نقشه‌ی تشخیص عمق بررسی کرده و با توجه به آنها عمق شیء را تشخیص دهیم. برای تخمین عمق اشیاء از روش میانگین گیری استفاده شده است، بدین معنا که عمق هر شیء برابر است با میانگین عمق تک تک پیکسل های آن شیء روی نقشه‌ی تشخیص عمق:

$$D_I = \frac{1}{n}\sum_{p\in I} D_p$$

در این رابطه، D_I عمق شیء مورد نظر است، n تعداد کل پیکسل های آن شیء است که توسط Segmentation Mask مشخص می‌شود، و $\sum_{p\in I} D_p$ عمق تمامی پیکسل های متعلق به آن شیء را جمع می‌کند.

شکل ۷ – نمونه یک تصویر و Segmentation Mask تولید شده برای اسب موجود در تصویر

ارزیابی: حال می‌خواهیم بر اساس معیار های عملکرد هر کدام از شبکه های جداگانه، عملکرد شبکه ی نهایی را تخمین بزنیم.

قبل از ورود به جزئیات بیشتر باید یک نکته ی مهم را یادآوری کنیم: شبکه‌ی تشخیص اشیاء، دو معیار مختلف را برای عملکرد شبکه در اختیار ما قرار می‌دهد، یکی AP_{box} که دقت تولید Bounding Box را نشان می‌دهد، و دیگری AP_{mask} که دقت تولید Segmentation Mask را نشان می‌دهد. با توجه به اینکه ما در معماری خود از Segmentation Mask های تولید شده توسط شبکه‌ی تشخیص اشیاء (و نه از Bounding Box های تولید شده) برای تخمین عمق استفاده می‌کنیم، بنابراین دقت AP_{box} تاثیری در عملکرد تشخیص عمق اشیاء ندارد. به عبارتی، ما در ابتدا با استفاده از معیار هایی که در ادامه معرفی می‌شوند و با ترکیب کردن AP_{mask} و معیار های عملکرد شبکه‌ی تشخیص عمق، دقت تخمین عمق اشیاء کلی را بدست می‌آوریم، و سپس بین عدد بدست آمده و عدد AP_{box} مینیمم میگیریم تا دقت شبکه‌ی نهایی بدست آید. به عبارتی، ما دقت شبکه‌ی نهایی را از مینیمم گرفتن بین دقت عملیات تشخیص عمق اشیاء و عملیات تشخیص اشیاء بدست می‌آوریم، امّا دقت عملیات تشخیص عمق اشیاء خود حاصل ترکیب دقت شبکه‌ی تشخیص عمق و AP_{mask} شبکه‌ی تشخیص شیء است. بنابراین ۲ مرحله‌ی زیر انجام می‌شود:

۱- با ترکیب دقت شبکه‌ی تشخیص عمق و AP_{mask} شبکه‌ی تشخیص اشیاء، دقت عملیات تشخیص عمق اشیاء را بدست می‌آوریم.

۲- بین عدد بدست آمده در مرحله‌ی ۱ و عدد AP_{box} مینیمم می‌گیریم تا دقت نهایی شبکه‌ی کلی بدست آید.

برای انجام مرحله‌ی اول، از دو روش زیر برای ترکیب استفاده می‌کنیم:

1- $\gamma_1 = \delta_1 \times AP_{mask} = 0.7960 \times 0.372 = 0.296$
2- $\gamma_2 = e^{-rms} \times AP_{mask} = 0.186$

بر این اساس، دقت شبکه‌ی نهایی به صورت زیر محاسبه خواهد شد:

$P_1 = min(\gamma_1, AP_{box})$	$P_2 = min(\gamma_2, AP_{box})$
29.6	18.6

جدول ۴ – دقت شبکه‌ی نهایی

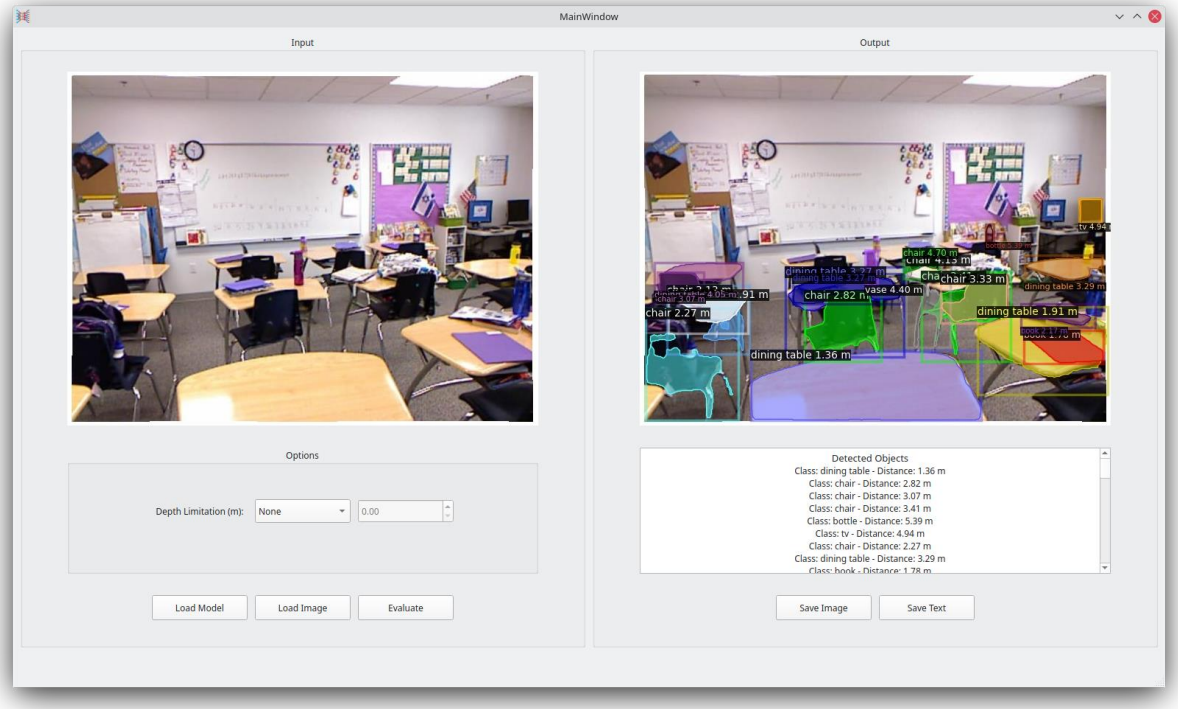
لازم به ذکر است که هر دو معیار معرفی شده برای سنجش دقت شبکه‌ی نهایی، کاملاً جدید بوده و توسط اینجانب معرفی شده اند و از هیچ مرجعی الهام گرفته نشده اند. نمونه ورودی و خروجی پروژه را در شکل ۸ مشاهده می‌کنید.

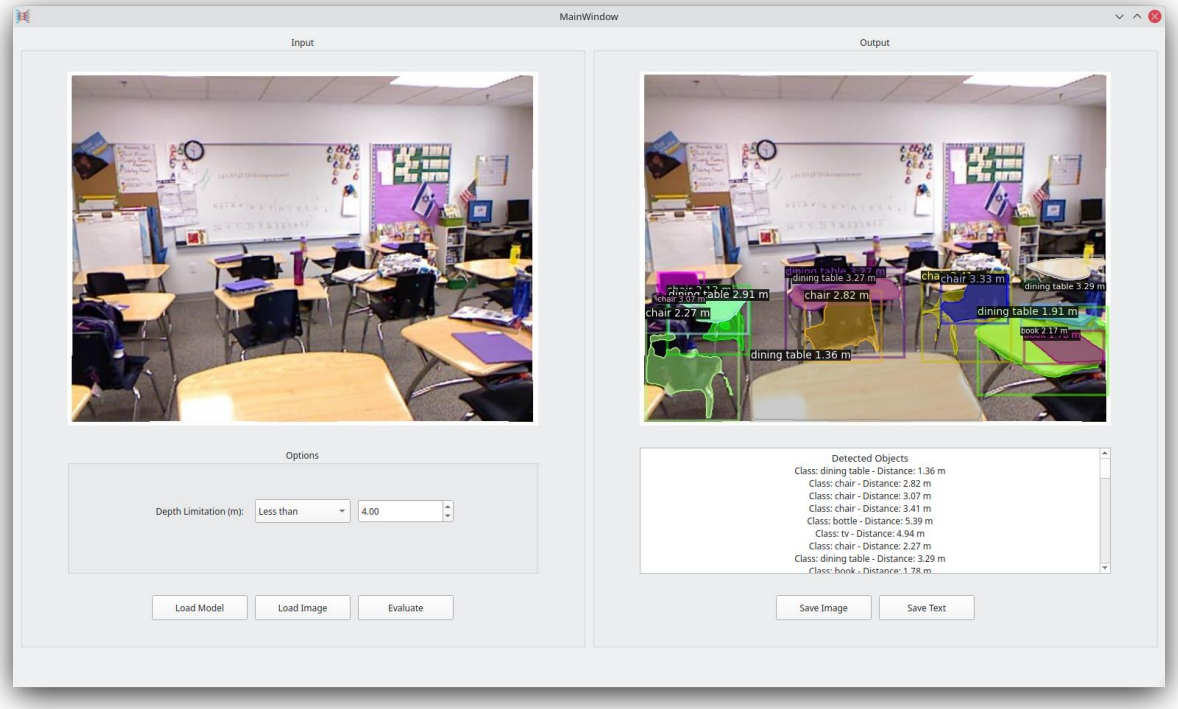
محیط گرافیکی:

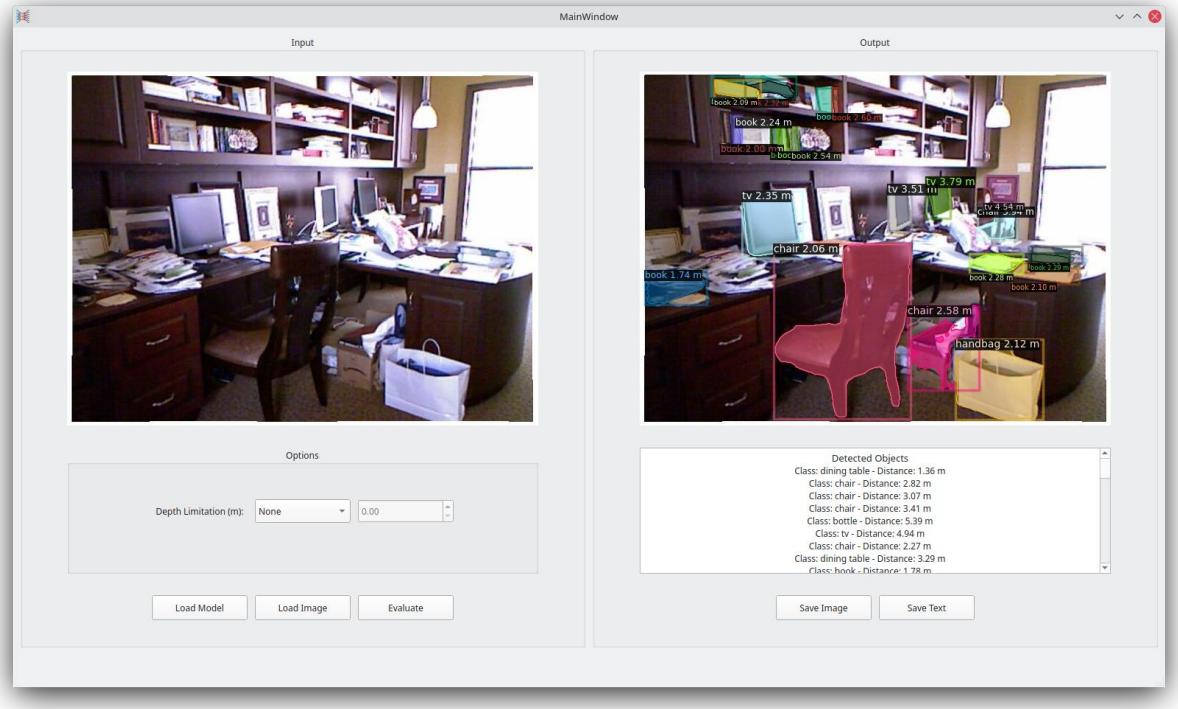
به منظور سهولت استفاده از این پروژه و همچنین جذابیت بصری آن، با استفاده از کتابخانه‌ی PyQt5 و ابزار Qt Designer، یک محیط گرافیکی تعاملی نیز برای آن ایجاد شده است. این محیط گرافیکی، مدل از پیش آموزش دیده و یک تصویر را از کاربر به عنوان ورودی دریافت می‌کند و با انجام عملیات Object Detection و Depth Estimation بر روی تصویر ورودی، تمامی اشیاء موجود و عمق هر شیء را مشخص کرده و در تصویر خروجی اعمال می‌کند. همچنین، تمامی اشیاء تشخیص داده شده و عمق آنها به صورت متنی جدای از تصویر نیز نمایش داده می‌شوند. آموزش قدم به قدم استفاده از محیط گرافیکی و سایر جزئیات در فایل README موجود در گیت هاب به طور کامل شرح داده شده است. نمونه ورودی و خروجی پروژه را در شکل ۸ مشاهده می‌کنید.

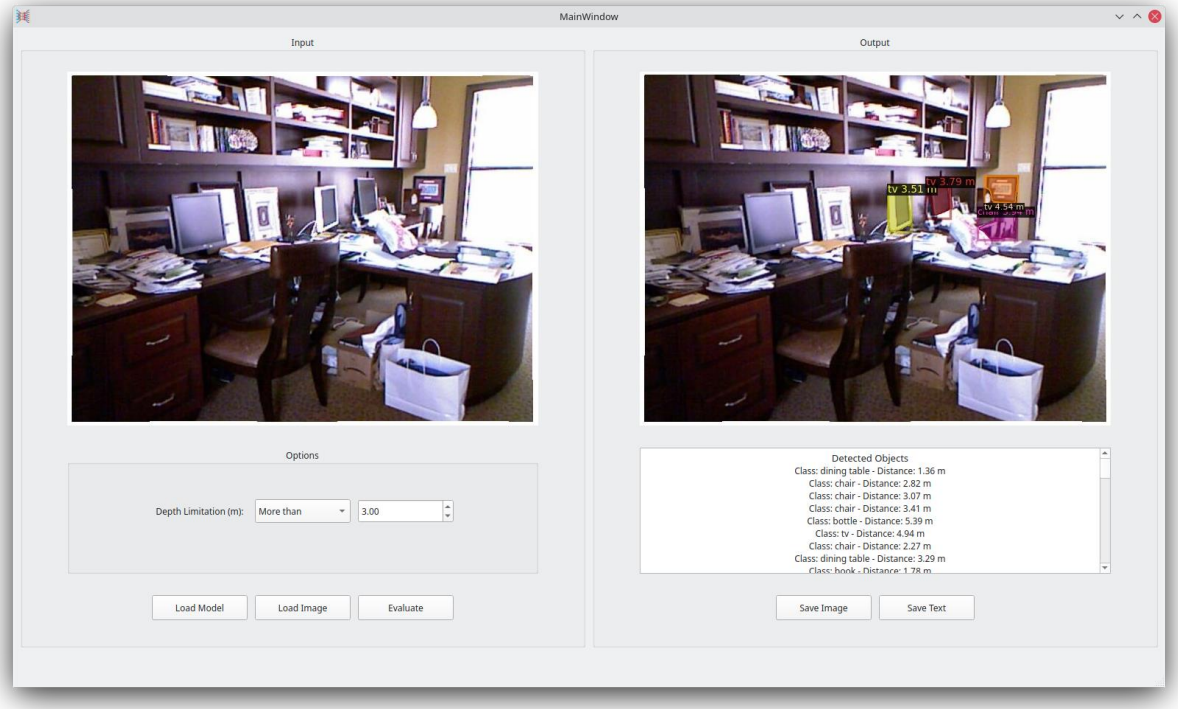
گیت هاب:

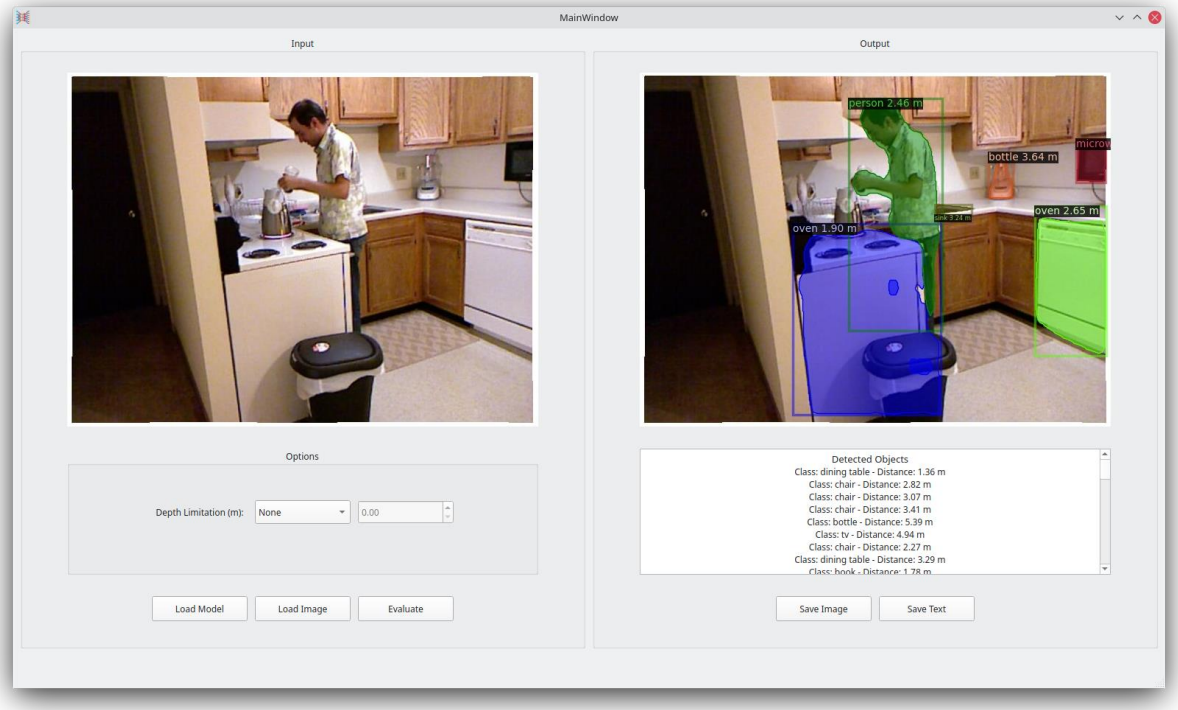
این گزارش به همراه تمامی فایل ها و کد های پروژه، نحوه‌ی نصب کتابخانه های مورد نیاز و سایر آموزش های مورد نیاز برای کار با آن در Repository گیت هاب موجود می‌باشد که جناب آقای بهراد احمدپور به آن اضافه شده اند. در صورتی‌که نیاز بود شخص دیگری نیز به Repository گیت هاب اضافه شود با ایمیل shokuhi.mh@gmail.com یا آیدی تلگرام @Mhosein_shokuhi در ارتباط باشید.


MainWindow
Input
Output
Options
Depth Limitation (m):
None
0.00
Load Model
Load Image
Evaluate
Save Image
Save Text
Detected Objects
Class: dining table - Distance: 1.36 m
Class: chair - Distance: 2.82 m
Class: chair - Distance: 3.07 m
Class: chair - Distance: 3.41 m
Class: bottle - Distance: 5.39 m
Class: tv - Distance: 4.94 m
Class: chair - Distance: 2.27 m
Class: dining table - Distance: 3.29 m
chair 2.27 m
chair 2.82 m
vase 4.40 m
chair 3.33 m
dining table 1.91 m
dining table 1.36 m


MainWindow
Input
Output
Options
Depth Limitation (m):
Less than
4.00
Load Model
Load Image
Evaluate
Save Image
Save Text
Detected Objects
Class: dining table - Distance: 1.36 m
Class: chair - Distance: 2.82 m
Class: chair - Distance: 3.07 m
Class: chair - Distance: 3.41 m
Class: bottle - Distance: 5.39 m
Class: tv - Distance: 4.94 m
Class: chair - Distance: 2.27 m
Class: dining table - Distance: 3.29 m
chair 2.27 m
dining table 3.27 m
chair 2.82 m
chair 3.33 m
dining table 1.91 m
dining table 1.36 m


MainWindow
Input
Output
book 2.24 m
tv 2.35 m
tv 3.51 m
tv 3.79 m
chair 2.06 m
book 1.74 m
chair 2.58 m
handbag 2.12 m
Options
Depth Limitation (m):
None
0.00
Load Model
Load Image
Evaluate
Detected Objects
Class: dining table - Distance: 1.36 m
Class: chair - Distance: 2.82 m
Class: chair - Distance: 3.07 m
Class: chair - Distance: 3.41 m
Class: bottle - Distance: 5.39 m
Class: tv - Distance: 4.94 m
Class: chair - Distance: 2.27 m
Class: dining table - Distance: 3.29 m
Save Image
Save Text


MainWindow
Input
Output
Options
Depth Limitation (m):
More than
3.00
Load Model
Load Image
Evaluate
Detected Objects
Class: dining table - Distance: 1.36 m
Class: chair - Distance: 2.82 m
Class: chair - Distance: 3.07 m
Class: chair - Distance: 3.41 m
Class: bottle - Distance: 5.39 m
Class: tv - Distance: 4.94 m
Class: chair - Distance: 2.27 m
Class: dining table - Distance: 3.29 m
Save Image
Save Text



شکل ۸ – چند نمونه ورودی و خروجی پروژه