

باسمه تعالی



پروژه‌ی درس یادگیری عمیق

نام استاد:

دکتر فاطمی‌زاده

دانشجویان:

محمد حسین زارعی / ۴۰۰۲۰۱۴۷۸

محمد سینا حسن‌نیا / ۹۶۱۰۸۵۱۵

مسعود ناطقی / ۹۶۱۰۲۵۶۷

تاریخ تحویل:

۱۴۰۰/۱۱/۱۶

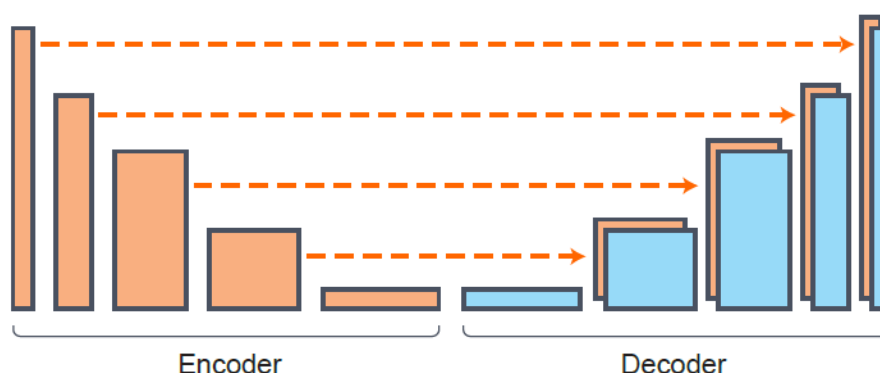
در این گزارش ابتدا به بررسی داده‌ها و دو مدل به کار رفته در شبکه می‌پردازیم و سپس مدل ترکیبی و اپلیکشین طراحی شده را معرفی می‌کنیم. در نهایت مباحث تئوری رو بیان می‌کنیم.

داده‌ها:

داده‌های مورد استفاده در این پروژه، از مجموعه‌ی NYUv2 هستند که پارامترهای آن در سایت معرفی شده در صورت پروژه، توضیح داده شده‌است. در این پروژه، از آنجا که صرفاً قصد داریم یک شبکه‌ی تخمین عمق آموزش دهیم، از پارامترهای images و depths استفاده می‌کنیم. مورد اول شامل آرایه‌های سه بعدی به صورت $H*W*C$ است که با توجه به ساختار داده‌ها، $H=480$ و $W=640$ و $C=3$ است. داده‌های عمق هم به صورت یک عکس دو بعدی $H*W$ است که هم اندازه با تصاویر این مجموعه‌ی داده‌است. با مطالعه‌ی توضیحات این مجموعه‌ی داده، متوجه شدیم که حداکثر عمق به کار رفته در تصاویر این مجموعه برابر ۱۰ متر است. در ادامه از این عدد ۱۰، استفاده خواهیم کرد. ضمناً ما از داده‌های فایل nyu_depth_v2_labeled استفاده کردیم. از آنجا که هدف ما صرفاً آموزش شبکه بود، از داده‌های پردازش شده برای عمق بهره بردیم که عمق تمامی پیکسل‌های تصاویر را به صورت مشخص شده در خود دارد.

شبکه‌ی تخمین عمق:

برای شبکه‌ی تخمین عمق از این مقاله^۱ استفاده کرده‌ایم. مدل به کار رفته در این شبکه DenseDepth نام دارد. ساختار اصلی این شبکه استفاده از یک ساختار autoencoder (ترکیب یک encoder و decoder) به همراه ایده‌ی skip connection است.



¹ <https://arxiv.org/abs/1812.11941>

برای encoder از شبکه‌ی از قبل آموزش‌دیده‌ی densenet169 استفاده شده‌است که ساختار آن را در درس مشاهده کرده‌ایم که عدد ۱۶۹ عمق این ساختار را مشخص می‌کند. در اینجا فقط تا خروجی قبل از بردار نهایی مدل densenet استفاده می‌کنیم. (یعنی خروجی لایه‌ی norm5 با نام features) خروجی encoder دارای ابعاد $15 \times 20 \times 1664$ است. یعنی نسبت به ورودی 480×640 ، با نرخ ۳۲، downsample رخ داده‌است. از ۴ بخش این شبکه، شاخه‌هایی را برای پیاده کردن ایده‌ی skip connection استفاده می‌کنیم. اول، خروجی لایه‌ی کانولوشن ابتدای کار است. دوم، خروجی maxpool اول است. که همان خروجی‌های denseblock های اول و دوم هستند.

در بخش decoder از یک سری بلوک‌های upsample استفاده می‌کنیم. این upsampling را در ۴ مرحله انجام می‌دهیم تا در نهایت به ابعادی برابر نصف ابعاد تصویر ورودی برسیم. همچنین در هر مرحله از خروجی متناظر بخش encoder استفاده می‌کنیم. هر مرحله شامل یک لایه‌ی upsample با نرخ ۲، ترکیب با بردار encoder و دو لایه‌ی کانولوشنی است. به این ترتیب ساختار کلی مدل آماده می‌شود.

اطلاعات کامل لایه‌ها را در تصویر زیر مشاهده می‌کنید. (لایه‌های نقطه‌چین همان لایه‌های شبکه‌ی densenet است.)

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

نکته‌ی مهم به کار رفته در این شبکه، این است که به جای استفاده از مقادیر عمق، از وارون آن‌ها برای آموزش شبکه استفاده کرده‌است. از آنجا که داده‌های NYUV2 در عمقی کمتر از ۱۰ متر تعریف می‌شوند، بنابراین خروجی مدل ده

برابر وارون عمق واقعی در نظر گرفته شده است. دلیل این موضوع برای این است که زمانی که عمقها زیاد باشد، طبیعتاً خطا هم زیاد خواهد بود. زیرا بازه‌ی تغییرات زیاد می‌شود. برای جبران این موضوع در این مقاله از وارون عمق واقعی استفاده شده است. در داده‌های KITTI که تصاویر outdoor هستند، حداکثر عمق برابر ۸۰ است. به همین علت برای این داده‌ها از ضریب ۸۰ برای مقیاس کردن وارون عمق استفاده شده است.

تابع خطای به کار رفته در مدل، از سه بخش تشکیل شده است. یک بخش همان تابع نرم یک بوده که با ضریب ۰.۱ با سایر بخش‌ها جمع شده است. طبیعتاً هدف این بخش از خطا این بوده که خروجی واقعی شبیه به خروجی درست باشد. بخش دوم معیاری از اندازه‌ی تغییرات پیکسل‌های مجاور است که تغییرات با فرکانس بالا را تا حد امکان کاهش دهد. بخش سوم هم معیار SSIM است که به عنوان معیاری از تشابه در بازسازی عکس‌ها به کار می‌رود و به صورت زیر تعریف می‌شود.

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

در رابطه‌ی بالا u همان میانگین و سیگما همان واریانس یا کواریانس تصاویر است. پارامترهای c_1 و c_2 اعداد ثابتی هستند. مشخص است زمانی که x و y یکسان باشند، این معیار به بیشترین مقدار خود یعنی یک می‌رسد. بنابراین در بخش سوم خطا، گزینه‌ای از آن به عنوان خطای در نظر گرفته می‌شود.

برای سنجش این شبکه‌ها از پارامترهای گوناگونی استفاده می‌شود. به طور مرسوم پارامترهایی که بیشتر استفاده می‌گردند عبارتند از :

- Percentage of Pixel (PP)
- PP-MVN
- Absolute Relative Difference (ARD)
- Square Relative Difference (SRD)
- RMSE-linear
- RMSE-log
- Scale-Invariant Error (SIE)

در ادامه با بررسی هر یک از این پارامترها سعی می‌کنیم تا به صورت بیشتر با این پارامترها آشنا شویم:

Percentage of Pixel (PP)

$$Percentage\ of\ pixel(PP) = \max\left(\frac{d_i}{d_i^*}, \frac{d_i^*}{d_i}\right) = \gamma < threshold$$

این معیار با نام دلتا مورد استفاده قرار می‌گیرد و حد آستانه‌ی آن به صورت توان‌های ۱.۲۵ تعریف می‌شود.

PP-MVN

در واقع این معیار از معیار PP بهره گرفته است که در بالا معرفی شد. در واقع این معیار مشابه معیار قبلی است با این تفاوت که در اینجا از Mean variance Normalized استفاده می کنیم که در فرمول ها زیر با حرف اختصاری MVN آورده شده است.

$$PP - MVN = \max\left(\frac{MVN(d_i)}{MVN(d_i^*)}, \frac{MVN(d_i^*)}{MVN(d_i)}\right) = \gamma < threshold$$

ARD

این روش که در واقع مخفف Absolute Relative Difference است، معیار را به صورت حاصل جمع قدر مطلق تفاضل d_i و d_i^* ها تعریف می کند. داریم:

$$ARD = \frac{1}{N} \sum_i |d_i - d_i^*| / d_i^*$$

SRD

این روش که تقریباً مشابه ARD است، از توان ۲ به جای قدر مطلق استفاده می کند. داریم:

$$ARD = \frac{1}{N} \sum_i (d_i - d_i^*)^2 / d_i^*$$

RMSE-linear

در این روش از نرم ۲ استفاده می کنیم.

$$RMSE - Linear = \sqrt{\frac{1}{N} \sum_i ||d_i - d_i^*||^2}$$

RMSE-log

$$RMSE - log = \sqrt{\frac{1}{N} \sum_i ||\log(d_i) - \log(d_i^*)||^2}$$

بحث: این ۲ معیار علی رغم شباهتی که در فرمول دارند تفاوت های عمیقی در عملکرد می توانند داشته باشند. این تفاوت را در قالب دو مثال بررسی می کنیم. اگر فرض کنیم x و y داده شده باشد. حال اگر $10x$ و $10y$ به دو معیار بدهیم، $RMSE-log$ مقداری یکسان را گزارش می کند. در حال که $RMSE$ معمولی مقداری متفاوت محاسبه خواهد کرد که این مقدار ۱۰ برابر مقدار گزارش شده در حالت قبلی است. در واقع $RMSE-log$ نرخ تغییرات را مقایسه می کند. برای شفاف سازی موضوع یک مثال دیگر ارائه خواهیم داد. اگر در حالت اول x و y و در حالت دوم $x+c$ و

$y+c$ را به معیارها دهیم، (یعنی اختلاف ها ثابت باشد) این بار RMSE-linear است که مقداری مشابه در دو حالت دارد. با توجه به موارد بالا و این که ما در این جا تصویر داریم و نرخ تغییرات برای ما بیشتر اهمیت دارد، به نظر استفاده از RMSE-log نسبت به RMSE-linear نتیجه بهتری خواهد داشت.

Scale-invariant-error

$$SIE = \frac{1}{N} \sum_i (\log(d_i) - \log(d_i^*)) + \frac{1}{N} \sum_j (\log(d_j) - \log(d_j^*))^2$$

دو روش دیگر برای مشاهده این معیار توسط فرم های معادل زیر ارائه شده است:

در این رابطه اگر $\alpha_i = \log(d_i) - \log(d_i^*)$ تعریف کنیم آنگاه خواهیم داشت:

$$SIE = \frac{1}{N^2} i((\log(d_i) - \log(d_j)) + \log(d_i^*) - \log(d_j^*))^2 \quad (*)$$

$$SIE = \frac{1}{N} \sum_i \alpha_i^2 - \frac{1}{N^2} \sum_{i,j} \alpha_i \alpha_j \quad (**)$$

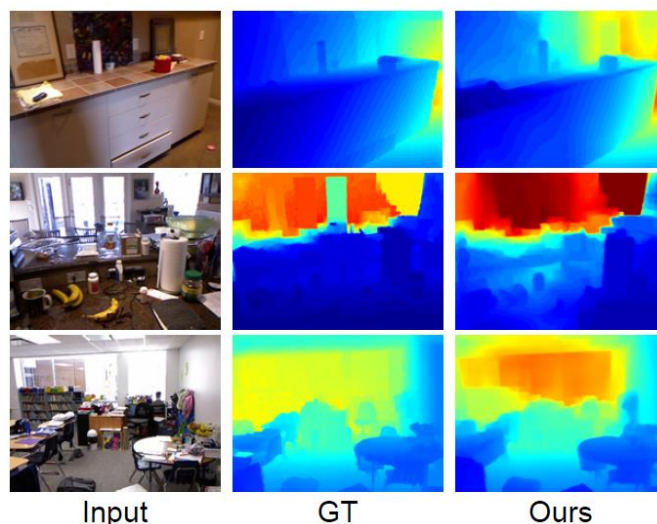
رابطه * خطا را با مقایسه بین جفت پیکسل های i و j در خروجی بیان می کند. برای آن که خطای کمی داشته باشیم، باید هر جفت پیکسل پیش بینی شده از نظر عمق با یک مقدار مشابه با جفت ها در ground truth متناظر تفاوت داشته باشد. رابطه ** در واقع معیار را به یک l2 error ولی با یک ترم اضافی $\frac{1}{N^2} \sum_{i,j} \alpha_i \alpha_j$ ارتباط می دهد.^۲

پس از بررسی معیارهای ارزیابی، به بیان خطای بیان شده ی شبکه ی آماده در مقاله ی مدل، می پردازیم. نتیجه ی موجود در مقاله در مورد داده های NYUV2 و تعدادی از نتایج روش های دیگر، به صورت زیر است:

Method	$\delta_1 \uparrow$	$\delta_2 \uparrow$	$\delta_3 \uparrow$	rel \downarrow	rms \downarrow	$\log_{10} \downarrow$
Eigen et al. [6]	0.769	0.950	0.988	0.158	0.641	-
Laina et al. [23]	0.811	0.953	0.988	0.127	0.573	0.055
MS-CRF [37]	0.811	0.954	0.987	0.121	0.586	0.052
Hao et al. [14]	0.841	0.966	0.991	0.127	0.555	0.053
Fu et al. [9]	0.828	0.965	0.992	0.115	0.509	0.051
Ours	0.846	0.974	0.994	0.123	0.465	0.053
Ours (scaled)	0.895	0.980	0.996	0.103	0.390	0.043

همچنین نمونه ای از خروجی های این شبکه را مشاهده می کنید:

² David Eigen , Christian Puhrsch , Rob Fergus “Depth Map Prediction from a Single Image using a Multi-Scale Deep Network”, Dept. of Computer Science, Courant Institute, New York University



همان طور که مشاهده می‌کنید، نتایج مدل بسیار قابل قبول بوده و نسبت به مدل‌های دیگر برتری نسبی دارد. به همین منظور، سعی کردیم که همین مدل را روی داده‌های NYUv2 آموزش دهیم.

آموزش شبکه‌ی تخمین عمق:

ساختار به کار رفته را دقیقاً مشابه مدل مقاله پیاده کردیم. همچنین خروجی‌ها را وارون و سپس در ۱۰ ضرب کردیم. از شبکه‌ی densenet با وزن‌های آموزش دیده استفاده کردیم و آن را freeze نکردیم. صرفاً مقادیر اولیه‌ی وزن‌ها را همان مقادیر آموزش دیده قرار دادیم.

نحوه‌ی تقسیم‌بندی داده‌ها هم به این شکل است که ۶۰ درصد را برای آموزش، ۲۰ درصد را برای ارزیابی و ۲۰ درصد را برای تست در نظر گرفتیم. خطای آموزش را فقط برابر نرم یک قرار دادیم و از سایر بخش‌های خطا صرف نظر کردیم. پارامتر دقت هم برابر MSE است. به دلیل محدودیت سخت‌افزار، سایز هر batch را نتوانستیم بیشتر از ۴ انتخاب کنیم. پارامترهای آموزش به شکل زیر است:

batch size = 4

learning rate = 0.0001

num epoch = 10

optimizer = Adam

نتایج آموزش به شرح زیر است:

Epoch=1

Training: loss=0.765 acc=0.901: 100% | ██████████ | 218/218 [10:58<00:00, 3.02s/it]

Validation: loss=1.110 acc=2.891: 100% | ██████████ | 73/73 [01:14<00:00, 1.02s/it]

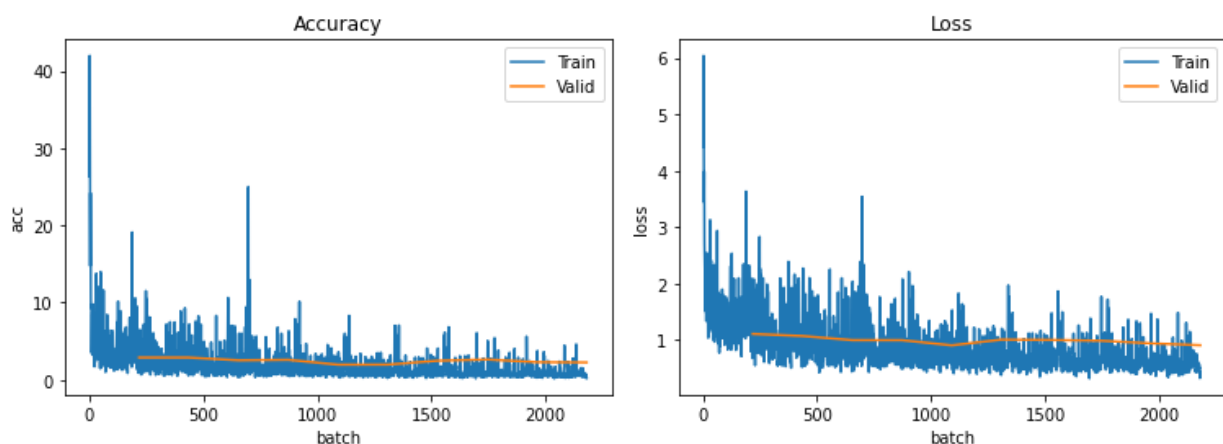
Epoch=2

Training: loss=0.639 acc=0.689: 100%|██████████| 218/218 [10:27<00:00, 2.88s/it]
 Validation: loss=1.073 acc=2.889: 100%|██████████| 73/73 [01:12<00:00, 1.00it/s]
 Epoch=3
 Training: loss=0.464 acc=0.406: 100%|██████████| 218/218 [10:36<00:00, 2.92s/it]
 Validation: loss=1.000 acc=2.538: 100%|██████████| 73/73 [01:12<00:00, 1.00it/s]
 Epoch=4
 Training: loss=0.542 acc=0.564: 100%|██████████| 218/218 [10:34<00:00, 2.91s/it]
 Validation: loss=0.999 acc=2.597: 100%|██████████| 73/73 [01:14<00:00, 1.02s/it]
 Epoch=5
 Training: loss=0.733 acc=1.148: 100%|██████████| 218/218 [10:35<00:00, 2.91s/it]
 Validation: loss=0.907 acc=1.966: 100%|██████████| 73/73 [01:15<00:00, 1.03s/it]
 Epoch=6
 Training: loss=0.388 acc=0.289: 100%|██████████| 218/218 [10:38<00:00, 2.93s/it]
 Validation: loss=1.009 acc=1.974: 100%|██████████| 73/73 [01:15<00:00, 1.03s/it]
 Epoch=7
 Training: loss=0.429 acc=0.432: 100%|██████████| 218/218 [10:30<00:00, 2.89s/it]
 Validation: loss=1.003 acc=2.479: 100%|██████████| 73/73 [01:13<00:00, 1.01s/it]
 Epoch=8
 Training: loss=0.505 acc=0.451: 100%|██████████| 218/218 [10:33<00:00, 2.90s/it]
 Validation: loss=0.989 acc=2.637: 100%|██████████| 73/73 [01:14<00:00, 1.02s/it]
 Epoch=9
 Training: loss=0.394 acc=0.269: 100%|██████████| 218/218 [10:38<00:00, 2.93s/it]
 Validation: loss=0.943 acc=2.293: 100%|██████████| 73/73 [01:14<00:00, 1.03s/it]
 Epoch=10
 Training: loss=0.403 acc=0.254: 100%|██████████| 218/218 [10:36<00:00, 2.92s/it]
 Validation: loss=0.910 acc=2.256: 100%|██████████| 73/73 [01:15<00:00, 1.03s/it]

با بررسی داده‌های تست به نتایج زیر می‌رسیم که قابل مقایسه با جدول بالا خواهد بود.

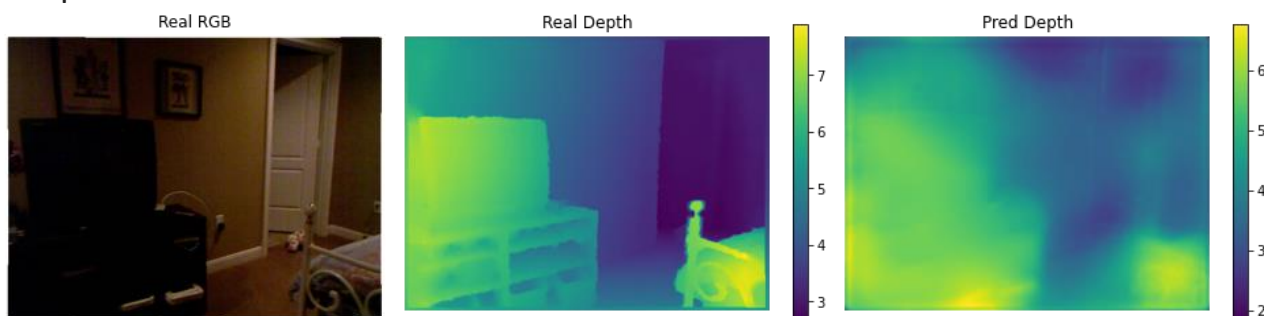
Testing: loss=0.964 acc=2.364: 100%|██████████| 73/73 [01:19<00:00, 1.09s/it]
 delta1=0.652, delta2=0.903, delta3=0.972, abs_rel=0.189, rmse=1.448, log_10=0.088

نمودار خطای آموزش و اعتبارسنجی در نمودار زیر آمده‌است: (loss: L1, acc: MSE)

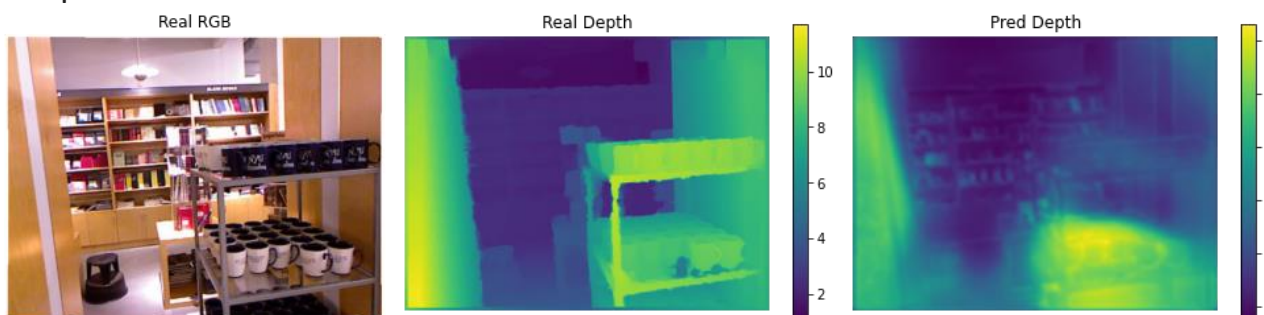


خروجی دو نمونه از تصاویر داده‌های تست به صورت زیر است: (تصاویر زیر مربوط به ۱۰ برابر وارون عمق است).

Sample 1:



Sample 2:



همان طور که مشاهده می‌کنید، تصاویر از دقت خوبی برخوردار هستند و می‌توان به آن‌ها اعتماد کرد. البته این نتایج پس از ۱۰ epoch و تقریباً دو ساعت آموزش به دست آمده‌است. اگر محدودیت سخت‌افزار نبود و تعداد epoch بیشتری مدل را آموزش می‌دادیم، نتایج نسبتاً بهبود می‌یافت. البته برای رسیدن به مقدار عمق، باید خروجی مدل را وارون کرده و سپس در ۱۰ ضرب کنیم. نکته‌ی دیگری که در عمق خروجی مدل مشاهده می‌شود، به خصوص در عکس پایین، تغییرات خروجی در نواحی نزدیک کتاب‌ها است که عمق تخمین زده شده، نوساناتی دارد. دلیل این موضوع را می‌توان در قرار ندادن بخش گرادیان در خطا دانست که از چنین مواردی جلوگیری می‌کرد.

کد این مدل در فایل `Train.ipynb` موجود است. همچنین مدل ترین شده را می‌توانید از این لینک^۳ دانلود کنید. برای لود کردن مدل از همان فایل `Train` استفاده کنید.

در مدل نهایی، چون مدل مقاله‌ی مورد نظر دقت بسیار بالایی داشت، برای افزایش دقت، از مدل از قبل آموزش‌دیده‌ی موجود در این آدرس^۴ استفاده کردیم. از آنجا که مدل موجود در این آدرس با زبان تنسورفلو نوشته شده بود و ما قصد داشتیم صرفاً از یک کلاس برای پیاده‌سازی مدل‌ها استفاده کنیم، ابتدا مدل را با کلاس تنسورفلو لود کردیم و سپس ضرایب آن را به کلاس متناظر پایتورچ منقل کردیم و با ذخیره‌ی این کلاس، عملاً به مدل `pytorch` رسیدیم.

^۳ <https://drive.google.com/file/d/1C0pyf3g7rnw6swQTGPAdZXPXto16iZ-w/view?usp=sharing>

^۴ <https://github.com/ialhashim/DenseDepth>

علاوه بر مدل آموزش دیده روی داده های NYU که مربوط به صحنه های indoor است، مدل دیگری با همین ساختار وجود دارد که روی داده های KITTI آموزش دیده و برای داده های outdoor است. مشابه این مدل را به فرمت pytorch ذخیره می کنیم. این دو مدل با نام های ⁵nuy.pt و ⁶kitti.pt از این لینک قابل دانلود است. در ادامه برای تخمین عمق در مدل نهایی، بسته به انتخاب کاربر که تصویر مورد نظرش outdoor یا indoor باشد، از یکی از این دو مدل استفاده می کنیم.

شبکه ی تشخیص اشیا:

برای تشخیص اشیا از شبکه ی yolov5 استفاده کردیم که در قالب pytorch ارائه شده و مدل آن از این لینک ⁷ قابل استفاده است. اطلاعات ورژن های مختلف yolo در این مقاله ⁸ قابل مشاهده است. ساختار پایه ی شبکه ی yolo را در تمرین دوم بررسی کردیم و گفتیم که در این شبکه ابتدا تصویر ورودی به تعدادی grid تقسیم شده و سپس به هر grid تعدادی box اختصاص داده می شود تا اشیا داخل آن تشخیص داده شود. ویژگی های هر box برابر مختصات مرکز و طول و عرض است. همچنین یک ویژگی confidence هم وجود دارد که نشان می دهد با چه احتمالی یک شی در این box وجود دارد. برای بررسی درستی یک box، از معیار IOU استفاده می شود که برابر نسبت اشتراک یک box با box های ground truth به اجتماع آنها است و مثلاً اگر از یک آستانه ای کمتر باشد، آن box برچسب منفی می گیرد. یعنی اینکه در آن شیئی وجود ندارد. ولی اگر مثبت باشد، به معنی برچسب مثبت و وجود شیء است. علاوه بر این ها یک توزیعی از کلاس ها برای هر grid هم تخمین زده می شود. این موارد ساختار پایه ی شبکه ی yolo است که در ورژن های بالاتر بهبودهایی ایجاد شده است. ساختارهای ورژن های ۳ و ۴ و ۵ بسیار مشابه هم بوده و تفاوت های جزئی دارند. برای مثال در ورژن ۲، به جای آنکه برای هر grid از یک توزیع برای کلاس ها استفاده شود، برای هر box این توزیع تخمین زده می شود. همچنین از لایه های batch normalization هم استفاده می شود. در ورژن ۳ هم از box های با نسبت ابعادی متفاوت مورد استفاده قرار می گیرد. در ورژن های بعدی، ساختار پایه شبیه ورژن ۳ بوده، با این تفاوت که در لایه های استخراج ویژگی، از مدل های متفاوتی استفاده می شود.

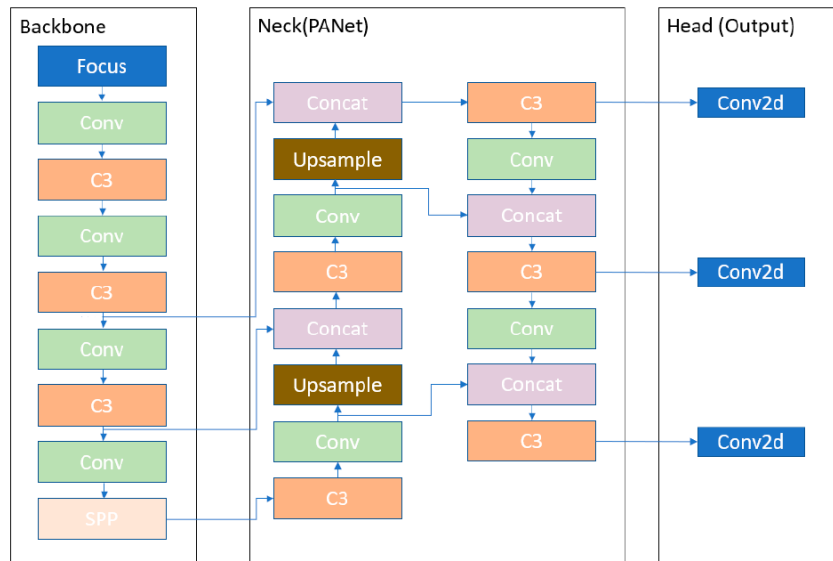
ساختار شبکه ی yolov5 را در تصویر زیر مشاهده می کنید.

⁵ <https://drive.google.com/file/d/1-1ie-JdkIR5MnysQyX5uldGOAHEqQPbf/view?usp=sharing>

⁶ <https://drive.google.com/file/d/1-D3zSiSPNa7l6lHSGgZn4X0PYQs7QiOl/view?usp=sharing>

⁷ <https://github.com/ultralytics/yolov5>

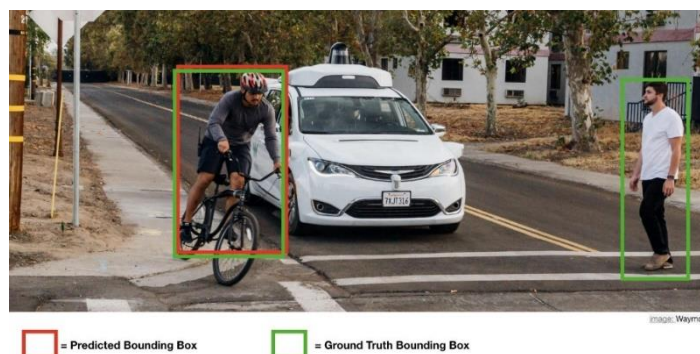
⁸ <https://www.mdpi.com/1424-8220/22/2/464/htm>



همان طور که در شکل بالا می بینید، شبکه از سه بخش **backbone**، **neck** و **head** تشکیل شده است. بخش **backbone** برای استخراج ویژگی های تصویر ورودی به کار می رود. در بخش **neck** این ویژگی های به دست آمده، به شکل های مختلف ترکیب می شود. زیرا در داخل این بخش، از مسیرهای **skip connection** در قسمت های مختلف استفاده شده و همچنین به کار رفتن ساختار هرمی در این بخش (کاهش تعداد بلوک ها در لایه های پایانی) امر ترکیب ویژگی ها را محقق ساخته است. بخش آخر هم با نام **head** مربوط به ساختار پایه ی **yolo** بوده و برای تخمین زدن پارامترهای **box** ها به کار می رود.

از تفاوت های ورژن ۵ نسبت به ورژن های ۳ و ۴، استفاده از بلوک **focus** است که به جای ۳ لایه کانولوشنی به کار رفته و باعث بهبود عملکرد شبکه چه از لحاظ حافظه و چه از لحاظ سرعت شده است. همچنین لایه **SPP** یک لایه **pooling** است که استفاده از آن باعث شده تا قید ثابت بودن ابعاد ورودی برداشته شود.

معیار ارزیابی اصلی یک شبکه ی تشخیص اشیا، معیار **mAP** است. همان طور که قبلا گفتیم، برای هر **box** پیشبینی شده، اگر **IOU** آن بیشتر از یک حد آستانه نسبت به **ground truth** باشد، آن **TP** شناخته می شود. ولی اگر کمتر باشد، **FP** است. **FN** هم یعنی اینکه برای یک **box** در **ground truth** اصلا **box** پیشبینی نشده باشد. با این تفاسیر، **precision** و **recall** قابل تعریف است.



برای مثال در تصویر بالا، دو box در ground truth و یک box تخمین زده شده داریم. برای box تخمینی، چون IOU آن مناسب است، (در مقایسه با حد آستانه) بنابراین TP برابر یک خواهد بود. همچنین چون هیچ box تخمینی نداریم که IOU آن پایین باشد، پس FP صفر است. اما چون یک box در gt داریم که هیچ box متناظر تخمینی ندارد، پس FN یک است و داریم:

$$\text{Precision} = \text{TP} / (\text{TP} + \text{fP}) = 1 \quad \text{Recall} = \text{TP} / (\text{TP} + \text{FN}) = 0.5$$

پس از محاسبه ی precision و recall برای هر کلاس، با اندازه گیری سطح زیر منحنی precision-recall مقدار AP یا همان average precision به دست می آید. این منحنی با در نظر گرفتن آستانه های متفاوت محاسبه می شود. حال به کمک رابطه ی زیر مساحت زیر منحنی قابل محاسبه است:

$$AP = \sum_{k=0}^{k=n-1} [\text{Recalls}(k) - \text{Recalls}(k+1)] * \text{Precisions}(k)$$

$\text{Recalls}(n) = 0, \text{Precisions}(n) = 1$
 $n = \text{Number of thresholds.}$

حال با میانگین گیری از مقدار AP برای تمامی کلاس ها معیار mAP محاسبه می شود. طبیعتاً تمایل به این است که مساحت زیر منحنی زیاد باشد. زیرا در آن صورت هم precision و هم recall به یک نزدیکتر خواهند بود. در طی این مراحل نقش score این است که از بین box های تخمینی که IOU آستانه را دارند و اشتراک زیادی هم با سایر box ها دارند، آن هایی را که ضریب اطمینان پایینی دارد، حذف کند. به این کار non maximum suppression می گویند.

مقایسه ی ورژن های yolo را در تصویر زیر مشاهده می کنید.

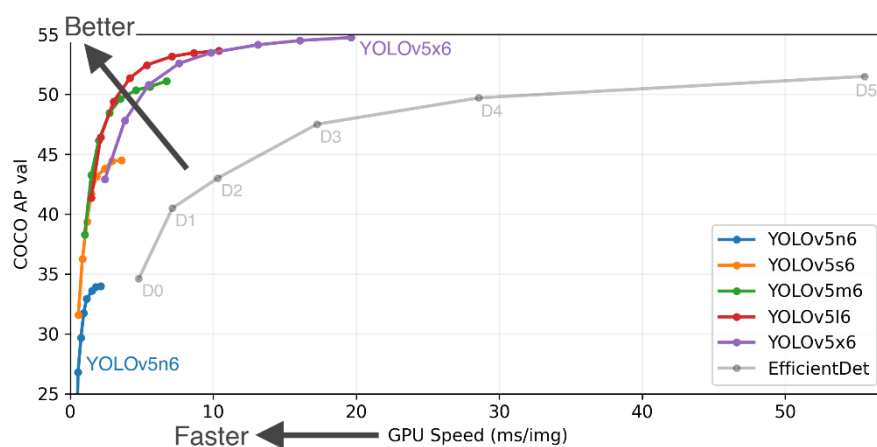
Measure	YOLOv3	YOLOv4	YOLOv5l
Precision	0.73	0.69	0.707
Recall	0.41	0.57	0.611
F1 Score	0.53	0.63	0.655
mAP	0.46	0.607	0.633
PC Speed (FPS)	63.7	59	58.82
Jetson Speed (FPS)	7.5	6.8	5

نمونه ای از عملکرد yolov5 در تصویر زیر قابل مشاهده است.



همان طور که مشخص است، این شبکه دقت بسیار بالایی داشته و با اطمینان بالایی اشیا را تشخیص داده است.

در آدرس yolov5 که دل‌های آن قرار دارد، فرمت‌های مختلفی از این شبکه با نام‌های n, s, m, l و x موجود است که این نام‌گذاری بر اساس حجم مدل‌ها است. یعنی هرچه از مدل nano به سمت مدل xlarge می‌رویم، ابعاد شبکه بزرگتر شده و طبیعتاً دقت آن‌ها بیشتر می‌شود. اما سرعت شبکه پایین می‌آید. نمودار زیر مقایسه‌ای بین این مدل‌ها است:



در مدل ترکیبی، ما از نوع S استفاده کرده‌ایم، هرچند نوع مدل قابل تغییر است.

مدل پیاده‌شده:

در مدل ترکیبی، ما از حالت ساده‌ی موازی استفاده کرده‌ایم. برای تشخیص اشیا از مدل yolov5 آماده و برای تخمین عمق از DenseDepth آماده استفاده کرده‌ایم که لینک مدل‌ها و سایت‌های مربوطه در توضیح هر بخش قرار داده شده است. در این مدل، ما عکس ورودی را به صورت مجزا به هر کدام از دو مدل داده و سپس متناسب با box های

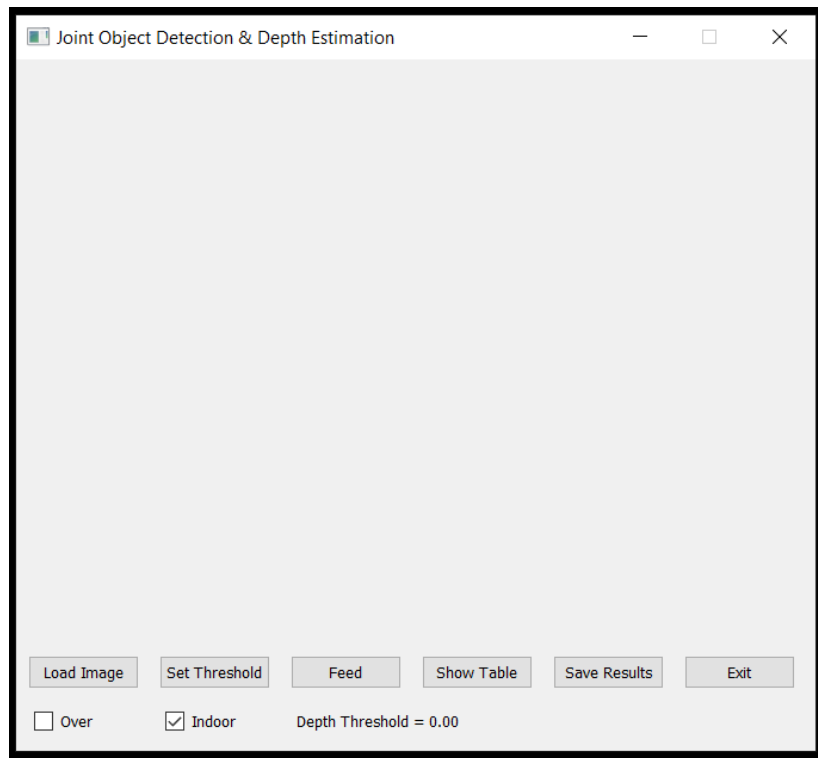
خروجی مدل تشخیص اشیاء، از خروجی مدل تخمین عمق میانگین‌گیری می‌کنیم تا عمق متوسط هر شی به‌دست آید. ناحیه‌ی میانگین‌گیری را برابر ۰.۴ طول و عرض box اصلی در نظر می‌گیریم. به این ترتیب خروجی ما آماده می‌شود. برای راحتی کار، از کتابخانه‌ی PyQt استفاده کردیم تا برنامه را مشابه یک application طراحی کنیم. حال به بررسی جزئیات می‌پردازیم.

با توجه به حجم بودن فایل مدل‌ها، آن‌ها را باید جداگانه دانلود کنید. لینک‌های مربوط به مدل تخمین عمق قبلاً قرار داده شده‌است. مدل‌های تشخیص اشیاء هم از قسمت Assets در این لینک^۹ قابل دانلود است. app نوشته‌شده، در پوشه‌ی Application در git قرار دارد و می‌توانید آن را دانلود کنید. برای اجرای app، کافی است فایل run.py را بدون هیچگونه عبارت اضافه‌ای اجرا کنید. البته قبل از آن باید مدل‌های دانلود شده را در پوشه‌ی Application-Project->Models قرار دهید. مدل پایه برای شبکه‌ی تشخیص اشیاء نوع S است. بنابراین کافی است فقط مدل yolov5l.pt را دانلود کرده و در پوشه‌ی Models قرار دهید. در صورتی که مایل به استفاده از سایر نوع‌ها هستید، پس از دانلود مدل مورد نظر و قرار دادن آن در پوشه‌ی Models، باید در فایل Graphic.py و در داخل تابع main در انتهای آن، آدرس Models/yolov5l را به محل مدل خود تغییر دهید. همچنین می‌توانید به صورت دستی، محل‌های دلخواه خود را وارد کنید. پس از انجام این مراحل، app آماده می‌شود.

حال فایل run.py را اجرا کنید. برای اجرا می‌توان از cmd استفاده کرده عبارت python ryn.py را وارد کرد. همچنین با رفتن به Properties فایل run.py، می‌توان Opens with آن را تغییر داد و از پایتون سیستم استفاده کرد تا با دابل کلیک کردن روی آن، فایل اجرا شود. در این صورت، یک فایل cmd علاوه بر فایل windows app باز می‌شود که نتایج اجرا در آن نوشته شده‌است. اگر مشکلی در اجرا باشد و یا پکیجی نصب نشده باشد، می‌توانید در آنجا مشاهده کنید. یکی از warning هایی که به کرات مشاهده می‌شود، مربوط به PyQt است که مورد خاصی نیست و مشکلی ایجاد نمی‌کند. همچنین کتابخانه‌های مورد نیاز برای این app، در داخل پوشه‌ی yolov5 و فایل requirements.txt نوشته شده‌است. پوشه‌ی yolov5 همان ریپازیتوری clone شده است.

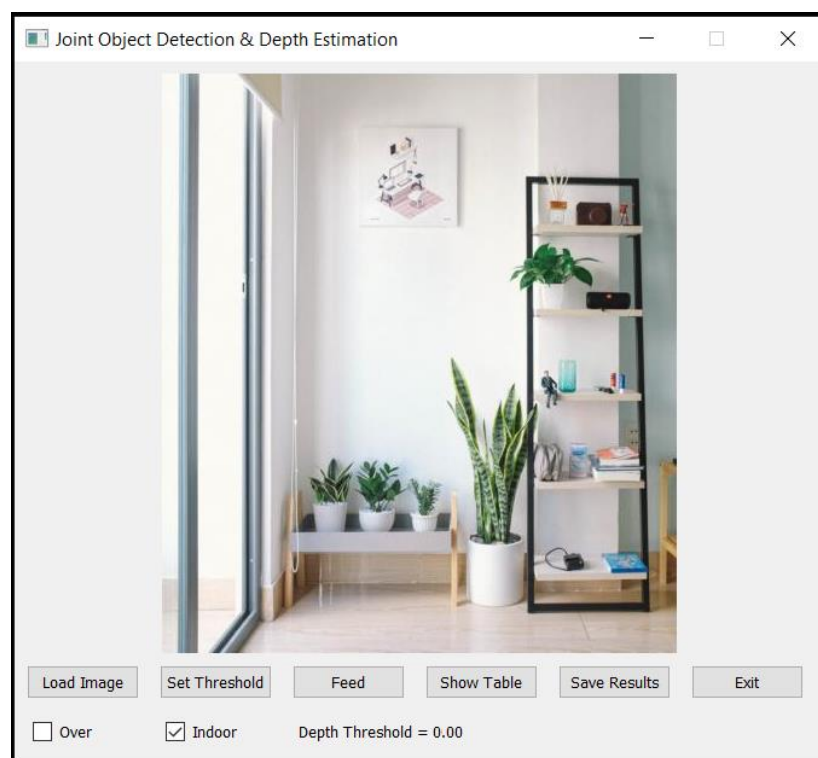
به دلیل اینکه در ابتدای اجرا، مدل‌ها لود می‌شوند و این کار زمانبر است، به همین دلیل، شروع برنامه با تاخیر است. همچنین در صورتی که زمان زیادی این کار طول کشید، یک بار بسته و دوباره اجرا کنید. با باز شدن app، مدل‌ها هم لود شده‌اند و با چنین صفحه‌ی خالی‌ای روبه‌رو می‌شود.

^۹ <https://github.com/ultralytics/yolov5/releases>

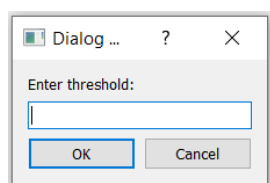


حال به معرفی دکمه‌ها می‌پردازیم. (از سمت چپ)

Load Image: با زدن این دکمه، تصویر محل تصویر ورودی از شما خواسته می‌شود. تنها تصاویر با فرمت **.jpg** مجاز است. همچنین محدودیتی در ابعاد تصویر ورودی نیست اما شبکه به ازای تصاویر با ابعاد بیشتر از ۱۰۰۰ پیکسل، بسیار کند کار می‌کند. ترجیحا تصویر در همان ابعاد داده‌های **NYU** باشد. پس از لود تصویر، **app** به شکل زیر در می‌آید:



Set Threshold: از این دکمه برای تعیین عمق آستانه استفاده می‌شود. پس از فشردن این دکمه، دیالوگ جدید باز می‌شود که عمق آستانه را باید در آن وارد کنید. با وارد کردن عبارت غیر عددی، آستانه‌ی قبلی تغییری نمی‌کند. مثلاً با وارد کردن عدد ۵، نتیجه به شکل زیر می‌شود. دقت شود که اگر آستانه صفر باشد، در خروجی تمامی اشیاء با تمامی عمق‌ها تشخیص داده می‌شود.

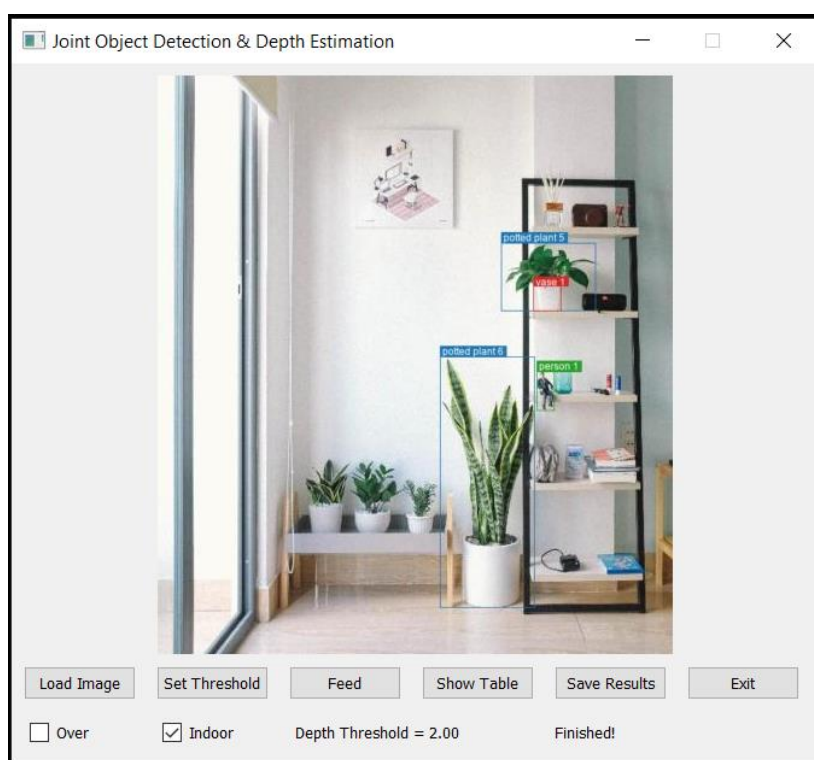


Depth Threshold = 5.00

Over: در پایان app، یک checkbox با نام **over** وجود دارد. اگر این گزینه چک باشد، نتیجه‌ی نهایی شامل اشیایی است که در عمقی بیشتر از عمق آستانه هستند و اگر تیک نخورده باشد، نتیجه‌ی نهایی شامل اشیایی است که در عمقی جلوتر از آستانه قرار دارند.





Indoor: با توجه به اینکه ما دو مدل آماده برای تخمین عمق داریم، یکی برای **indoor** و یکی برای **outdoor**، در این قسمت از کاربر می‌خواهیم که مشخص کند که تصویر مربوط به یک صحنه‌ی **indoor** است یا **outdoor**. با تیک زدن این گزینه، عملاً از مدل مربوط به **indoor** استفاده می‌شود.

Feed: پس از آماده‌سازی تصویر، با زدن این دکمه، تصویر به مدل‌ها داده می‌شود. این بخش از کار هم کمی زمانبر بوده و برای مدتی app قفل می‌شود. به محض آماده شدن خروجی، تصویری شامل **box** های متناظر اشیای تشخیص داده شده نشان داده می‌شود. خروجی زیر مربوط به تصویر بالا و حالت عمق‌های نزدیکتر از ۲ متر است:



همان‌طور که می‌بینید، تعدادی box روی تصویر ظاهر می‌شود که نام اشیا بالای آن نوشته شده‌است.

Show Tabel: برای مشاهده‌ی اطلاعات دقیق این box ها، روی این دکمه کلیک کنید. با زدن آن، پنجره‌ی زیر ظاهر می‌شود:

		Name	Confidence	Depth
1		person 1	0.40	1.89
2		potted plant 5	0.75	1.68
3		potted plant 6	0.86	1.91
4		vase 1	0.29	1.92

در این لیست، نام اشیا، رنگ قاب متناظرشان، ضریب اطمینان و عمق مشخص شده‌است. همان‌طور که می‌بینید، فقط عمق‌های کمتر از ۲ مشخص شده‌است.

Save Results: برای ذخیره‌ی این اطلاعات از این دکمه استفاده می‌شود. پس از فشردن آن، مسیری برای ذخیره درخواست می‌شود. همچنین نام فایل را هم باید وارد کنید. دقت شود که صرفاً نام را بدون پسوند وارد کنید. زیرا قرار است یک عکس همین تصویر خروجی است، به همراه یک فایل txt که شامل اطلاعات اشیا است، ذخیره شود. خروجی‌ها ذخیره شده به صورت زیر است:



Name	Depth	Confidence
person 1	1.89	0.40
potted plant 5	1.68	0.75
potted plant 6	1.91	0.86
vase 1	1.92	0.29

Exit: با فشردن این دکمه، برنامه بسته می‌شود.

سعی شده‌است تا حد امکان، از موارد غیرمنطقی جلوگیری شود. مثلاً تا زمانی که عکسی آپلود نشده، دکمه‌های Feed و Show Tables و Save Results کار نکنند و با زدن آن‌ها یک پیغام متناسب ظاهر شود. همچنین با آپلود عکس جدید، اطلاعات قبلی از بین می‌رود.

تصویر بالا را بار دیگر برا عمق‌های دورتر از ۲ متر تست می‌کنیم.

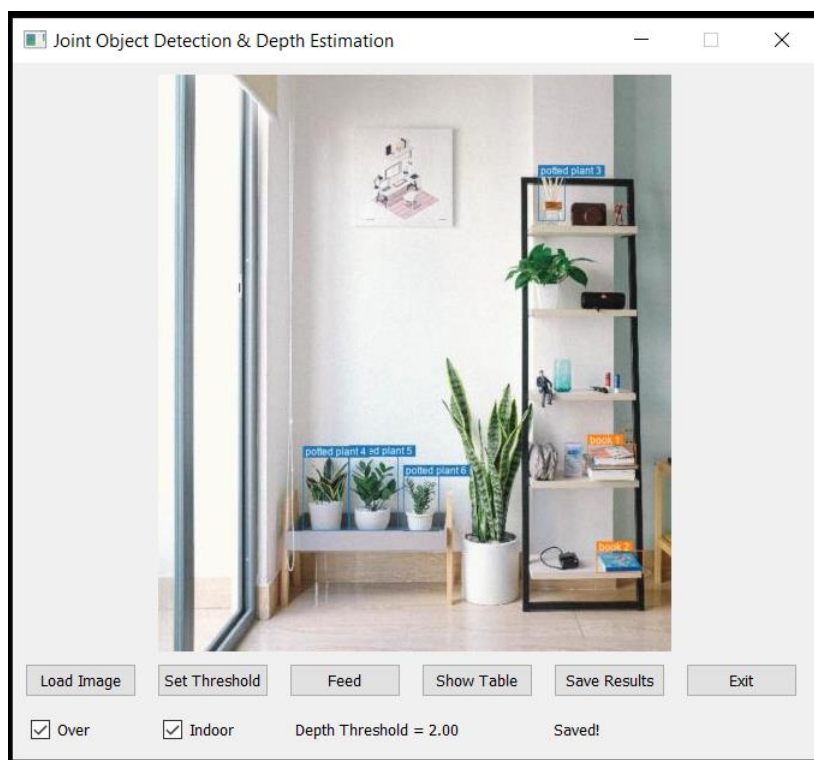








Table				
		Name	Confidence	Depth
1		book 1	0.42	2.06
2		book 2	0.53	2.14
3		potted plant 3	0.28	2.02
4		potted plant 4	0.74	2.14
5		potted plant 5	0.76	2.16
6		potted plant 6	0.81	2.19

همان طور که مشخص اشیایی خروجی، عمقی بیشتر از ۲ متر دارند. تعدادی از نتایج در پوشه‌ی Samples در git قرار دارند. این خروجی‌ها را برای آستانه‌ی صفر در نظر گرفتیم تا تمامی اشیاء تشخیص داده‌شود. (این نتایج مربوط به شبکه‌ی yolov5s6 است.)

نمونه‌ی زیر هم برای یک تصویر outdoor است:



		Name	Confidence	Depth
1		car 1	0.42	44.64
2		car 2	0.79	14.24
3		person 1	0.43	31.81
4		person 2	0.48	22.14
5		traffic light 1	0.29	45.76
6		traffic light 2	0.35	38.98
7		traffic light 3	0.66	46.58

فایل Graphic.py شامل اطلاعات مربوط به گرافیک و کلاس PyQt است. فایل DenseDepth.py شامل کدهای کلاس تخمین عمق و توابع لازم برای لود کردن مدل‌هاست که از git مربوطه استفاده شده‌است.

ترکیب مدل‌ها:

خطای مدل‌های به‌کار رفته به صورت زیر است: (تصویر اول، خطای شبکه تخمین عمق و تصویر دوم، خطای شبکه‌ی تشخیص اشیا است).

Method	$\delta_1 \uparrow$	$\delta_2 \uparrow$	$\delta_3 \uparrow$	rel \downarrow	rms \downarrow	$\log_{10} \downarrow$
Ours (scaled)	0.895	0.980	0.996	0.103	0.390	0.043

Measure	YOLOv5l
Precision	0.707
Recall	0.611
F1 Score	0.655
mAP	0.633
PC Speed (FPS)	58.82
Jetson Speed (FPS)	5

موازی:

مشخص است که در این حالت هیچکدام از شبکه‌ها اثری روی یکدیگر نگذاشته و هرکدام به تنهایی کار خود را انجام می‌دهد. بنابراین چه در حالت آموزش و چه در حالت تست، شبکه‌ها جداگانه عمل می‌کنند و خطا شبکه‌ی ترکیبی همانند جدول‌های بالا می‌شود.

سریال نوع یک:

در این حالت ابتدا تصویر را به شبکه‌ی تخمین عمق می‌دهیم. سپس متناسب با ورودی کاربر، محل‌هایی از تصویر را که در محدوده‌ی موردنظر نیستند، حذف کرده و نتیجه را به شبکه‌ی تشخیص اشیا می‌دهیم.

در این حالت کاری که می‌توان برای بهبود عملکرد شبکه‌ی تشخیص اشیا نسبت به حالت عادی انجام داد، این است که با توجه به کاهش تعداد پیکسل‌های تصویر، ابعاد grid های مورد پیشبینی را کاهش داد در حالی که تعداد grid ها نسبت به حالت عادی ثابت بماند. در این صورت، تراکم box های تخمینگر افزایش یافته و به این ترتیب دقت شبکه‌ی تشخیص اشیا بالا می‌رود. زیرا ابعاد این کار را هم در زمان آموزش و هم در زمان تست می‌توان انجام داد. دقت شود که در زمان آموزش برای بررسی خطای شبکه‌ی تخمین عمق، مستقیماً از خروجی همین شبکه برای سنجش خطا استفاده می‌کنیم.

با توجه به اینکه در این مدل سریال، ابتدا شبکه‌ی تخمین عمق مورد استفاده قرار می‌گیرد و خطای آن بدون توجه به شبکه‌ی تشخیص اشیا محاسبه می‌شود، پس انتظار می‌رود که نتیجه‌ی خروجی شبکه‌ی تشخیص عمق مشابه اعداد بالا باشد. اما برای شبکه‌ی تشخیص اشیا این موضوع متفاوت است. اولاً به خاطر افزایش تراکم تعداد box ها در تصویر، (در واقع به واسطه‌ی کاهش ابعاد تصویر، تعداد grid ها را ثابت می‌گیریم و ابعاد آن‌ها را کوچکتر انتخاب می‌کنیم تا قدرت تشخیص شبکه نسبت به تصاویر متراکم بیشتر شود). احتمال آنکه این box ها، box های ground truth پوشش دهند، بیشتر می‌شود. در نتیجه انتظار می‌رود که تعداد بیشتری از box های gt (ground truth) تشخیص داده‌شود و یا معادلاً IOU آن بیشتر باشد. در نتیجه recall افزایش می‌یابد. اما طبیعتاً چون تعدادی از box های

اضافه شده ممکن است مناسب نباشند و IOU آن‌ها با box های gt کم باشد، پس precision کاهش می‌یابد. البته باید دقت کرد که چون این افزایش تراکم box ها به صورت یکنواخت است، پس میزان کاهش precision چندان زیاد نیست و بعضی از box های اضافه شده نسبت به حالت تصویر بدون mask، همچنان دارای IOU بالایی هستند و برچسب مثبت می‌خورند. با توجه به این دو نکته، مساحت منحنی precision-recall افزایش یافته و معیار mAP هم زیاد می‌شود. بنابراین انتظار می‌رود که نتایج شبکه‌ی تشخیص اشیایی که با این شیوه آموزش دیده‌است، نسبت به جداول بالا بهبود داشته‌باشد. حتی اگر از یک شبکه‌ی تشخیص اشیا که به صورت عادی هم آموزش دیده، با این روش استفاده کنیم، باز هم دقت خروجی نسبت به مقادیر بالا بهتر خواهد بود.

اگر به توضیحات بالا دقت شود، می‌بینیم که با سریال کردن دو شبکه، خطای شبکه‌ی اول چندان دستخوش تغییر نمی‌شود. (مخصوصاً در روش پیشنهادی ما که خطای شبکه‌ی اول را هم مستقیماً سنجیدیم و از خروجی شبکه‌ی دوم استفاده نکردیم.) اما تأثیری که شبکه‌ی اول روی شبکه‌ی دوم می‌گذارد، باعث بهبود عملکرد شبکه‌ی دوم می‌شود. در واقع، شبکه‌ی اول به گونه‌ای باعث تغییر پارامترهای شبکه‌ی دوم شده و خروجی نهایی را بهبود می‌دهد.

سریال نوع دو:

در این حالت ابتدا تصویر ورودی را به شبکه‌ی تشخیص اشیا داده و پس از یافتن box های اشیا، از این اطلاعات در شبکه‌ی تخمین عمق استفاده می‌کنیم. همانند حالت قبل، خروجی شبکه‌ی تشخیص اشیا، مستقیماً با ground truth سنجیده شده و شبکه‌ی دوم یعنی تخمین عمق، تأثیری روی شبکه‌ی اول ندارد. اما این روش باعث بهبود شبکه‌ی دوم و خروجی عمق می‌شود.

نکته‌ی اصلی به کار رفته در این حالت، این است که در یک تصویر، انتظار می‌رود که عمق پیکسل‌های متناظر یک شی تقریباً مشابه باشد. اصطلاحاً پیکسل‌های مربوط به یک شی، تخت هستند. به همین منظور، در فرآیند آموزش شبکه‌ی دوم، علاوه بر تابع خطای مذکور در توضیحات شبکه‌ی DenseDepth، یک تابعی مانند واریانس از مقادیر مربوط به پیکسل‌های داخل هر box (تخمین زده شده توسط شبکه‌ی yolo) استفاده می‌کنیم. به این ترتیب شبکه به این سمت حرکت می‌کند که برای پیکسل‌های داخل هر box، عمق‌های مشابهی را خروجی دهد. به این ترتیب انتظار می‌رود که خروجی شبکه‌ی تشخیص عمق نسبت به اعداد بالا، بهبود یابد.

در زمان تست هم تقریباً مشابه بالا عمل می‌کنیم. یعنی علاوه بر تصویر ورودی، box های خروجی تشخیص اشیا را به آن می‌دهیم تا در خروجی، پیکسل‌های هر box دارای عمق نزدیکی باشند.

درباره‌ی نکات بالا، باید چندین مورد را در نظر گرفت. اولاً هر box ای که توسط شبکه‌ی تشخیص اشیا در خروجی ایجاد می‌شود، لزوماً تمامی آن توسط شی متناظر پوشانده نمی‌شود. به همین دلیل می‌توان به جای واریانس تمام

پیکسل های آن، صرفاً از واریانس پیکسل های یک box هم مرکز با آن و با ابعاد کوچکتر (مثلاً ۰.۴) در خطای نهایی استفاده کرد. برای از بین بردن وابستگی خطا به بازه ی عمق، به جای واریانس خود عمق، از واریانس لگاریتم آن استفاده می کنیم. زیرا لگاریتم نسبت به مقیاس خنثی است و فقط نسبت اعداد اهمیت دارد. (توضیح داده شده در معیارهای شبکه ی تخمین عمق)

نکته ی دیگر این است که شبکه ی تخمین عمق، ورودی ای غیر از تصویر ورودی نمی گیرد. به همین علت باید مکانیزمی را در نظر بگیریم که box های به دست آمده توسط شبکه ی تشخیص اشیا را به عنوان ورودی به شبکه ی تخمین عمق داد. به این منظور یک شبکه ی کمکی طراحی می کنیم که ابتدا تصویر اصلی و box ها را به عنوان ورودی گرفته و در خروجی یک تصویر اصلاح شده بدهد که ورودی شبکه ی تخمین عمق است. به این ترتیب در حین آموزش، هم شبکه های اصلی ما آموزش می بینند و هم شبکه ی کمکی. البته دقت شود که در این حالت امکان استفاده از شبکه هایی که به صورت عادی آموزش دیده اند، وجود ندارد.