Monocular Multilayer Layout Estimation for Warehouses

Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science in Computer Science and Engineering by Research

by

ANURAG SAHU
2018121004
anurag.sahu@research.iiit.ac.in

International Institute of Information Technology
(Deemed to be University)
Hyderabad - 500 032, INDIA
MARCH 2023

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "Monocular Multilayer Layout Estimation for Warehouses" by Anurag Sahu, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date

Adviser: Prof. K. Madhava Krishna

To People who dare to Leap.

Acknowledgments

I would like to express my sincere gratitude to my professor, Prof. K. Madhav Krishna, for their invaluable guidance and support throughout my research project. Their expertise and knowledge in the field were instrumental in helping me to shape my research, and their constant encouragement and constructive feedback helped me to stay motivated and focused. I am deeply thankful for the time and effort they dedicated to my project, and I am truly grateful for the opportunity to work with such a talented and inspiring mentor. I would also like to thank Prof. Ravi Kiran Sarvadevabhatla, for his insightful reviews that has widened my perspective towards my work, his critical comments has helped to enhance the overall quality of the thesis.

I deeply thank all my colleagues from the RRC lab for the refreshing discussions, working towards the deadlines, and all the fun we had together that made the time memorable. Special thanks to my Teammates Shashwat Nigam, Puru Gupta, Avinash Prabu, Tanvi Karandikar, Pranjlai Pathre for their continuous motivation, rigorous discussions, help conducting experiment and discussions to keep the morale up.

I want to thank my friends Deeksha, Niharika, Samhita, Samanvaya, Rajashekhar, Sai Siddharth, Krishna for always being there for me and keeping the mood light. I want to thank my Father for teaching me how to keep up during tough time, aim high regardless of the situation, to be sane when things are going insane and finally for the unconditional love, respect, trust and support he has given me throughout this journey.

Abstract

With the booming online market, managing the warehouse inventory is one of the most essential and challenging tasks. The management of inventory will be more efficient if it is automated using robots. The robots can work faster than humans, the robots work at a constant speed with no breaks, and do tasks in more repetition than humans like fetching inventory from warehouse. But for the robots to perform tasks like putting objects in racks, fetching objects from rack, re-organising the racks to make more space they need to have an understanding of the warehouse environment.

To understand the warehouse environment the robots needs to create a 3D map of the warehouse consisting space to move for the robot as well as identify the racks and boxes so that its able to plan the execution of tasks. Towards solving this problem in this thesis we address problem of freespace estimation for rack shelves. Given a monocular RGB image captured from a camera mounted on a robotic arm. We aim to predict the Top-view and Front-view layouts so as to create a 3D reconstruction of rack and objects present in the Monocular RGB image.

We propose a simple yet effective network architecture RackLay, which takes a monocular RGB image as input and outputs the Top-view and Front-view layout of all the shelves comprising the rack visible in the image. The Network can learn two kinds of layout representations, one in the canoncial frame centered on the shelf, called the shelf-centeric layout and the other in a frame with respect to the camera, called the ego-centric layout. Apart from portraying the versatility of the network, they lend to various useful applications.

Since there are very few publicly available datasets for warehouse settings, we also introduce the synthetic data generation pipeline termed as WareSynth, which can be used to generate 3D warehouse scenes, automate the process of data capture and generate corresponding annotations. WareSynth can be used for various tasks such as 2D/3D object detection, semantic/instance segmentation, layout estimation, 3D scene navigation and mapping, 3D reconstruction etc. The same pipeline can also be modified to other kind of scenes such as supermarkets, greenhouses by changing the database of objects and placement parameters hence this pipeline open gates for further research in similar environments.

Contents

List of Figures

List of Tables

Table Page

Chapter 1

Introduction

Figure 1.1 A Robotic Arm fetching object from warehouse rack in a real warehouse(left) and synthetic warehouse(right).

The need for warehouse automation is growing, and the future looks like it will involve a fleet of robots managing warehouses with minimal human intervention[11]. Currently, approximately 30% of warehouses operate without any Warehouse Management Systems (WMS) [2] and the warehouses which have WMS are software based that are managed by humans where they manually need to Feed/Scan the inventory incoming and outgoing from the warehouse. Based on information fed the system tells if there is space present at a particular location but this information is subject to human error. These types of management systems are unorganized, and there is a need for an automatic system that can automatically reason about the space in the warehouse without human intervention.

If the robots have the ability to get the 3D space in the rack they can be used for automatically mapping the warehouse to find out the free space and more importantly the robots can also manipulate the warehouse, The ability to manipulate the warehouse is a key advantage because it allows robots to optimize the warehouse for space, reducing the space needed to store the same number of boxes.

For the Robots to understand the warehouse they need to understand the environment around them and most importantly the racks and objects kept on the rack. The robots should be able to a 3D Reconstruction of the warehouse and using these 3D reconstructions they can get How much **free space** is present? Where is it present? Can it be manipulated in a way to accommodate another box? This will also enable the robot to plan a path through the warehouse that allows it to move around the boxes without damaging them. For example, a drone might need to decide whether it can simply pass through the racks or if it should go above the racks.

After the introduction of deep learning-based methods in robotics, autonomous driving has gained a lot of traction, but warehouse management systems are still overlooked. The lack of progress in research for warehouse automation can be attributed to the lack of datasets for warehouses, while there are many datasets for driving scenes such as KITTI, Oxford Radar RobotCar Dataset, Waymo Open Dataset, and Landmarks. This has created a need for a warehouse dataset to kick-start deep learning-based research in warehouse management systems.

1.1 Problem Addressed

In this thesis, we address the untackled problem of freespace estimation for rack shelves. We aim to construct a 3D representation of the rack scene in monocular settings by combining the top view layout and front view layout. Hence We propose a simple yet effective network architecture *RackLay*[1], which takes a monocular RGB image as input and outputs the *top-view* and *front-view* layout of all the shelves comprising the rack visible in the image.

The network can learn two kinds of layout representations: one in the canonical frame centered on the shelf, called the *shelf-centeric* layout, and the other in a frame with respect to the camera, called the *ego-centric* layout. Apart from portraying the network's versatility, they lend to various useful applications.

It is important to note that the problem is not immediately reducible to any standard formulation of object recognition, layout estimation, or semantic segmentation. Objects on rack shelves are amenable to semantic segmenation [62] or object detection [28]. However, this is not the case for racks themselves, which appear as occluded, diffused and thin structues. For very similar reasons, existing approaches cannot be trivially adapted for localizing rack shelves,as shown in our baseline comparisons. Unlike standard layout formulations that estimate the layout with reference to a single dominant plane (for example ground plane) [61], warehouse rack shelves are disconnected and distinct planer segments present at multiple heights relative to the ground plane. Thus, an important novelty of our formulation is the

[1]Project page: https://avinash2468.github.io/RackLay/

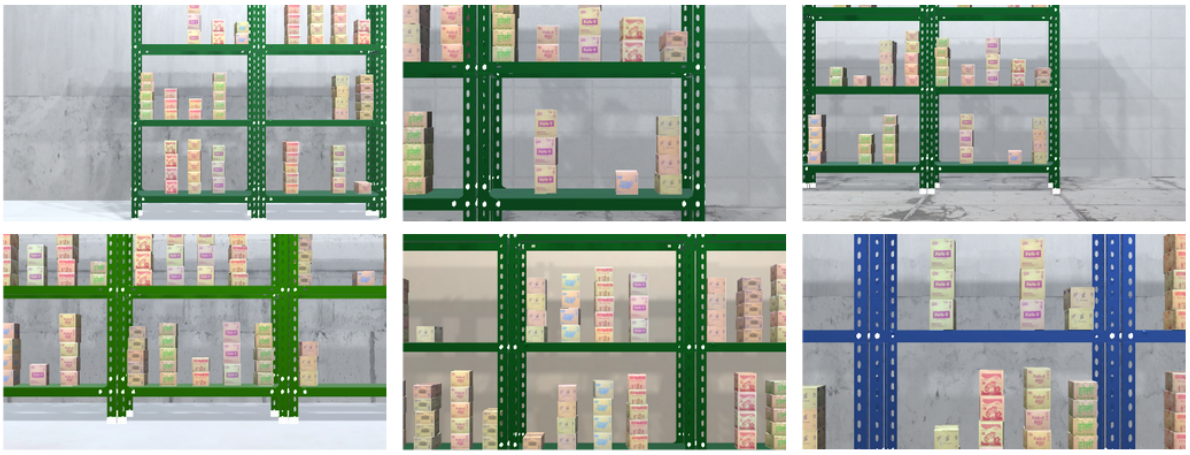

Figure 1.2 Warehouse Scenes with multiple shelves, thin rack structures.

adaption of deep architectures to the problem of layout estimation over multiple shelves the constitute a rack and contents thereof.

1.2 Contribution

Specifically, the thesis contributes as follows:

- We propose a novel architecture (Sec. 3.1), the keynote of which is the multi-channel decoder that infers layouts for each shelf of a given rack, located at different heights. Our network is versatile and can produce *shelf-centric* or *ego-centric* layouts (Sec. 3.0.1, 3.0.2), according to the contingent application. This is achieved through a multi-channel decoder architecture, wherein each channel provides the layout for a particular shelf in the rack. In this way it contrasts itself with a variety of prior art layout architectures that provide only for a single layout of a given scene.

- We open-source a flexible data generation pipeline *WareSynth* with domain randomization capabilities that enables sim2real transfer. We also release relevant instructions that enable the user to create and customize their warehouse scenes and generate 2D/3D ground truth annotations needed for their task automatically, as discussed in Sec. 5. We release the *RackLay* synthetic dataset consisting of 20k RGB images and 500 Real world Images along with layout annotations of shelves and objects in both the top and front-view. This does not restrict or limit the user to our dataset alone but provides for possibilities to create new datasets with the ability to customize as required.

- We show results on real world scenes, which demonstrates the sim2real transferability of our approach. Specifically, the inevitability of such synthetic datasets is portrayed as

accurate layouts for real warehouse scenes are obtained by fine tuning with a minimal number of real images.

- Further we show several applications using these layout representations. These include 3D free volume estimation and obtaining SDF representation using such layouts. Layout enabled camera pose estimation is a novel application discussed in Sec. 6. Our approach can also be further used for trajectory planning, collision avoidance, Warehouse mapping, precise placement tasks which are vital components of any autonomous warehouse system.

Chapter 2

Related Works

In recent years, learning scene layouts and obtaining volumetric representations directly from an RGB image has garnered a lot of interest. Deep learning methods have become more reliable and accurate for many computer vision tasks like object detection, semantic segmentation and depth estimation. Interpreting scene layouts and obtaining volumetric representations from a monocular input is a challenging task therefore even a combination of the fundamental solutions like object detection, semantic segmentation and depth estimation does not suffice for higher-order tasks like shelf-layout estimation in warehouse management systems, which requires multi-layer top-view layout estimation. To that extent, we summarize the existing approaches and differentiate our method from the rest.

2.1 Indoor scene understanding

Room layout estimation from a single image [85, 89] is a popular problem in the context of indoor 3D scene understanding. There have also been a few approaches for amodal perception as well [38, 41]. Indoor scene understanding can rely on strong assumptions like a Manhattan world layout, which works well for single room layouts.

2.2 Detection methods

There are several methods that do 3D Object detection that exclusively use lidar or a combination of camera and lidar sensors. However, there are only a handful of approaches comprising a region-proposal stage and a classification stage.

Another category of approaches map a monocular image to a bird's eye view representation [61], thereby reducing the task of 3D object detection to that of 2D image segmentation. Recently, BirdGAN [70] leveraged adversarial learning for mapping images to bird's eye view, where lidar object detectors such as [8] were re-purposed for object detection.

Such techniques usually require a pre-processing stage (usually a neural network that maps an image to a bird's eye view) after which further processing is applied. On the other hand, we demonstrate that we can achieve significantly higher accuracy by directly mapping from the image space to objects in bird's eye view, bypassing the need for a pre-processing stage altogether.

More notably, all the above approaches require a post-processing step that usually involves non-maximum suppression / thresholding to output object detections. The proposed methods neither requires pre-processing nor post-processing and they directly estimate scene layouts that can be easily evaluated(or plugged into other task pipelines).

2.3 Bird's eye view (BEV) representation

BEV semantic segmentation has been tackled mostly for outdoor scenes. Gupta et al. [26] demonstrate the suitability of a BEV representation for mapping and planning. Schulter et al. [66] proposed one of the first approaches to estimate an occlusion-reasoned bird's eye view road layout from a single color image. They use monocular depth estimation and semantic segmentation to aid their network that predicts occluded road layout. Wang et al. [81] build on top of [66] to infer parameterized road layouts. Parametric models might not account for all possible scene layouts, whereas our approach is non-parametric and thus more flexible. MonoOccupancy [48], uses a variational autoencoder (VAE) to predict road layout from a given image, but only for the pixels present in the image. MonoLayout [50], can be trained end to end on colour images, reasons beyond occlusion boundaries and does not need to be bootstrapped with these additional inputs. We predict the occlusion-reasoned occupancy layouts of multiple parallel planes(layers), with varying height, from a single view RGB.

2.4 Warehouse Datasets:

There are very few datasets publicly available for warehouse settings. Real-world datasets like LOCO [51] exist for scene understanding in warehouse logistics, but they provide a limited number of images, along with corresponding 2D annotations. Due to the difficulty in acquiring the 3D ground truth information from real scenes, there are no real warehouse datasets which provide information about the objects in the scene and their relative positions and orientation. For 3D deep learning applications, a large amount of diverse data along with 3D ground truth information is required. There are only a handful of general-purpose synthetic data simulators for generating photo-realistic images, like NVIDIA Isaac [1], which provide warehouse scenes. However, they can't be modified easily to generate annotations needed for the task at hand.

Chapter 3

Racklay: Multi-Layer Layout Estimation for Warehouse Racks

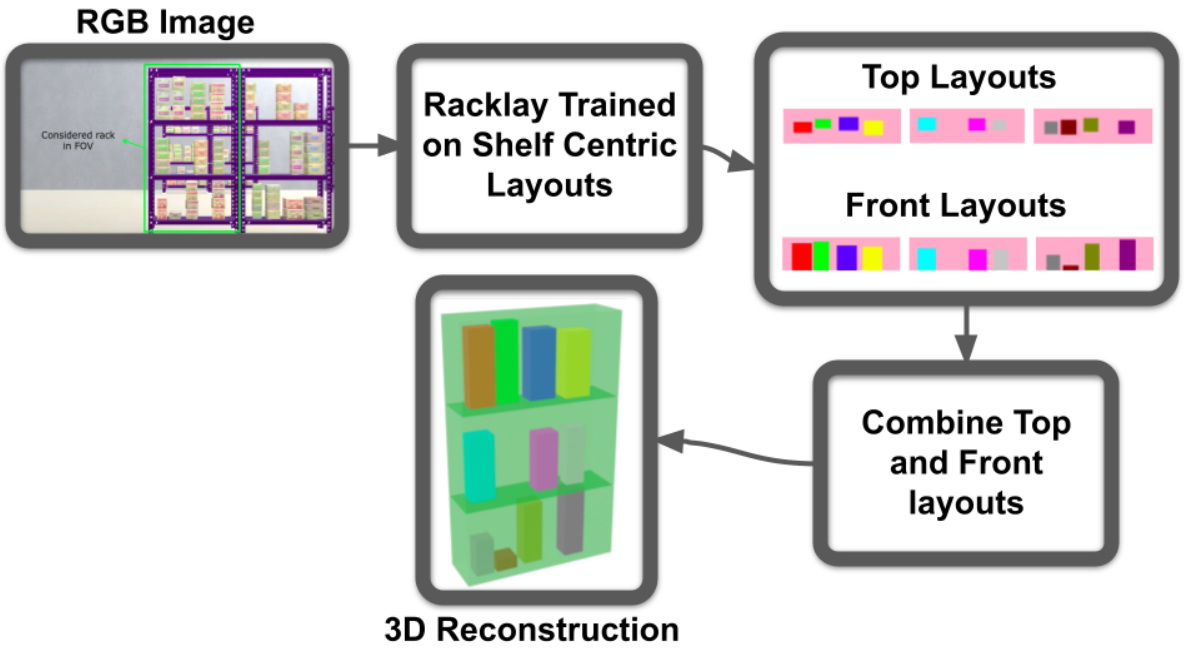

Figure 3.1 Diagram explaining how Racklay can be used to get 3D Reconstruction of a particular rack using the shelf-centric layouts.

Given a monocular color image $\mathbb{I}$ of a warehouse rack in perspective view, we aim to predict the top-view (bird's eye view) layout for each shelf of a rack, that stands within the range of detection **R** and is visible in the image. We consider **R** to be a rectangular area in a top-down orthographic view of the scene. The camera is placed at the mid-point of the lower side of the rectangle, directly facing the racks such that the image plane is orthogonal to the ground plane. (Fig. 3.2). The coordinate frame is such that the X axis points to the right, Y axis points upwards and Z axis points forward. The coordinate frame is positioned

at the rack center for shelf-centric layouts and the camera center for the ego-centric layouts (Fig. 3.2). The top-view layouts are hence parallel to the X-Z plane, at corresponding shelf heights.

Concretely, we wish to learn a labelling function that generates a top-view layout for all shelves of the rack, within a region of interest Ω. Our network can be used to predict two kinds of layouts dependent on the contingent application, as described:

3.0.1 Shelf-centric layout

We consider Ω to be a rectangular area, positioned such that its center coincides with the concerned rack's center, as shown in Fig. 3.2. This layout is hence with respect to the rack and is view-point agnostic. Shelf-centric layouts are useful for logistical tasks like free-space estimation.

3.0.2 Ego-centric layout

We consider Ω to be the same as the range of detection $\mathbf{R}$. This layout is with respect to the camera position (as in (iii) of Fig. 3.2), from which we can infer X and Z(depth) coordinates of the rack in the camera frame. Ego-centric layouts are slightly more complex for the network to learn (as the network has to learn the X and Z coordinates along with the layout, as discussed in Sec. 4.3) and prove useful for tasks like robotic navigation, camera pose estimation etc.

The labelling function must produce labels for all the points in Ω, regardless of whether or not they are imaged in $\mathcal{I}$. The points in Ω are labelled as *occupied*, *unoccupied* or *background*. In our problem context, a pixel with *occupied* label denotes the presence of an object (boxes, cartons, etc.) at that place on the shelf, and the *unoccupied* label denotes that the shelf is empty at that pixel. Label *background* denotes the area that is not occupied by the shelf.

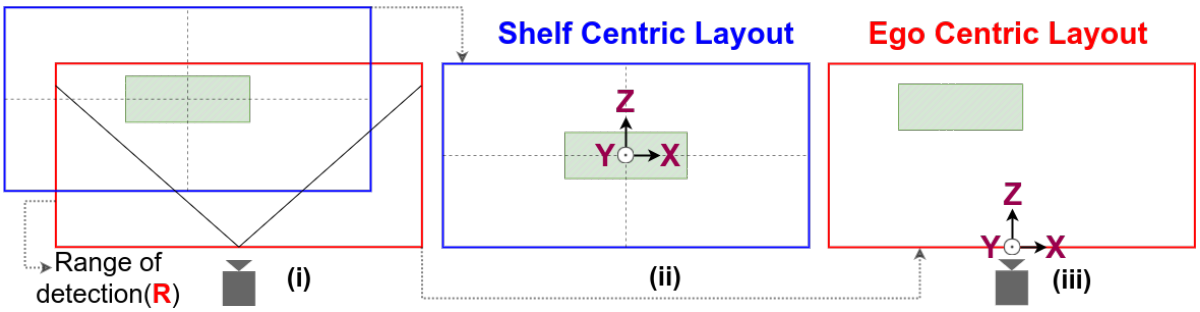

Figure 3.2 Top-view representation of the *shelf-centric* (ii) and *ego-centric* (iii) layouts for a given position of a shelf (in green) (i), and the reference coordinate frames for the same. Note the difference in the coordinate frame position for both layouts with respect to the shelf.

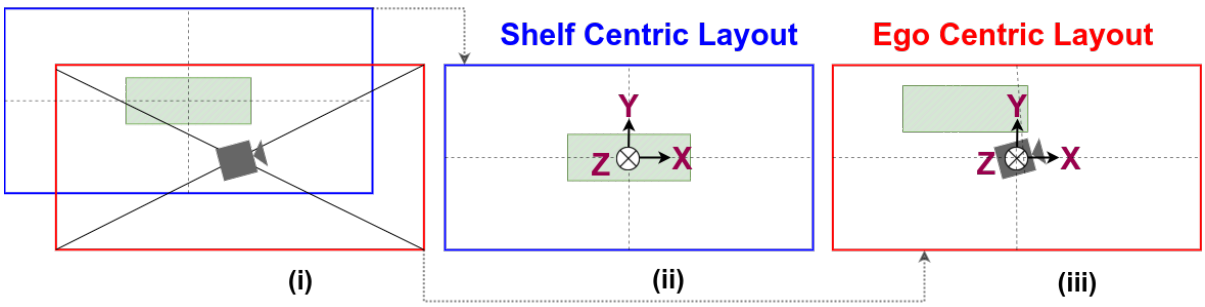

Figure 3.3 Front-view representation of the *shelf-centric* (ii) and *ego-centric* (iii) layouts for a given position of a shelf (in green) (i), and the reference coordinate frames for the same. The camera is positioned such that it is looking into the plane. Note the difference in the coordinate frame position for both layouts with respect to the shelf.

Front-view layouts: As an additional task, we aim to learn a similar labelling function for the points in the front-view layout, which is orthogonal to the top-view layout. The camera is at a certain height and is placed such that it is orthogonally viewing the concerned rack. Here, we classify the empty inter-shelf area as *unoccupied*. For ego-centric layouts, we consider the camera to be at a height which corresponds to the center of the layout (Fig. 3.3). This helps us infer the Y coordinate (height) of the shelf with respect to the camera.

3D reconstruction and SDF representation: Using a combined representation from the top-view and front-view ego-centric layouts, we obtain a 3D reconstruction of the rack in the camera frame (Fig. 6.3), which can be further used for 3D spatial reasoning tasks. As explained above, we infer the X, Z coordinates from the top-view and the Y coordinate from the front-view. We further discuss the applications in Sec. 6.

3.1 RackLay Architecture

The RackLay architecture (Fig. 3.4) comprises of the following components:

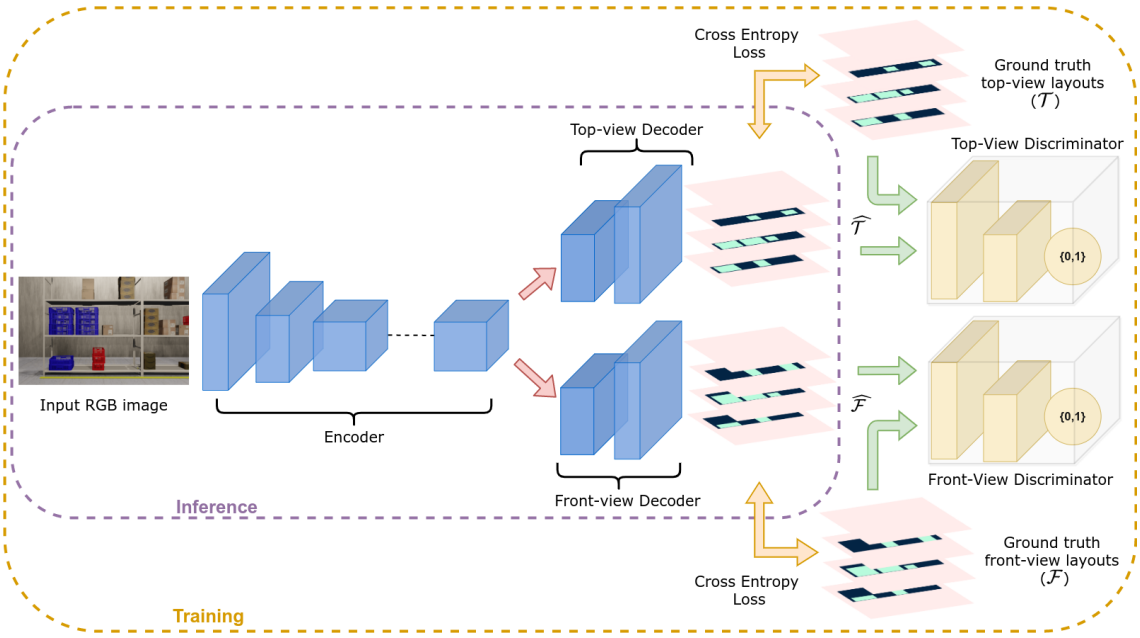

Figure 3.4 The figure shows architecture diagram for *RackLay-D-disc*. It comprises of a context encoder, multi-channel decoders and adversarial discriminators.

3.1.1 Context Encoder

The Context Encoder retrieves relevant 3D scene and semantic features for layout estimation, from the monocular input $\mathcal{I}$. This provides a representation that enables differentiation between *occupied*, *unoccupied* and *background* scene points. The encoder learns lower level feature representations for the high-dimensional input image and generate multi-scale feature maps. The encoded features enable the model to infer about the semantics and the depth of the scene, which is essential for layout prediction in bird's eye view. Our context encoder is built on top of a ResNet-18 encoder. The network usually takes in RGB images of size 3*512*512 as input, and produces a 512*32*32 feature map as output. In particular we use the ResNet-18 architecture without bottleneck layers.

3.1.2 Top-view Decoder

The Top View Decoder generates layouts for each shelf of the rack that is visible in the image, from the representation learned by the context encoder. It decodes the context from the feature extractor (context encoder) via a series of deconvolution and upsampling layers that map the context to a semantically rich bird's eye view. The decoder outputs an $\mathcal{R} \times \mathcal{D} \times \mathcal{D}$ grid which represents the top-view layout $\mathcal{T}$, where $\mathcal{R}$ is the number of output channels and $\mathcal{D}$ is the resolution for the output layouts. Each channel denotes a shelf and is a per-

pixel label map of the corresponding top-view layout. Here we emphasize the novelty of the design choice: using a multi-channel output to predict occupancy layouts that lie at different heights in the rack.

3.1.3 Discriminator

We introduce adversarial regularizers (discriminators) . The layouts estimated by the front view decoder and top view decoders are input to these patch-based discriminators. The Discriminators regularize the distribution of the output layouts to match a prior data distribution of conceivable scene layouts.

The discriminator architecture is inspired by Pix2Pix [33]. We found the patch based regularization in Pix2Pix to be much better than a standard DC-GAN [58]. So, we use patch level discriminators that contain four convolution layers(kernel size 3*3, stride 2), that outputs an 8*8 feature map. This feature map is passed through a tanh nonlinearity and used for patch discrimination.

To predict both views (top, front) from a single network, we extend the above architecture by adding an identical decoder followed by a discriminator to the existing encoder, which predicts front-view layout for each shelf ($\mathcal{F}$), analogous to the top-view layout ($\mathcal{T}$).

3.2 Loss function

The network parameters ϕ, ψ, θ of the context encoder, the top-view decoder and discriminator respectively are optimized using stochastic gradient descent. The Loss term is a sum of three loss terms.

$$\min_{\phi,\psi,\theta} \mathcal{L}_{sup}(\phi,\psi) + \mathcal{L}_{adv}(\phi,\psi,\theta) + \mathcal{L}_{discr}(\phi,\psi,\theta)$$

Here, (L_{sup}) refers to the loss which penalizes deviation of the predicted layout labels ($\widehat{\mathcal{T}}$) from corresponding ground-truth values ($\mathcal{T}$), here f is a per-pixel cross entropy loss function.

$$L_{sup}(\widehat{\mathcal{T}};\phi,\psi) = \sum_{j=1}^{N}\sum_{i=1}^{\mathcal{R}} f\left(\widehat{\mathcal{T}}_i^j, \mathcal{T}_i^j\right)$$

$$[0.75]L_{adv}(\widehat{\mathcal{T}};\phi,\psi,\theta) = {}_{\theta\sim p_{fake}}\,[(\widehat{\mathcal{T}}(\theta)-1)^2]$$

$$[0.75]L_{discr}(\widehat{\mathcal{T}};\theta) = {}_{\theta\sim p_{true}}\,[(\widehat{\mathcal{T}}(\theta)-1)^2] + {}_{\theta\sim p_{fake}}\,[(\widehat{\mathcal{T}}(\theta)-0)^2]$$

where $\widehat{\mathcal{T}}$ and $\mathcal{T}$ are the predicted and the ground truth top-view layouts for each shelf, $\mathcal{R}$ is the maximum number of shelves considered and N is the mini-batch size.

L_{sup} is the per-pixel cross entropy loss which penalizes deviation of the predicted layout labels ($\widehat{\mathcal{T}}$) from corresponding ground-truth values ($\mathcal{T}$). The adversarial loss L_{adv} promotes

the distribution of layout estimates from the top-view decoder (p_{fake}) to be similar to the true data distribution (p_{true}). The loss term L_{discr} is the discriminator update objective [25]. Analogous loss terms exist for front-view layout estimation as well.

Chapter 4

Experiments

Until now we have talked about how learning scene layouts and obtaining volumetric representations directly from an RGB image has garnered a lot of interest, How layout estimation from a single image has been a propular problem and we have proposed an Architecture that Given a monocular color image of a warehouse rack predicts the Birds eye view of the shelf along with the objects kept on the shelf.

Following that in this chapter we describe the training procedure, the dataset we have used, the Evaluations methods and compared the performance of Racklay with the other methods to estimate the layout of a warehouse rack.

4.1 RackLay Dataset

For training and testing our network, using *WareSynth*, we generated 2 datasets, a simple dataset with 10k images and a complex dataset with 20k images[1]. We describe and display results for our more diverse and complex dataset consisting of 20k images, which we split into 16k/2k/2k for train/test/validation. We randomized various parameters during data generation to add diversity and complexity in the scenes such that the resulting scenes mimic counterparts from real warehouses and also enable transfer to real scenes. We describe these randomizations below.

4.1.1 Domain Randomization:

We have used the following domain randomization techniques in the dataset used by Racklay to train. We have carefully picked these randomization techniques so that the network is not just able to predict similar looking synthetic scenes but it is able to predict the layouts for real world scenes with some fine-tuning. We show the randomizations we introduce using randomly selected images from our dataset through Fig. 6.1:

[1]Download RackLay dataset: http://bit.ly/racklay-dataset

- Boxes have random sizes, textures, rotation about the vertical axis, colors and reflective properties.
- Box placement varies from dense to sparse.
- Color and texture of racks is randomized.
- Height to which boxes are stacked vertically is randomized .
- Background is either a wall, or a busy warehouse.
- Color, textures of floors, walls are randomized.
- The position of the camera with respect to the rack is varied within a range. This affects the number of shelves visible in the image from 1 to $\mathcal{R}$. For our dataset, we set $\mathcal{R}$=3 (observe all rows of Fig. 6.1).

We find that this large diversity in the dataset has enabled the network to not overfit on the domain of the synthetic data, but rather learn features that can be transferred to real-world scenes as demonstrated in Section 4.5.

4.2 Evaluated Methods and Metrics

We have compared the following variants of RackLay:

- *RackLay-S*: It has a Single decoder architecture, can be used to predict either front-view layout *or* top-view layout.
- *RackLay-D*: This architecture has two decoders in the architecture, this network can be used to predict both the front-view layout *and* the top-view layout.
- *RackLay-S-disc*: This is a Single decoder architecture with discriminator, this architecture can be used predict either the front-view layout *or* the top-view layout.
- *RackLay-D-disc*: This network architecture has two decoders with discriminators and the end of both decoders, and it predicts both front-view layout *and* top-view layout.

We evaluate the layouts predicted by all these networks on two types of criteria which are Mean Intersection-Over-Union (mIoU) and Mean Average-Precision (mAP). These metrics are calculated on a per-class basis hence we have these matrices values for shelves (represented by gray pixels in layout) and similarly for boxes (represented by white pixels in layout).

4.3 Results

[2]

We first trained *RackLay-S* for top-view and front-view separately. Having achieved superior results compared to baselines, we trained *Racklay-D* for both front-view and top-view simultaneously. We observed performance gains as discussed in Sec. 4.6. We further trained *Racklay-D-disc* to capture the distribution of our layouts which led to the best results (discussed in Sec. 4.6).

In Fig. 4.3, observe how our best network *RackLay-D-disc* is able to estimate both *shelf-centric* and *ego-centric* layouts on our domain-randomized dataset. We find that the ego-centric results are at par with shelf-centric results qualitatively (Fig. 4.3) but slightly lesser quantitatively (Table 4.1). This is because of the additional information (position of the concerned rack) that the network has to learn. Note that since the ground truth ego-centric layouts encode the position of the rack of interest (explained in Sec. 4.1), the quantitative scores also reflect the effectiveness of *RackLay-D-disc* at estimating the position of the rack with respect to the camera from a single image.

4.4 Comparison with baselines

4.4.1 PseudoLidar-PointRCNN:

We perform 3D object detection using PointRCNN [68] on a PseudoLidar input [80]. The PointRCNN architecture was designed for detecting objects on a road-like scene, their approach assumes a single dominant layer (the ground plane). This is an assumption used by many methods designed for bird's eye view, and hence may not perform well for indoor scenes where multiple objects are scattered at different heights relative to the ground plane. We observed that the success enjoyed by PointRCNN does not translate in the presence of multi-layer data. Their network is able to identify only the bottom shelf and objects kept on it due to which we cannot estimate the height between two shelves. Therefore, we report metrics only for the bottom shelf layouts (refer Table 4.1) and do not report front-view shelf layout scores. This again highlights the importance of our work because we reason about bird's eye view representation for multiple layers, rather than a single dominant layer.

Also, the approach fails to recognise objects when they are very closely spaced as the network is designed for outdoor LiDAR scans where objects are well separated in space. Another reason for our superior performance relative to PointRCNN is our approach's accuracy in estimating bounding box rotation (wrt vertical axis). This factor is crucial since minor misalignments in the rotation induce significant errors in predicted layout, especially when

[2]More results: https://avinash2468.github.io/RackLay/

	Top View				Front View			
	Rack		Box		Rack		Box	
Method	**mIoU**	**mAP**	**mIoU**	**mAP**	**mIoU**	**mAP**	**mIoU**	**mAP**
RackLay-D-disc	**98.48**	**99.24**	**94.46**	**97.32**	**98.89**	**99.51**	97.62	98.79
RackLay-D	96.96	99.09	92.67	96.02	98.29	99.33	96.23	98.35
RackLay-S-disc	94.57	97.32	91.71	96.32	97.60	98.61	**97.67**	**98.93**
RackLay-S	93.02	98.61	91.61	96.07	94.30	98.09	95.11	97.56
PseudoLidar-PointRCNN[68]	73.28	77.40	55.77	81.26	–	–	63.05	89.45
MaskRCNN-GTdepth[28]	–	–	35.57	47.44	–	–	76.48	82.48
RackLay-D-disc (Ego-centric)	97.04	98.13	92.06	96.06	98.48	99.09	97.87	98.93

Table 4.1 Quantitative results: We benchmark the 4 different versions of our network on shelf-centric layouts- *RackLay-S*, *RackLay-S-disc*, *RackLay-D* and *RackLay-D-disc*, along with two baselines- *PseudoLidar-PointRCNN*[80, 68] and *MaskRCNN-GTdepth*[28] (as described in Sec. 4.2). We also show the results for *RackLay-D-disc* (the best performing network on shelf-centric) on the ego-centric representation of the layouts. Note that *RackLay-S* and *RackLay-S-disc* are single decoder models and hence cannot predict top-view and front-view simultaneously. The top-view and front-view results displayed for each of these two models were trained separately.

the objects are small and closely placed. Also, PseudoLidar based inputs don't allow the use of deep feature encoders like ResNet[27] needed to extract the rich visual features necessary for hallucinating amodal scene layouts.

4.4.2 MaskRCNN:

We also compare with a classical approach wherein we use instance segmentation from MaskRCNN[28] and pair it with ground truth depth-maps. We project detected boxes to 3D shelf-wise, as boxes on a particular shelf will have similar vertical image coordinate. We then take a projection on the horizontal plane to obtain the box layouts for each shelf, by computing a convex hull for each box. Since this approach can only reason about visible points, it is clear from Table 4.1 that our network performs much better as it is able to perform amodal perception and is able to complete shapes and parts of the layout unseen in the input

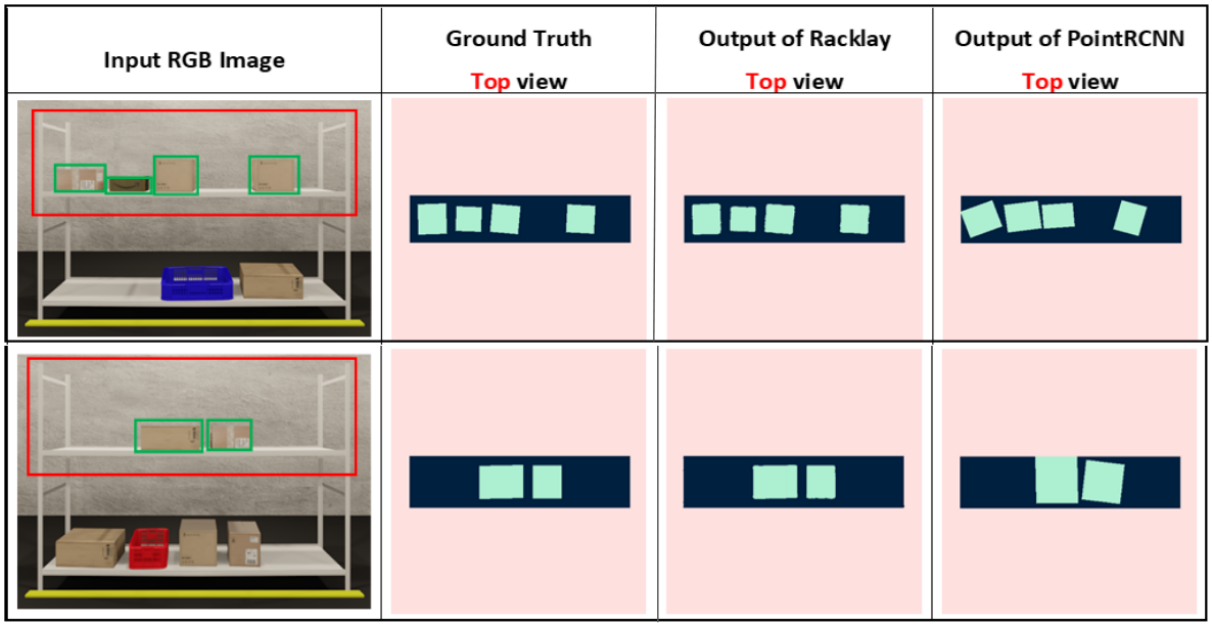

Figure 4.1 Comparing layouts produced by Racklay vs the layouts produced by PointRCNN. Note that the boxes in PointRCNN is able to predict the layouts only for one shelf where the size and the rotation of boxes are very different when compared to Ground Truth, where as the layouts produced by racklay are very close to the Ground Truth.

image. As discussed in Sec. 1, MaskRCNN fails to predict segmentation maps for thin structures like shelves with good accuracy.

4.5 Real World Results

To test the performance of our network on real-world data, we use two real-world datasets. The scene for the first dataset, *RRC_dataset*, was set up within our lab. It consists of 442 training images and 100 testing images. The 2nd dataset, the *rapyuta_dataset*, was collected from the Rapyuta warehouse [3] in Japan. It consists of 250 training images and 50 testing images. These datasets are further augmented using horizontal flip and color jitter. To test the performance of our network on the real world test images we use three methods of training:

1. Using only the 16,000 synthetic training images as described in Sec. 4.1. (results in row 1 of Table 4.2)

2. Using only the respective real world dataset's real world images. (results in row 2 of Table 4.2)

	Top View				Front View			
	Rack		Box		Rack		Box	
Training Method	**mIoU**	**mAP**	**mIoU**	**mAP**	**mIoU**	**mAP**	**mIoU**	**mAP**
Synthetic	78.19	93.66	77.30	91.43	85.47	97.69	81.01	94.34
Real	76.98	93.80	80.07	92.20	88.42	97.79	81.20	93.04
Synthetic + Real	**95.05**	**97.46**	**92.47**	**97.61**	**95.15**	**99.29**	**94.73**	**98.31**

Table 4.2 Results on real-world data: Here, we showcase the performance of *Racklay-D-disc* on real-world data.

3. Using a combination of the two methods: train with the large synthetic data and finetune using the real-world data. (results in row 3 of Table 4.2)

The first two methods, though perform well, are not at par with the results in Table 4.1. The last method, however performs the best and is at par with the results in Table 4.1. *This shows that training the network on large amounts of synthetic data followed by finetuning with real world data is the best strategy. This also highlights the importance of our synthetic data generation pipeline when the number of real world images are low.* The results of the final model on the two real-world datasets are shown in Fig. 4.2.

4.6 Ablation studies

We conduct ablation studies to analyze the performance of various components in the pipeline.

1) Multi-task learning: We observed that using two decoders to estimate both top-view and front-view layouts such that the encoder becomes the shared representation leads to a faster converging loss and a quantitative improvement (Table 4.1) in performance for almost all cases. There is a considerable improvement qualitatively as shown in Fig. 4.4, the network better predicts the space between closely spaced boxes and avoids spurious noise. Making the network learn these two tasks together forces it to learn the relevant features related to the occupancy much faster and improves the performance metrics.

2) Adversarial learning: We add a discriminator at the end of both the decoders to capture the distribution of *plausible* layouts. We observed a considerable improvement both quantitatively (Table 4.1) and qualitatively (Fig. 4.5). We obtain much sharper and realistic layouts. Most notably, the use of a discriminator reduces stray pixels which are misclassified as boxes and outputs cleaner box-like structures. It also helps in predicting the space between closely spaced boxes.

4.7 Extended Layouts

Throughout this thesis we have experimented with one rack in focus, which gave the layouts for only one rack hence we are able to reason that one rack, but as a further extension when we want to reason about the other racks then the camera has to me moved so that the other rack is in focus although that rack is present partially in the previous image. To get the layouts of all the racks present in the image we have experimented with a dataset where all the racks are present in layout, This task is harder for the network as compared previous task where just one rack was in focus because now with all the racks present in the layout the network has to predict the racks which are partially visible and the same area of layout now has to accommodate a wider layout hence the number of pixels representing the rack and boxes are lesser than compared to the previous case where one rack was in focus.

We trained the network with 10k Data one with layouts having only one rack and the other having all the racks in layout. The results of the network predicting all the racks are inferior as compared the the network predicting only one rack. The performance of the network predicting all the racks in layout can be improved by increasing the number of datapoints, we trained the network with 20k images instead of 10k images and the results improved.

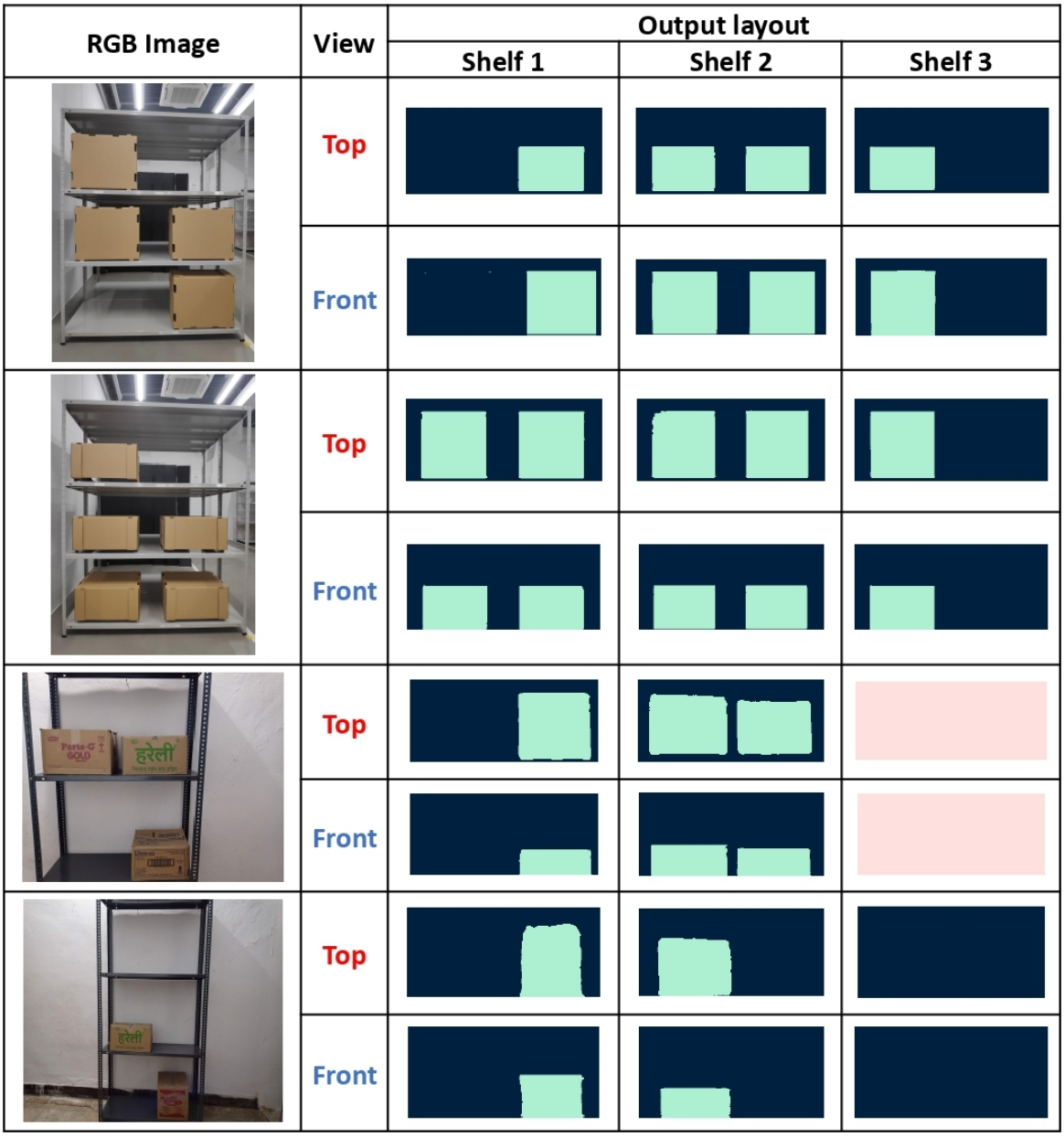

Figure 4.2 Real World Images: Here, we display shelf-centric results on real-world data. The first 2 rows showcase the results on rapyuta dataset and the last two rows showcase results on RRC dataset. We can observe that the network performs well in two different environments with different lighting conditions, rack types and box types in the real-world. Note that we do not include the background class around the shelf prediction for the sake of brevity.

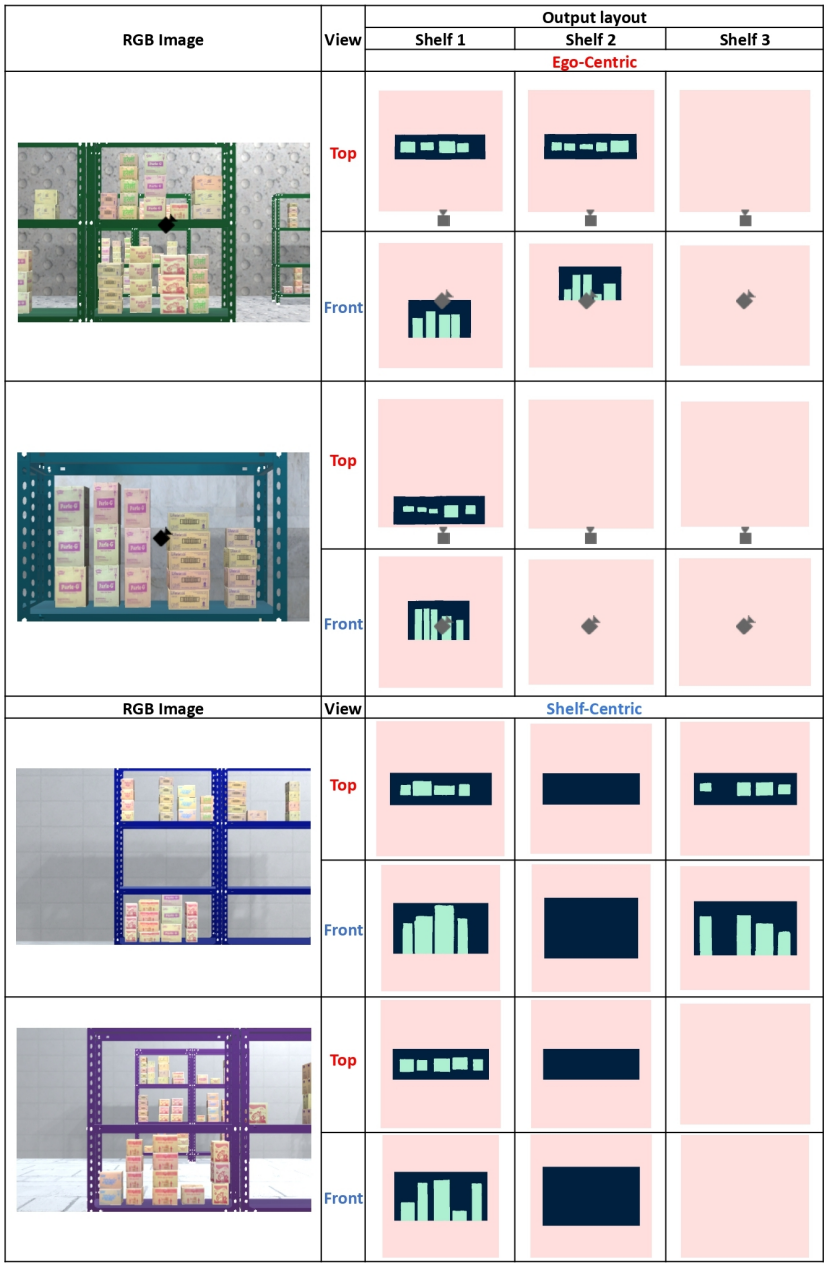

Figure 4.3 Shelf-centric and Ego-centric Results: Here, we present the results of our network trained on shelf-centric (**last 2 rows**) and ego-centric (**top 2 rows**) layouts. The camera symbol in the ego-centric results represents the center of the camera in the RGB image and the positon of the camera in the layouts. For the ego-centric layouts, observe how as the camera moves farther away (one way to notice this is that the size of the rack becomes smaller), the top-view prediction moves further up the layouts (**compare rows 2 and 1**). Also, observe how the front-view layouts for shelves at different heights in the scene are at different heights in the layouts.

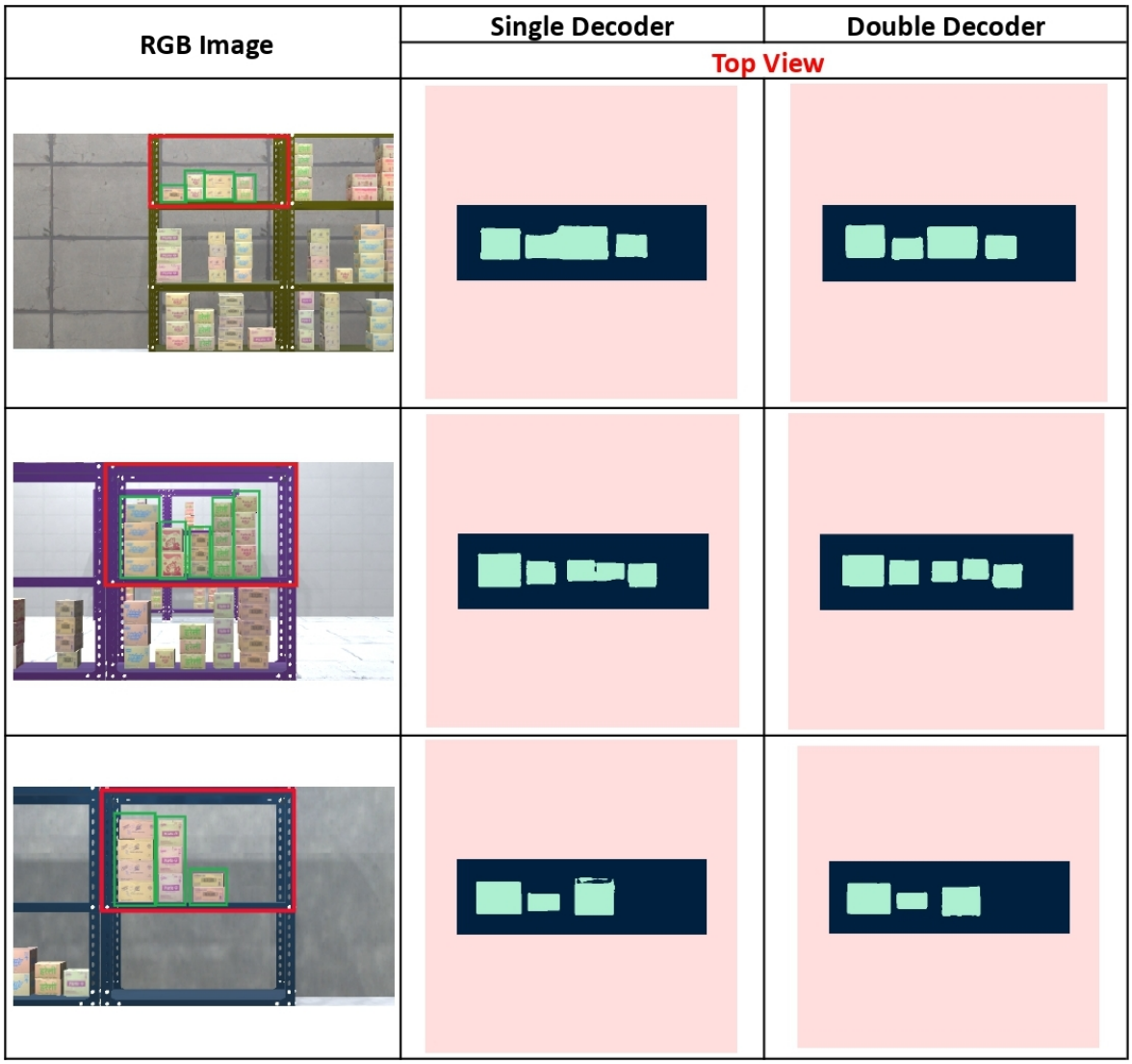

Figure 4.4 Effect of dual-task learning on top-view layout estimation: The output layouts are being displayed only for the shelf bounded with a red box. The corresponding boxes for the bounded shelf are bounded with green boxes. Observe how the double decoder model (**column 3**) results in better prediction of the space between closely placed boxes more accurately (**rows 1 and 2**), reduction of noise around the box predictions (**row 3**) and better prediction the position of boxes (**row 2**).

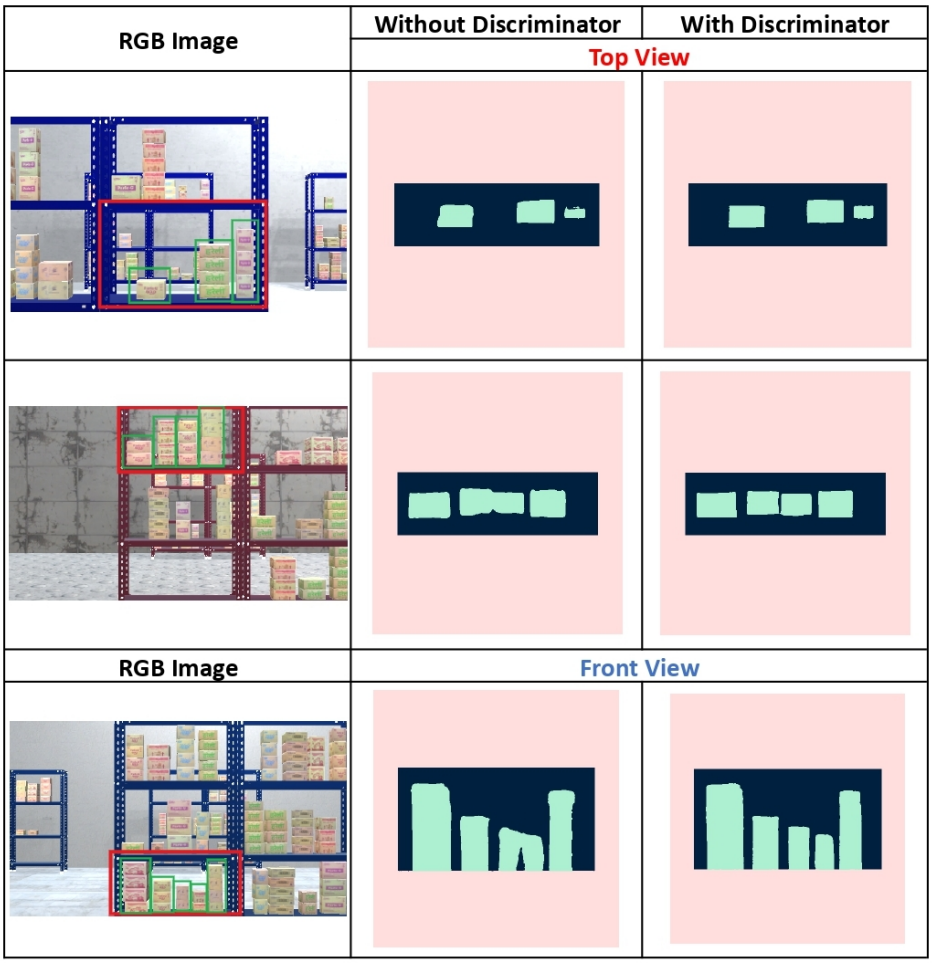

Figure 4.5 Effect of Discriminator on qualitative performance: The output layouts in **columns 2-3** are being displayed only for the shelf bounded with a red box in **column 1**. The corresponding boxes for that shelf are bounded with green boxes. Observe how using a discriminator leads to more box-like layouts (less noise) for boxes and also avoids merging the layouts for 2 boxes (**rows 2, 3**).

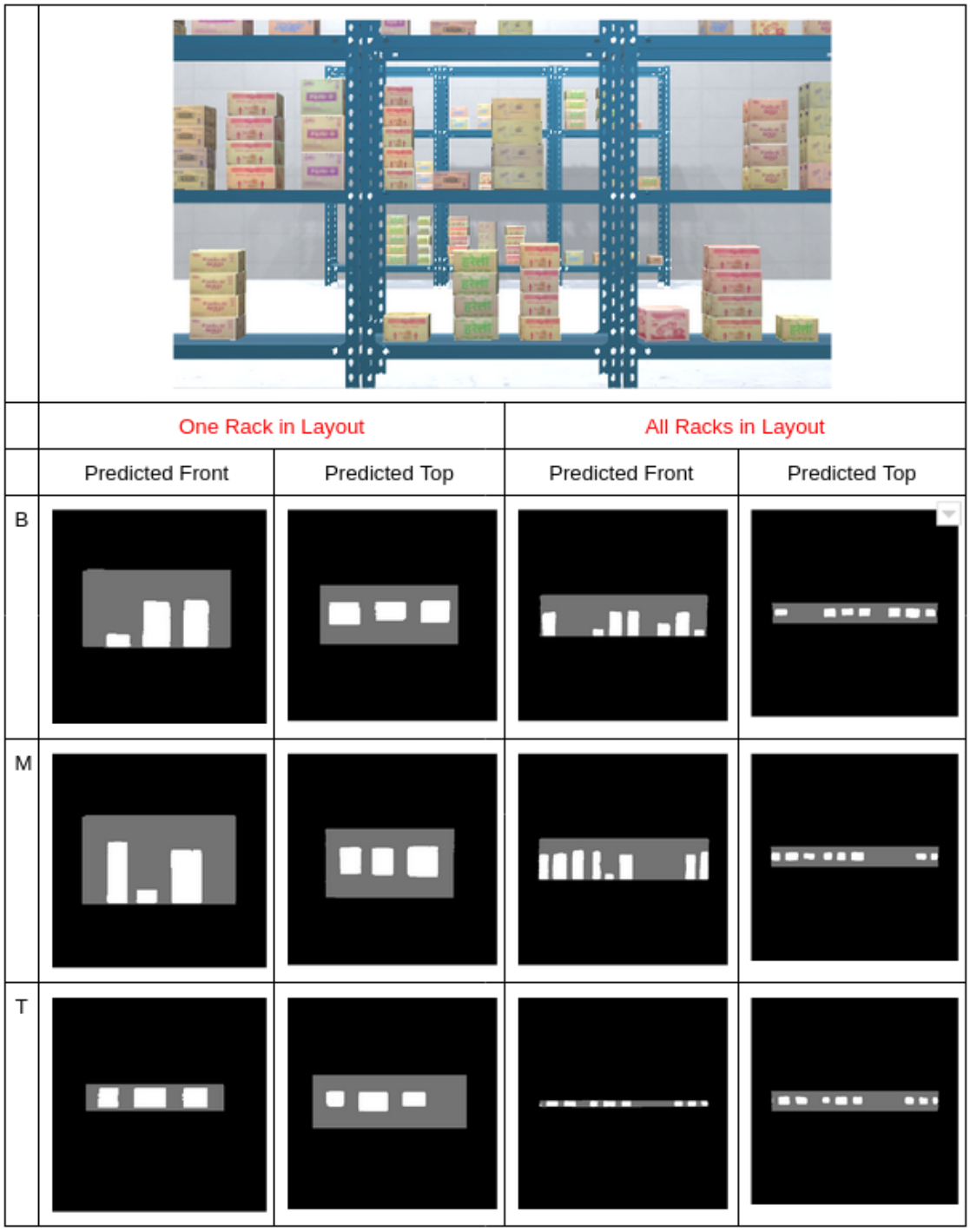

Figure 4.6 Comparison of layouts predicted by Network trained on Layouts with one rack vs all the racks. The layouts predicted by network trained on layouts with one shelf are better than the layouts produced by network with all the racks in layout.

Chapter 5

Dataset Generation Pipeline

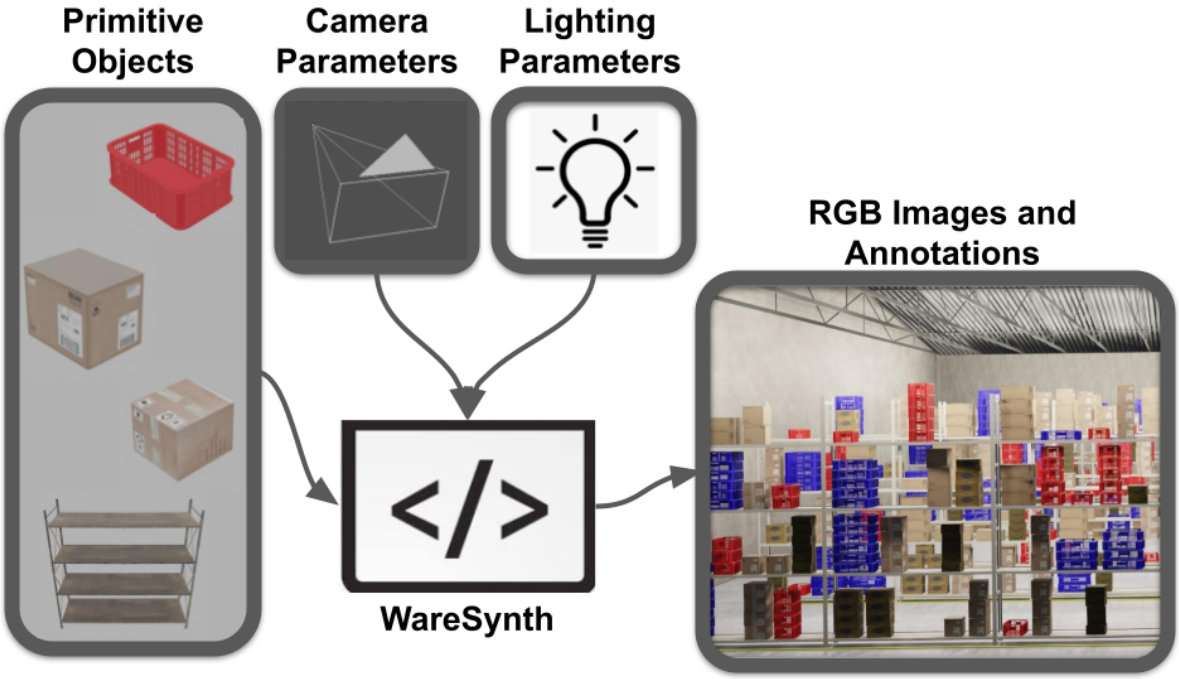

Figure 5.1 An Overview of how WareSynth works.

In this chapter, We introduce the synthetic data generation pipeline termed as WareSynth, which can be used to generate 3D warehouse scenes, automate the process of data capture and generate corresponding annotations.

There are very few datasets publicly available for warehouse settings. Real-world datasets like LOCO [51] exist for scene understanding in warehouse logistics, in which they provide a limited number of RGB images, along with corresponding 2D annotations. Due to the difficulty in acquiring the 3D ground truth information from real scenes there aren't any real warehouse datasets which provide information about the objects in scene and their relative

positions and orientation. For 3D deep learning applications, large amount of diverse data along with 3D ground truth information is required. There are general purpose synthetic data simulators like NVIDIA Isaac [1], which provide warehouse scenes. However, they provide lesser control as to specifying properties for the warehouse, and can't be modified easily to generate annotations needed for our task. To this end, We introduce our dataset generation pipeline.

5.1 Primitive Objects

We use the Open source 3D graphics toolset Blender(version 2.91) for modelling the Primitive Objects which is used to create the bigger warehouse, In the beginning we used freely available 3D models from platforms like Sketchup and Free3d, but since the structure of these models were not regular hence we extracted the textures from them and created regular structure using the Blender's modelling tool. These objects include: boxes, crates, racks, warehouse structures, forklifts, fire extinguishers etc. These Objects are freely available to use for anyone at Link[1].

5.2 Generation process

Our generation process entails placement of objects in the scene procedurally in a randomized fashion, followed by adjustment of the lighting and textures. We perform texture editing and manipulate roughness and reflectance properties of objects to mimic real warehouse scenes. We start with an empty warehouse. Racks are placed inside the warehouse according to a randomly generated 2D occupancy map. Lighting in the warehouse is also according to the same map, where we illuminate the corridors and also introduce ambient lighting. We keep the inter-shelf height and number of shelves in racks, width of corridors and overall rack density of the warehouse as parameters which can be tuned as per requirements.

It is important to note that WareSynth is not constrained by our specific settings. The existing models can be readily substituted with custom box and rack models to configure the warehouse. We also randomize the placement of boxes on each particular rack by specifying parameters which control the density of boxes placed and minimum distance between the boxes. We vary the density of rack occupancy by sampling the associated parameter from a uniform distribution between 0 (empty shelf) and 1 (fully occupied shelf). This ensures that the data is not imbalanced. Our algorithm picks a random box from available box models, and positions the same at a random angle varying between $\pm r^{\circ}$ (where r can be specified). The boxes can also be stacked over each other, on individual shelves. This probabilistic procedure

[1]Link to primitive objects: https://tinyurl.com/bdhxtd2t

ensures that boxes are placed on the shelves randomly, but within the specified constraints, which helps us generate a large variety of realistic data.

5.3 Domain Adaptation

Since we wanted the network to train majorly in Synthetic Images and be able to predict the layouts from the real world images which is significantly different from the Synthetic Images hence we have used various domain adaptation techniques. As a part of domain adaptation we have used various intensities of ambient light, varied the sizes of Boxes, varied the color of rack.

We also varied the surrounding of the rack space so that the network is able to gauge the rack's placement hence we used several combinations of wall and floor textures. We collected the textures from capturing the textures from around our labs, hostels and home. All these domain adaptation features are inbuilt into the Data-set generation pipeline and we have seen that the network has shown improvement in layout prediction when we introduced the data-sets with domain adaptation.

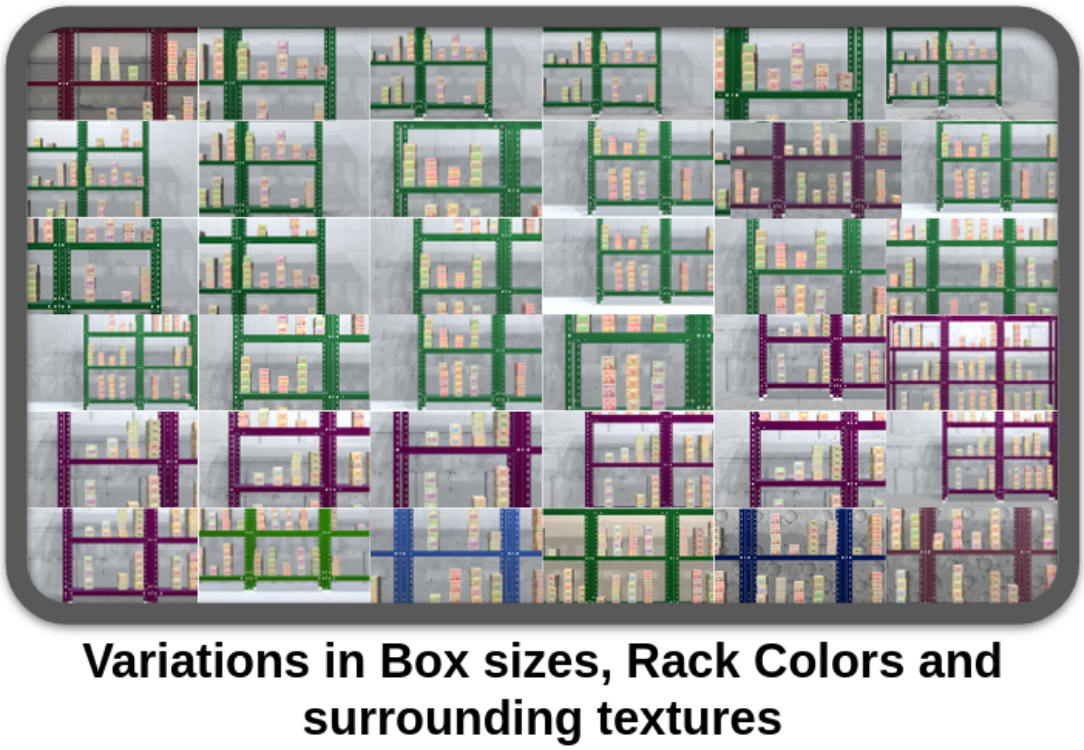

Figure 5.2 Figure showing the various Domain randomization techniques applied.

5.4 Simulated Real World Dataset

We have also created the dataset where all the textures are captured from real world be it the the textures from boxes or rack or walls and floor, This simulated real world data is very useful because textures from the real world make the synthetic very real compared the textures we used from the other online sources. In order to create similar simulated real world dataset for a warehouse there are 4 major subsections one needs to handle which are Objects placed the racks, Racks, wall and floor textures and placement of lights.

For recreating the boxes from the real world into simulated world we captured the box from all 6 sides and measured the dimensions of the boxes and created the box with same dimensions in blender where although the actual dimensions of the boxes are altered but the ratios between these sides are maintained. We have also gathered small amount of Real world data which we have used to fine tune the network, In order to gather this real world data we manually measured the sizes boxes and racks, along with this we also measured the relative position of boxes and rack with respect to the camera, using this information we are able to create the layouts for the real world images.

5.5 Camera Movement

We have two types of camera movement in order to capture the images. For Racklay we place the camera in front of the racks facing the racks, and then the position of the camera is moved close/far and up/down so that atleast one shelf is visible and at max all the shelves are visible. For vRacklay we place the camera such that a rack is visible and then the camera is moved rightward with very small steps such that a sequence of images is created.

5.6 Rendering on Unity vs Blender

We have used to tested simulators for generating and capturing the RGB images along with corresponding annotations. In the beginning we used blender to models the objects so we used blender to render the images. But later we also tested Unity as a simulator where the models created in Blender were imported and placed in the warehouse and then RGB images were captured.

Since we have tried two simulators we would like to point out the some very interesting observations. One of the most important observations is the rendering time between Unity and Blender, We used a simple Core i7 5th generation chip laptop. It took 2 Minutes to render one Image in Blender whereas in the same time Unity can render 30 Images. Unity is a better for rendering real looking images because its designed for it.

5.7 Camera Output

We capture data via movement of a 6-DoF camera around the warehouse corridors by specifying a path or a set of discrete positions. The camera parameters can be varied in order to produce a diversity of views. The camera rotation, focal length, height above the ground etc. can all be manipulated and constrained according to the kind of views desired. As per the requirement, we can capture the RGB images at the desired resolution for each of these camera positions, along with the camera intrinsic, and extrinsic parameters. Although we only capture these parameters other information such as the segmentation mask, the depth maps, the 3D bounding boxes.

We can also extract 2D annotations such as 2D bounding boxes and semantic and instance segmentation masks of the objects. We can also obtain the 3D positions, orientations and 3D bounding boxes for all objects present in the camera FOV, along with depth maps and normal information. Our pipeline can also be used to obtain stereo-information. The obtained data can be exported to various popular annotation formats such as KITTI, COCO, Pix3D, BOP etc.

5.8 Applications and extensions

WareSynth can be used for various tasks such as 2D/3D object detection, semantic/instance segmentation, layout estimation, 3D scene navigation and mapping, 3D reconstruction etc. The same pipeline can also be modified to other kind of scenes such as supermarkets, greenhouses by changing the database of objects and placement parameters. The generation procedure and data capture methods are efficient, very flexible and can be customized as per user requirement. This makes the pipeline very useful for future research and generating annotated data at a large scale.

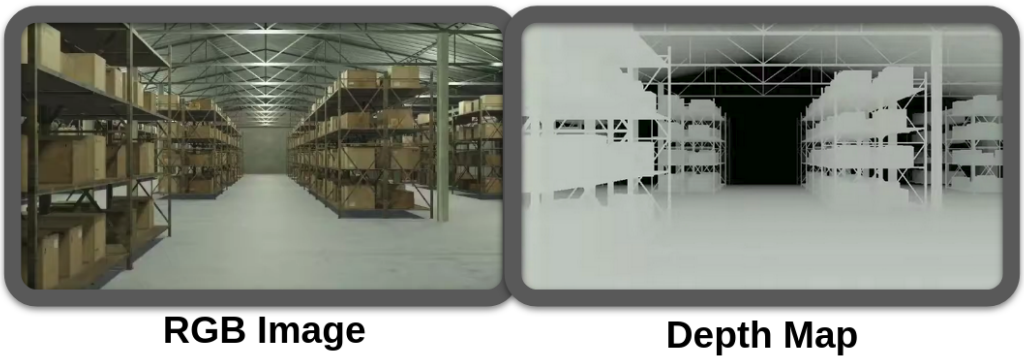

Figure 5.3 RGB Image vs Depth Map extracted from the WareSynth Pipeline.

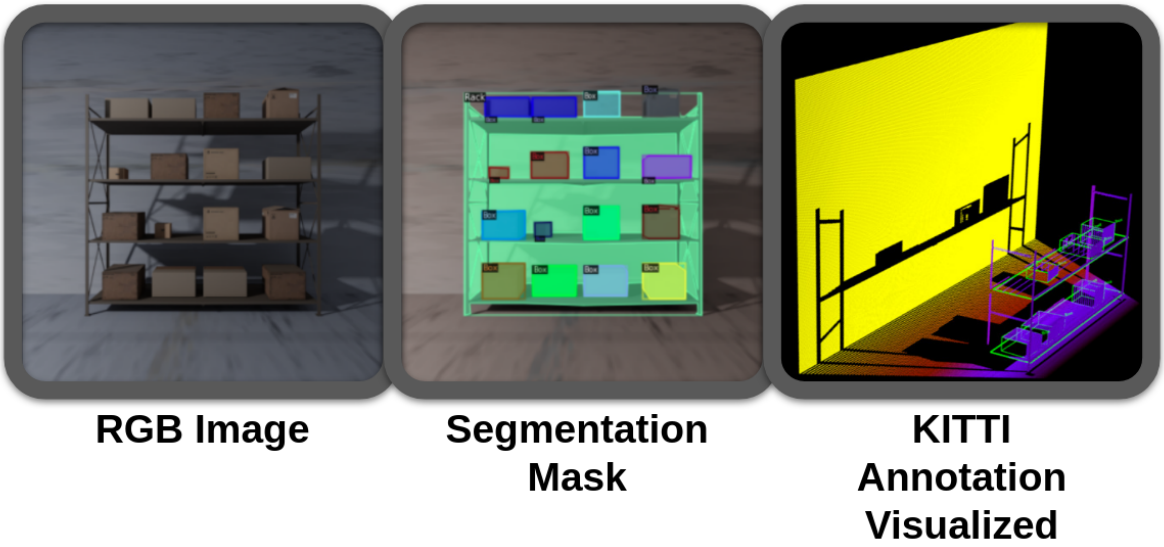

Figure 5.4 The RGB Image, Segmentation Mask and the Kitti Annotations Visualized.

Chapter 6

Applications

We had started with a goal in chapter 1 that we wanted to gather 3D understanding of the scene around the robot in a warehouse setting. Then using the Racklay we are able to get the Top-view layout and Front-view layout. These layouts can be used for multiple applications. In these chapter we will discuss some of the most important applications like how do we obtain the 3D Representation of the warehouse, how can we get the SDF representation of the warehouse both these representations are crucial to robot path planning.

6.1 Counting Number of Boxes

Using the Layouts obtained from Racklay we can apply thresholdling to extract the boxes out of layouts, These boxes are disjoint and they can be counted by counting the number of disjoint components. And these disjoint components give us the count of boxes kept on a particular shelf.

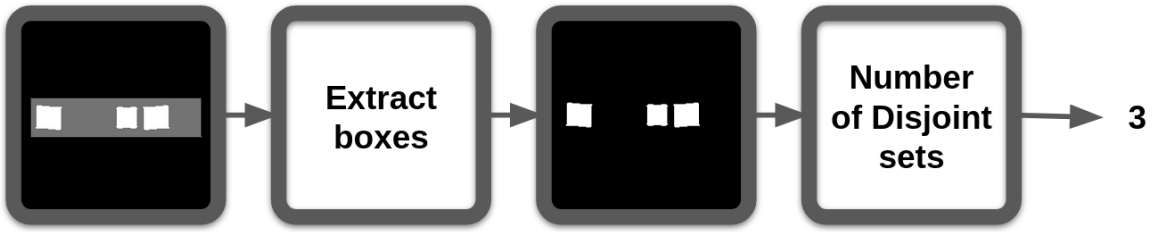

Figure 6.1 Diagrammatic explanation of how to get the number of boxes from the layout. step 1) using thresholding get the boxes out from the layouts. step 2) Using Digital Image processing the get the count of disjoint boxes. hence obtaining the number of boxes.

6.2 3D Free Space Estimation

3D Free space estimations has been one of main feature requested by warehouse management systems. We solve this problem of estimating the free space in a rack space with our proposed network. Using a monocular RGB image which can be captured by a robot Racklay predicts very accurate Top-View and Front-View layouts.

Using these Top-View and Front-View layouts we threshold to get the rack and boxes in a binary image. Then using these binary images we can obtain the 2D Bounding boxes for the boxes and rack in top-view and front-view.

Now we can assume a coordinate system where the direction along the rack can be assumed X, direction towards the height in front view can be assumed to be Y and the direction along the depth of the rack can be assumed to be Z axis. (also given in 6.2 for understanding).

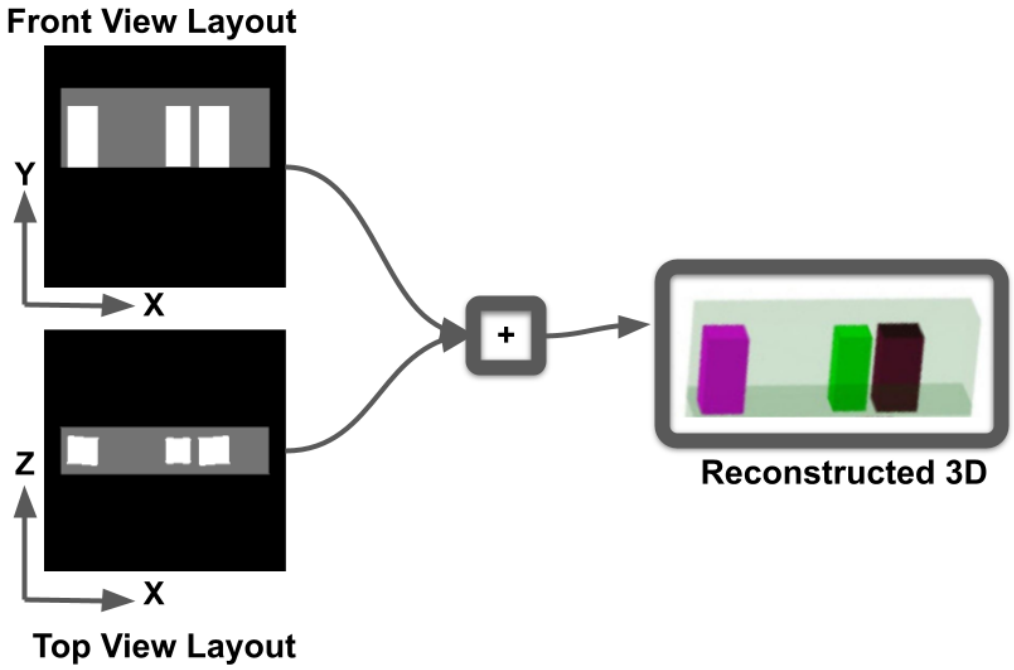

Figure 6.2

Following that we have all the information(X, Y, Z) needed to make a 3D representation of the rack provided in the RGB Image. From this representation we know that free space in the rack is all the space inside the racks excluding the space occupied for boxes. Since the we know the dimensions of the rack and boxes we can get this free by the following formula FRS = $\sum_i^n (X_{rack}i * Y_{rack}i * Z_{rack}i) - \sum_i^n (X_{box}i * Y_{box}i * Z_{box}i)$.

This value accounts for every little space present in the rack-space. with in not really helpful. Hence we would like to calculate the free space where a box of reasonable size can be placed. Since we know the position and dimension of all the boxes and rack and given a minimum box size we can identify where-all we can place the boxes. this includes the spaces in the rack and also the space on top of the existing boxes.

6.3 SDF Representation:

SDF Stands for "Signed Distance Function". This function tells the minimum distance from the surface/boundary of an object or environment. The value of this function is positive if the point is outside the object and its negative for the points inside the object. This type of representation is very useful for a robot to know how far this the closest point on object with respect to the robotic arm.

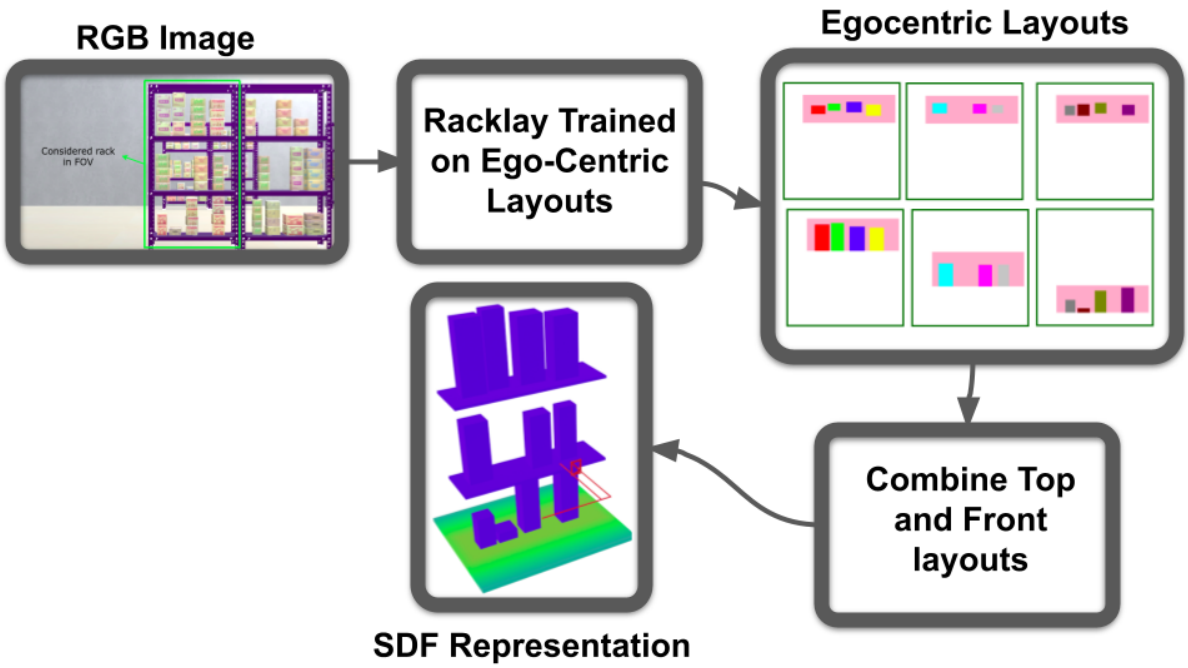

Figure 6.3 Diagram explaining how Racklay can be used to get SDF of the racks using the ego-centric layouts.

If we assume that the camera is mounted on the robotic arm and it is moving and we create the 3D reconstruction of the racks using the ego-centric layouts, we can get the camera/Robotic arm trajectory using these 3D representation described below.

We first obtain the dimensions and coordinates of the boxes and shelves from the top-view and front-view ego-centric layouts, from which we create a Signed Distance Function

(SDF) representation for the scene (Fig. 6.3). Then from this 3D representation we sample-out points to get a point cloud representation. Then we repeat this reconstruction and sampling step for all the points where we want to estimate the trajectory of camera, for each point cloud we also know the previous point cloud and we can apply ICP (Iterative Closest Point) algorithm that tries to align the two point clouds and outputs the Rotation and Translations between those two point clouds and since these point clouds are made from the ego-centric layouts we can get Rotation and Translation of camera between those to frames.

We are able to precisely estimate pose of a camera moving in the vicinity of the rackspace using the layout representations, that results in **accurate** computation and updating of SDF from various viewpoints and **noiseless** representation of the same with respect to a global frame.

Chapter 7

Conclusion and Future Works

7.1 Conclusion

Warehouse Management Systems are in the future will be operated by robots and for the robots to operate the they need to estimate the 3D Map of warehouse. There are methods to estimate the 3D map of the warehouse from monocular images like MeshRCNN and depth guided methods like pseudo-lidar but since the warehouse racks are thin and they get occluded by the boxes kept on them and hence these methods do not work on the rack structures. We tackled this problem by estimating the layout of the images from top-view and front-view, and then by extracting box combining these both layouts we estimate the accurate 3D Representation of the environment.

We propose ***RackLay***, which to the best of our knowledge is the first approach that provides multi-layered layout estimation for racks from a single image. We also extend the network to predict pose aware (*ego-centric*) layouts, using which we can obtain a 3D reconstruction of the rack in camera frame using a single image.

We also introduce a versatile synthetic data generation pipeline, ***WareSynth***, that is capable of producing domain randomized data, which enables sim2real transfer. *RackLay*'s versatility is showcased across a large diversity of both synthetic and real images and is vastly superior to prior art adapted for the same task.

7.2 Future Work

So this work has established the base for the robot to map out a 3D environment visible in the camera the further works can be on how do we register the warehouse as the robot move which means as of now the robots is able to map only what is visible in camera's perception but after the registration is done the robot will be able to not only tell about what is visible in camera it will also be able to map the previous scenes from where the camera has seen.

Then the robot should be able to plan a path such that it is able to map the whole warehouse. Further more this 3D representation can be integrated with the path planning of the robotic arm such that robot is able to move a box from on rack to another another or even move the box in the same shelf such that more space can be made in the shelf to accommodate more boxes.

Related Publications

Monocular Multi-Layer Layout Estimation for Warehouse Racks: **Anurag Sahu**, Meher Shashwat Nigam, Avinash Prabhu, Puru Gupta, Tanvi Karandikar, N. Sai Shankar, Ravi Kiran Sarvadevabhatla, K. Madhava Krishna - In Proceedings of 12th Indian Conference on Computer Vision, Graphics and Image Processing (ICVGIP'21).

Bibliography

[1] NVIDIA Isaac SDK, 2019.

[2] Trends in warehouse automation-2021, Oct 2020.

[3] Rapyuta robotics, 2021. Accessed: 13-09-2021.

[4] Unity real-time development platform | 3d, 2d vr ar engine, 2021. Accessed: 13-09-2021.

[5] M. Bansal, A. Krizhevsky, and A. Ogale. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. *arXiv preprint*, 2018.

[6] J. Baxter. A model of inductive bias learning. *CoRR*, abs/1106.0245, 2011.

[7] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall. Semantickitti: A dataset for semantic scene understanding of lidar sequences. In *ICCV*, 2019.

[8] J. Beltrán, C. Guindel, F. M. Moreno, D. Cruzado, F. Garcia, and A. De La Escalera. Birdnet: a 3d object detection framework from lidar information. In *ITSC*, 2018.

[9] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Blender Institute, Amsterdam,

[10] E. Bochinski, V. Eiselein, and T. Sikora. High-speed tracking-by-detection without using image information. In *International Workshop on Traffic and Street Surveillance for Safety and Security at IEEE AVSS 2017*, 2017.

[11] P. Buxbaum. Many warehouses don't have wms, Aug 2018.

[12] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom. nuscenes: A multimodal dataset for autonomous driving. *arXiv preprint arXiv:1903.11027*, 2019.

[13] R. Caruana. Multitask learning. *Machine Learning*, 28:41–75, 2004.

[14] J.-R. Chang and Y.-S. Chen. Pyramid stereo matching network. In *CVPR*, 2018.

[15] M.-F. Chang, J. Lambert, P. Sangkloy, J. Singh, S. Bak, A. Hartnett, D. Wang, P. Carr, S. Lucey, D. Ramanan, et al. Argoverse: 3d tracking and forecasting with rich maps. In *CVPR*, 2019.

[16] X. Chen, K. Kundu, Z. Zhang, H. Ma, S. Fidler, and R. Urtasun. Monocular 3d object detection for autonomous driving. In *CVPR*, 2016.

[17] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia. Multi-view 3d object detection network for autonomous driving. In *CVPR*, 2017.

[18] D. Cireşan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. *arXiv preprint*, 2012.

[19] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. In *CVPR*, 2016.

[20] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

[21] N. Garnett, R. Cohen, T. Pe'er, R. Lahav, and D. Levi. 3d-lanenet: end-to-end 3d multiple lane detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2921–2930, 2019.

[22] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *CVPR*, 2012.

[23] R. Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[24] C. Godard, O. Mac Aodha, M. Firman, and G. Brostow. Digging into self-supervised monocular depth estimation. *arXiv preprint*, 2018.

[25] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27*. 2014.

[26] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik. Cognitive mapping and planning for visual navigation. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7272–7281, 2017.

[27] D. Hall, F. Dayoub, J. Skinner, H. Zhang, D. Miller, P. Corke, G. Carneiro, A. Angelova, and N. Sünderhauf. Probabilistic object detection: Definition and evaluation. *arXiv preprint arXiv:1811.10800*, 2018.

[28] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Maskrcnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[29] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015.

[30] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[31] X. Huang, P. Wang, X. Cheng, D. Zhou, Q. Geng, and R. Yang. The apolloscape open dataset for autonomous driving and its application. *arXiv preprint arXiv:1803.06184*, 2018.

[32] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint*, 2015.

[33] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *CVPR*, 2017.

[34] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *CVPR*, 2017.

[35] A. Kar, S. Tulsiani, J. Carreira, and J. Malik. Amodal completion and size constancy in natural scenes. In *International Conference on Computer Vision (ICCV)*, 2015.

[36] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representatiosn (ICLR)*, 2015.

[37] J. Ku, M. Mozifian, J. Lee, A. Harakeh, and S. L. Waslander. Joint 3d proposal generation and object detection from view aggregation. In *IROS*, 2018.

[38] C. Lee, V. Badrinarayanan, T. Malisiewicz, and A. Rabinovich. Roomnet: End-to-end room layout estimation. *CoRR*, abs/1703.06241, 2017.

[39] B. Li, W. Ouyang, L. Sheng, X. Zeng, and X. Wang. Gs3d: An efficient 3d object detection framework for autonomous driving. In *CVPR*, 2019.

[40] M. Liang, B. Yang, S. Wang, and R. Urtasun. Deep continuous fusion for multi-sensor 3d object detection. In *ECCV*, 2018.

[41] H.-J. Lin, S.-W. Huang, S. Lai, and C.-K. Chiang. Indoor scene layout estimation from a single image. *2018 24th International Conference on Pattern Recognition (ICPR)*, pages 842–847, 2018.

[42] S. Lin, R. Clark, R. Birke, N. Trigoni, and S. Roberts. Wise-ale: Wide sample estimator for aggregate latent embedding. 2019.

[43] C. Liu, A. Schwing, K. Kundu, R. Urtasun, and S. Fidler. Rent3d: Floor-plan priors for monocular layout estimation. In *CVPR*, 2015.

[44] M. Liu, X. He, and M. Salzmann. Building scene models by completing and hallucinating depth and semantics. In *European Conference on Computer Vision*, pages 258–274. Springer, 2016.

[45] C. Lu and G. Dubbelman. Hallucinating beyond observation: Learning to complete with partial observation and unpaired prior knowledge. *arXiv preprint*, 2019.

[46] C. Lu and G. Dubbelman. Semantic foreground inpainting from weak supervision. *arXiv preprint arXiv:1909.04564*, 2019.

[47] C. Lu and G. Dubbelman. Semantic foreground inpainting from weak supervision. *IEEE Robotics and Automation Letters*, 2020.

[48] C. Lu and vande Molengraft. Monocular semantic occupancy grid mapping with convolutional variational encoder-decoder networks. *IEEE Robotics and Automation Letters*, 2019.

[49] W. Maddern, G. Pascoe, C. Linegar, and P. Newman. 1 Year, 1000km: The Oxford RobotCar Dataset. *The International Journal of Robotics Research (IJRR)*, 36(1):3–15, 2017.

[50] K. Mani, S. Daga, S. Garg, S. Shankar, J. Krishna Murthy, and K. Madhava Krishna. Monolayout: Amodal layout estimation from a single image. *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2020.

[51] C. Mayershofer, D. M. Holm, B. Molter, and J. Fottner. Loco: Logistics objects in context. In *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 612–617, 2020.

[52] A. Mousavian, D. Anguelov, J. Flynn, and J. Kosecka. 3d bounding box estimation using deep learning and geometry. In *CVPR*, 2017.

[53] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[54] OpenStreetMap contributors. Planet dump retrieved from https://planet.osm.org . https://www.openstreetmap.org, 2017.

[55] O. Özyeşil, V. Voroninski, R. Basri, and A. Singer. A survey of structure from motion*. *Acta Numerica*, 26:305–364, 2017.

[56] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello. Enet: A deep neural network architecture for real-time semantic segmentation. *arXiv preprint arXiv:1606.02147*, 2016.

[57] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2536–2544, 2016.

[58] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[59] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[60] X. Ren, J. Luo, E. Solowjow, J. A. Ojea, A. Gupta, A. Tamar, and P. Abbeel. Domain randomization for active pose estimation. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 7228–7234. IEEE, 2019.

[61] T. Roddick, A. Kendall, and R. Cipolla. Orthographic feature transform for monocular 3d object detection. *arXiv preprint*, 2018.

[62] O. Ronneberger and Fischer. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, 2015.

[63] S. Rota Bulò, L. Porzi, and P. Kontschieder. In-place activated batchnorm for memory-optimized training of dnns. In *CVPR*, 2018.

[64] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. In *Advances in neural information processing systems*, 2016.

[65] A. Saxena, M. Sun, and A. Y. Ng. Make3d: Learning 3d scene structure from a single still image. *IEEE transactions on pattern analysis and machine intelligence*, 31(5):824–840, 2008.

[66] S. Schulter, M. Zhai, N. Jacobs, and M. Chandraker. Learning to look around objects for top-view representations of outdoor scenes. In *ECCV*, 2018.

[67] S. Shi, C. Guo, L. Jiang, Z. Wang, J. Shi, X. Wang, and H. Li. Pv-rcnn: Point-voxel feature set abstraction for 3d object detection. *arXiv preprint arXiv:1912.13192*, 2019.

[68] S. Shi, X. Wang, and H. Li. Pointrcnn: 3d object proposal generation and detection from point cloud. In *CVPR*, 2019.

[69] S. Srikanth, J. A. Ansari, S. Sharma, et al. Infer: Intermediate representations for future prediction. *arXiv preprint*, 2019.

[70] S. Srivastava, F. Jurie, and G. Sharma. Learning 2d to 3d lifting for object detection in 3d for autonomous vehicles. *arXiv preprint*, 2019.

[71] T. Standley, A. R. Zamir, D. Chen, L. J. Guibas, J. Malik, and S. Savarese. Which tasks should be learned together in multi-task learning? *CoRR*, abs/1905.07553, 2019.

[72] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine, V. Vasudevan, W. Han, J. Ngiam, H. Zhao, A. Timofeev, S. Ettinger, M. Krivokon, A. Gao, A. Joshi, Y. Zhang, J. Shlens, Z. Chen, and D. Anguelov. Scalability in perception for autonomous driving: Waymo open dataset, 2019.

[73] S. Thrun. Is learning the n-th thing any easier than learning the first? In *Proceedings of the 8th International Conference on Neural Information Processing Systems*, NIPS'95, page 640–646, Cambridge, MA, USA, 1995. MIT Press.

[74] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.

[75] A. Torralba, A. A. Efros, et al. Unbiased look at dataset bias. In *CVPR*, 2011.

[76] J. Tremblay, A. Prakash, D. Acuna, M. Brophy, V. Jampani, C. Anil, T. To, E. Cameracci, S. Boochoon, and S. Birchfield. Training deep networks with synthetic data: Bridging the reality gap by domain randomization. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 969–977, 2018.

[77] S. Tulsiani, S. Gupta, D. Fouhey, A. A. Efros, and J. Malik. Factoring shape, pose, and layout from the 2d image of a 3d scene. In *Computer Vision and Pattern Regognition (CVPR)*, 2018.

[78] S. Tulsiani, S. Gupta, D. F. Fouhey, A. A. Efros, and J. Malik. Factoring shape, pose, and layout from the 2d image of a 3d scene. In *CVPR*, 2018.

[79] R. Uittenbogaard, C. Sebastian, J. Vijverberg, B. Boom, D. M. Gavrila, et al. Privacy protection in street-view panoramas using depth and multi-view imagery. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10581–10590, 2019.

[80] Y. Wang, W.-L. Chao, D. Garg, B. Hariharan, M. Campbell, and K. Q. Weinberger. Pseudo-lidar from visual depth estimation: Bridging the gap in 3d object detection for autonomous driving. In *CVPR*, 2019.

[81] Z. Wang, B. Liu, S. Schulter, and M. Chandraker. A parametric top-view representation of complex road scenes. In *CVPR*, 2019.

[82] X. Weng and K. Kitani. Monocular 3d object detection with pseudo-lidar point cloud. *arXiv preprint*, 2019.

[83] J. Wu, C. Zhang, T. Xue, B. Freeman, and J. Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Advances in neural information processing systems*, pages 82–90, 2016.

[84] B. Xu and Z. Chen. Multi-level fusion based 3d object detection from monocular images. In *CVPR*, 2018.

[85] C. Yan, B. Shao, H. Zhao, R. Ning, Y. Zhang, and F. Xu. 3d room layout estimation from a single rgb image. *IEEE Transactions on Multimedia*, 22(11):3014–3024, 2020.

[86] B. Yang, W. Luo, and R. Urtasun. Pixor: Real-time 3d object detection from point clouds. In *CVPR*, 2018.

[87] Y. You, Y. Wang, W.-L. Chao, D. Garg, G. Pleiss, B. Hariharan, M. Campbell, and K. Q. Weinberger. Pseudo-lidar++: Accurate depth for 3d object detection in autonomous driving. *arXiv preprint*, 2019.

[88] Q. Yu, Y. Yang, F. Liu, Y.-Z. Song, T. Xiang, and T. M. Hospedales. Sketch-a-net: A deep neural network that beats humans. *International journal of computer vision*, 2017.

[89] C. Zou, A. Colburn, Q. Shan, and D. Hoiem. Layoutnet: Reconstructing the 3d room layout from a single rgb image. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2051–2059, 2018.