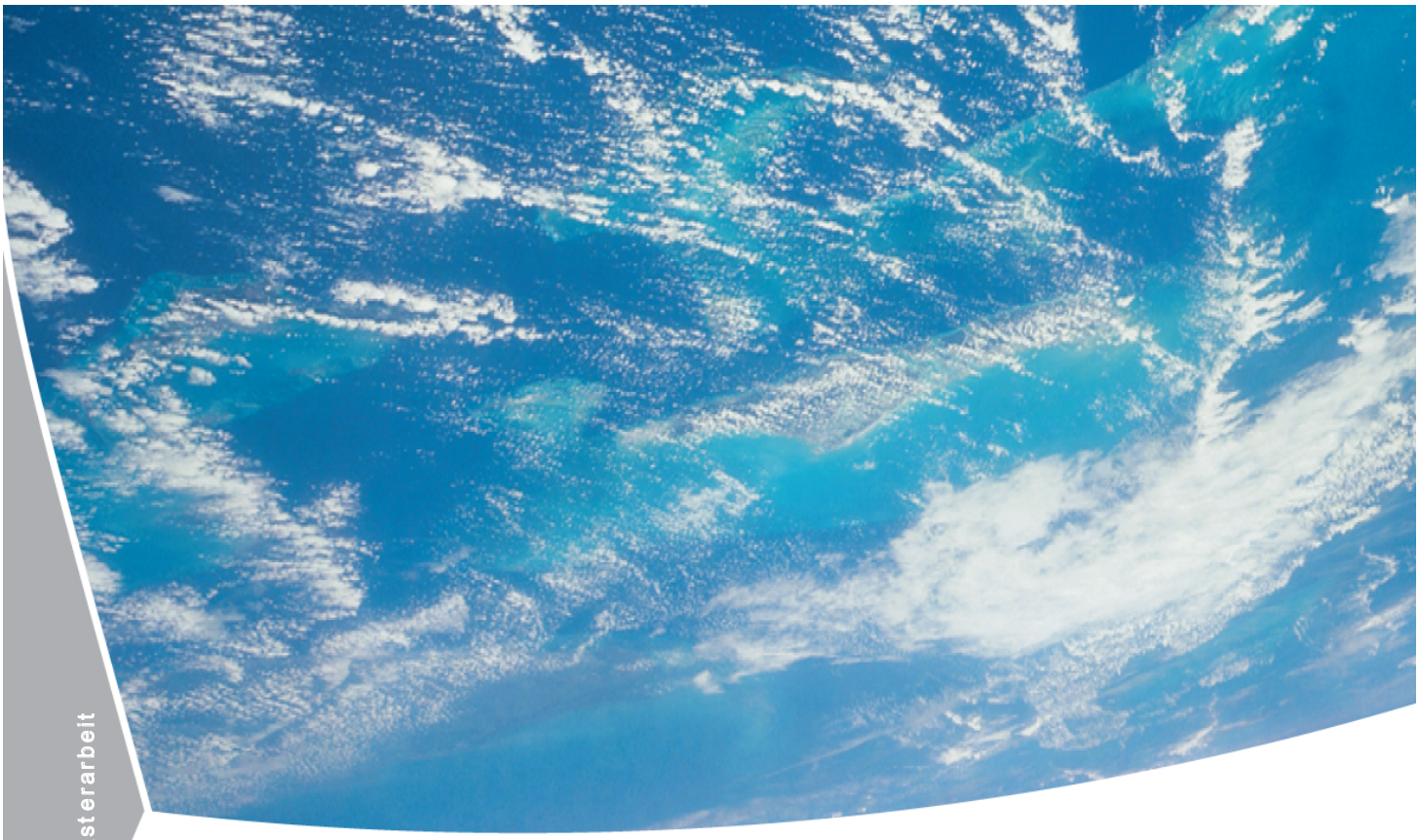


Masterarbeit

Learning from Simulation to Improve Robotic Sensing

Amrutha Saseendran

DLR

Institut für Robotik und Mechatronik

BJ.: 2019

IB.Nr.: DLR-IB-RM-OP-2019-56

MASTERARBEIT

LEARNING FROM SIMULATION TO IMPROVE ROBOTIC SENSING

Freigabe:	Der Bearbeiter:	Unterschriften
	Amrutha Saseendran	
	Betreuer:	
	Zoltán-Csaba Márton	
	Der Institutsdirektor	
	Prof. Alin Albu-Schäffer	

Dieser Bericht enthält 95 Seiten, 50 Abbildungen und 3 Tabellen

Ort: Oberpfaffenhofen	Datum: 28/03/2019	Bearbeiter: Amutha Saseendran	Zeichen: MS

Institut für Theoretische Elektrotechnik und
Mikroelektronik
Universität Bremen

Master Thesis

Learning from Simulation to Improve Robotic Sensing

Amrutha Saseendran
Control Microsystems and Microelectronics
Universität Bremen
Matriculation Number: 3062300

Supervisors:

Dr. Zoltán-Csaba Márton

Dipl.-Inf. Manuel Brucker

M.Sc Martin Sundermeyer
Institute for Robotics and Mechatronics, German Aerospace Centre (DLR)

Prof. Dr.-Ing. Alberto Garcia-Ortiz
Chair for Integrated Digital Systems, ITEM

Prof. Michael Beetz PhD
Head of the Institute for Artificial Intelligence, University of Bremen

March 2019

Acknowledgement

This thesis work was carried out in the year 2018/19 at the German Aerospace Center (DLR), at the Institute of Robotics and Mechatronics.

Foremost, I would like to thank my advisors at the DLR, Dr. Zoltán-Csaba Márton, Manuel Brucker and Martin Sundermeyer for their supervision, motivation, support and granted freedom. Their guidance helped me throughout my research and writing of this thesis. Furthermore, I would like to express my sincere gratitude to my university supervisors Prof. Dr.-Ing. Alberto Garcia-Ortiz and Prof. Michael Beetz PhD, for the supervision, advice and smooth execution of my thesis.

I am grateful to Prof. Dr.-Ing. Alin Albu-Schäffer, Head of the Institute of Robotics and Mechatronics at the DLR, for the opportunity to carry out this work in the department of Perception and Cognition.

I would also like to express my personal thanks for a lot of useful discussions to Ferenc Bálint-Benczédi.

Abstract

The ability of the robot to sense its environment is essential for its autonomous operation. Precise object detection and pose estimation, derived from the sensing capabilities, is crucial for the autonomy of any robotic system. Deep learning and neural networks have already revolutionized this field. The research community has successfully developed powerful deep learning algorithms using neural networks for faster and accurate object detection and pose estimation. However, these methods require a large amount of annotated dataset for the training of neural networks. Preparation of such high quality annotated dataset is expensive and time-consuming. An alternative method is to make use of the easily available simulation data instead of real data for training. Because of the domain gap between the simulation and the real data, networks trained with simulation data fails to perform well on real images.

This thesis explores the possibility of generating realistic looking images using Generative adversarial network (GAN), in order to overcome the existing domain gap between the simulated and real images. Recent research on GANs has shown that these networks are capable of producing photo-realistic images and is a promising new area of the research field for producing realistic synthetic images. Inspired from one of the variants of GAN known as CycleGAN, the proposed work explores the possibility of generating realistic images from simulated images using modified CycleGAN architecture. Using the T-LESS and YCB dataset as the benchmark, the generated images are then evaluated by training an object detector network. The quality of the generated images are measured by the accuracy of the detector and then compared with the real domain images.

Contents

List of Figures

List of Tables

List of Abbreviations

AI	Artificial Intelligence
AIMM	Autonomous Industrial Mobile Manipulator
BN	Batch Normalization
CAD	Computer Aided Design
C-GAN	Conditional GAN
CNN	Convolutional Neural Network
CoGAN	Coupled GAN
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DC-GAN	Deep Convolutional GAN
DLR	Deutsches Zentrum für Luft-und Raumfahrt
FPN	Feature Pyramid Network
GAN	Generative Adversarial Network
GPU	Graphical Processing Unit
HOG	Histogram of Oriented Gradients
IN	Instance Normalization
MAE	Mean Absolute Error
mAP	Mean Average Precision
MSE	Mean Squared Error
PixelRNN	Pixel Recurrent Neural Network
R-CNN	Region based CNN
R-FCN	Region based Fully Connected Network
RoI	Region of Interest
RPN	Region Proposal Network
SSD	Single Shot Detector
SVM	Support Vector Machine
T-LESS	Texture-less
VAE	Variational Autoencoder
YCB	Yale-CMU-Berkeley
YOLO	You Only Look Once

Chapter 1

Introduction

In order to enable robots to interact with their environment, the semantic gap between raw sensory data and meaningful representation of their surroundings needs to be overcome. For example, the humanoid robot DLR Justin [1] needs to interact with household objects and DLR AIMM [2] (Autonomous Industrial Mobile Manipulator) executes fetch and carry operations in unstructured environments. For robotic manipulation, fast and accurate object detection and pose estimation is essential.

Object detection is one of the classical research areas in computer vision and Artificial intelligence (AI), which has improved rapidly in the recent years. It deals with the problem of detecting and localizing objects in an image and predicting the bounding box of the objects of interest in the scene. Many approaches have been proposed for object detection, from traditional computer vision techniques to modern deep learning architectures. Learning-based methods like Convolutional neural networks (CNNs) have revolutionized this field of research [3]. Modern network architectures along with advanced computing technology have successfully enabled object detectors that surpass human performance. These models can accurately predict bounding boxes and class labels of objects in various environments. However, they require large amounts of labeled data during training to achieve the desired performance.

Why is the training data an important factor in deep learning? Deep learning models have a huge amount of parameters to tune and require a large dataset to produce a generalizable model. The efficiency of training a deep learning model depends on the quality of the training data provided. In real-world applications, there is little chance, that the objects of interest are part of a publicly available dataset. Therefore, the dataset for training the network has to be acquired manually which is often not practical and can be expensive. An alternative and promising method to overcome this problem is to use artificial or synthetic data. Generating synthetic data that replicates real-world data would help to transfer the impressive results achieved with deep learning to real-world applications. Synthetic data can be used for many applications like image inpainting [4], reinforcement learning [5], digital image enhancements, and natural language processing [6].

The application of synthetic data as training data is still an open research area. To date,

networks trained with synthetic data fail to match the detection performance of their counterparts trained with natural images. This so-called domain gap between synthetic and real dataset is a major challenge faced when using synthetically trained networks. A Generative model is an effective method of learning any kind of data distribution. These models have recently shown advances towards the goal of generating realistic synthetic data and are further explained in the next section of the chapter.

1.0.1 Generative Models

In general, machine learning models are categorized either as generative or discriminative models. Discriminative models try to classify or distinguish the data distribution whereas generative models try to model the underlying data distribution. In other words, if y is an output variable for a given input sample x then, discriminative models try to model the conditional/posterior probability p($y|x$), whereas generative models try to learn the joint probability function p(x,y). Depending upon the application one of these models is chosen appropriately.

Generative models focus on how the given data is generated. For example consider a dataset of samples $x_1, x_2, x_3, ..., x_n$ sampled from a true data distribution $p(x)$ as shown in Fig 1.1. The blue region in the figure corresponds to the region in image space that contains real images in the training dataset and the black dots indicates the images in the dataset. The generative model implicitly defines another data distribution $p(\hat{x})$ by mapping the points from the unit Gaussian distribution z through the neural network. The neural network is basically a function with parameter θ and this parameter is used to control the generated data distribution $p(\hat{x})$. The distribution $p(\hat{x})$ starts randomly and then during the training process by updating θ, the model tries to minimize the differences between the true and the generated distribution. In short, the objective here is to find the parameter θ that produces no difference between the two distributions.

Generative models do not produce the output by merely reproducing the data distribution but discovers the inherent characteristics in a data distribution and learns the underlying structure.

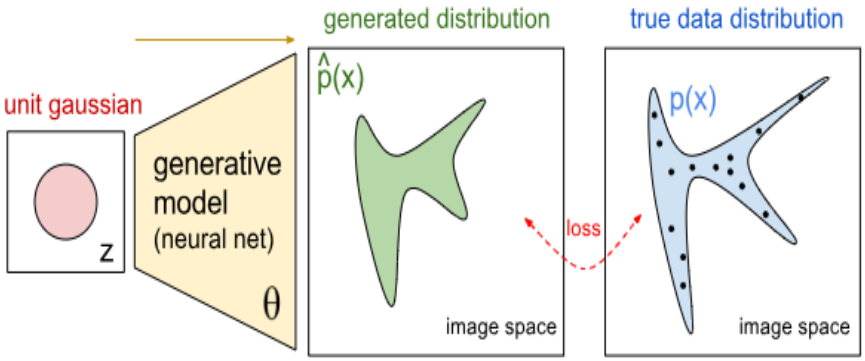


Figure 1.1: Generative model (adapted from [7]).

These models are thus one of the most promising approaches to understand and perceive the visual world around us. Three popular deep generative models are General Adversarial Networks (GANs) [8], Variational Autoencoders [9] (VAEs) and Pixel Recurrent Neural Networks (PixelRNNs) [10]. Generative adversarial networks comprise of two neural networks known as generator and discriminator. The generator generates images from input random noise and the discriminator tries to distinguish the real dataset from the generator's output to classify real and fake data. These two networks are trained simultaneously, where the generator tries to generate images close to the real images making it difficult for the discriminator to distinguish. This is often termed as a two-player mini-max game between generator and discriminator. VAEs use a probabilistic graph model based on Bayesian inference in which the probability distribution of the data is modeled to produce a new sample from the distribution. PixelRNNs are autoregressive neural networks that generate models which predict the conditional distribution of the pixels in an image when previous pixels are given. In this method, the model scans the image, one row and one pixel (within each row) at a time and predicts the distribution over the possible values for the next pixel. PixelRNNs are often used in image completion applications.

Recent research on GANs shows their ability to generate photorealistic images. It is also known to produce sharper images compared to the other generative models like VAE and PixelRNNs. The working principle of GANs is to reduce the distinguishability between the generated and the real data distribution. Because of the game theory approach and the brittle structure of GANs, they are well-known to be tricky to train. A lot of research in GANs is focused on understanding the reasons for this difficulty and the methods to reliably reach convergence.

1.1 Problem Formulation and Research Objective

This thesis focuses on the task of using synthetic images generated from generative adversarial networks for training object detectors. The major problems that we investigate are formulated as follows:

- The existing domain gap between real and synthetic images.

- The difficulty in stabilizing the training process of GANs.

The following aspects are evaluated as a part of this research:

- The state of the art generative adversarial networks for synthetic image generation

- The applicability of GANs to generate synthetic images that resemble real data.

- The possibility of training object detectors with generated synthetic images and no (or as few as possible) real data.

1.2 Methodology

The proposed research is an evaluation of GANs for synthetic image generation and explores their potential using the generated data to train object detectors. However, since the training of GANs can be challenging, suitable training methods are adopted and are further explained in chapter 4. The proposed method will use rendered 3D Computer Aided Design (CAD) models of the objects to be detected and real images as input to the generative model for augmenting the rendered images. The generated synthetic images are then used for training object detectors. The thesis is structured as follows:

• Investigate the potential of GAN for domain adaptation.

• Generate realistic images from CAD rendered images preserving the label information using GAN.

• Evaluate the generated images quality by training an object detector.

• Compare the performance of the object detector using real images, GAN generated images, CAD rendered images and combination of real and GAN images.

1.3 Framework

1.3.1 Tensorflow

Tensorflow is an open source machine learning library developed by the Google brain team and is extensively used nowadays for research and industrial purposes [11]. It was initially used by Google for internal research and production purposes. It was later released under Apache 2.0 license in November 2015. In Tensorflow, computations are done in data flow graphs where a node represents mathematical operations and edges represent multidimensional arrays called tensors. Tensorflow supports Central Processing Unit (CPU), Graphical Processing Unit (GPU) and distributed processing. The architecture of Tensorflow is highly flexible, modular and portable. It also aids easy visualization of complex neural networks. In this thesis Tensorflow version, 1.8 is used.

1.4 Datasets

In order to evaluate the network model, we use T-LESS [12] and YCB datasets [13].

1.4.1 T-LESS

T-LESS is a dataset consisting of a collection of 30 industry-relevant texture-less objects captured from a systematically sampled view sphere with 10-degree steps in elevation and 5-degree steps in azimuth [12]. This RGB-D dataset is used for the SIXD challenge for object detection and pose estimation. The dataset comprises of objects recorded using three different sensors: a Microsoft Kinect v2, a Canon IXUS 950 IS camera and a Primesense Carmine RGB-D sensor. Moreover, it contains manually created and semi-automatically reconstructed CAD models of the objects. The Dataset also includes per-image bounding-box values of the objects with annotations of the form

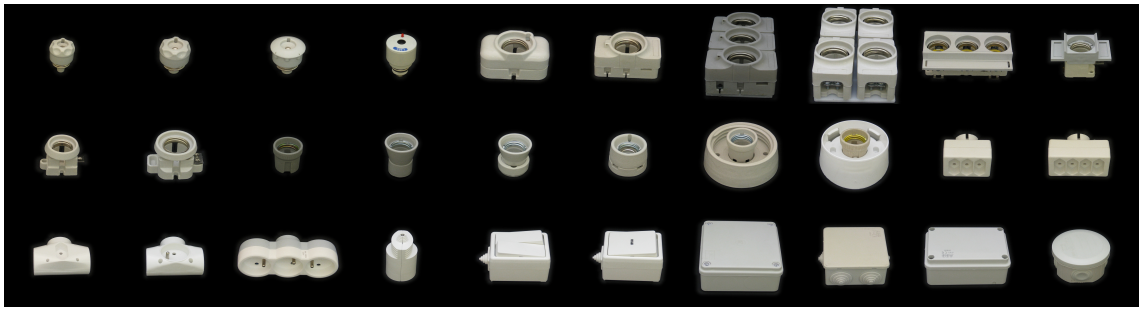


Figure 1.2: All T-less objects.

(x,y,x+w,y+h) where w and h are the width and height of the bounding box and the coordinate (x,y) is the upper left corner of the bounding box. Figure 1.2 shows the 30 T-LESS objects in ascending order. The T-LESS version v2 is used in this thesis.

1.4.2 YCB

Yale-CMU-Berkeley (YCB) dataset is a popular dataset used for robotic grasping and manipulation research. The dataset comprises of objects that are frequently used in daily life with varying sizes, shapes, textures, weight and rigidity. The database offers RGB-D images, high resolution RGB images, segmentation masks of the images, calibration information and 3D texture mapped mesh models [13]. The object set includes 77 objects that are widely used in manipulation tasks, divided in to 5 categories namely food, kitchen, shape, tool and task items. Figure 1.3 shows all 77 YCB objects. The dataset was prepared by using a scanning rig with 5 RGB-D sensors and 5 high resolution RGB cameras arranged in a quarter circular arc [13]. The objects were placed in a computer controllable turntable and then automatically rotated by 3 degrees at a time producing 120 turntable orientations.

Figure 1.3: All YCB objects.

Chapter 2

Related Work

This thesis focuses on generating synthetic images using GANs and to train object detectors with the generated images. Multiple approaches have been proposed for photorealistic image generation and this section covers information regarding some of the prominent works. This chapter includes the state of the art methods for synthetic image generation, Generative adversarial networks and then briefly reviews some of the well-known works in object detection to motivate our choice of object detectors.

2.1 Synthetic Images for Neural Networks

A great deal of work has been done to study the possibility of using synthetic datasets in image processing applications. Synthetic data usage has a well-established history in computer vision. An attempt by Nevatia et al. in which 3D CAD models were used for building object models is one of the earlier methods proposed in this field [14]. This work was proposed for solving the problem of scene analysis when multiple occluded objects are present or when specific objects in the scene are not known. The authors describe techniques to produce structured, symbolic description of complex curved objects in complex scenes by segmenting them into smaller subparts. The recognition is then further performed by comparing these descriptions with the stored description of 3D models of the objects. They demonstrated their results for a limited class of scenes. Several methods [15] [16] were proposed in which 3D CAD models were used as the labeled data for learning shape models to predict single object class like cars or motorcycles. Most of the proposed work uses a mixture of real and synthetic data to train neural networks. In [17] labeled image patches are used to generate synthetic images by using a method of cutting object instances and pasting them on a random background. Another approach by Hao Su et al. [18] uses 3D CAD models for viewpoint estimation using Convolutional neural networks. It out-performed the existing methods at that time on viewpoint estimation of 12 object classes from PASCAL 3D+ [19]. A similar type of work was done by Peng et al. [20]. They used rendered 3D CAD models both textureless and with texture by varying projections and orientations of the objects in PASCAL VOC 2007 dataset [21]. Their work demonstrated that augmenting the training data for contemporary Deep Convolutional Neural Networks (DCNN) is effective when there are few training samples or when the dataset is not matched to the target domain. The authors also investigated the sensitivity of convnets to various low-level cues in the training dataset such as 3D pose, foreground and background texture,

and colour. The deep convnet trained for the detection task was highly invariant to these cues. The training of the convnet using the synthetic images with simulated cues achieved the same performance as training on synthetic images without the cues. However, the authors also suggest that further experiments with other domains are necessary since their findings were preliminary. One of the recent approaches by Georgakis et al. [22] evaluated the usage of synthetically generated composite images by superimposing 2D images of related objects in real environments at different positions and scale for object instance detection. They demonstrated their results in GMU-Kitchens and Washington RGB-D scenes v2 dataset. Using hand labeled dataset together with synthetically generated dataset they achieved comparable performance to using only manually labeled data. However, all these approaches rely on real data to achieve competitive performances. Stefan Hinterstoisser et al. [23] presented a method of training object detectors with synthetic data generated from rendered 3D models. They employed a technique of freezing the weights of a feature extractor pre-trained on real data and adapting the weights of the remaining layers during training. Although they achieved good results using different object detectors, the 3D models used were extremely detailed.

2.1.1 Generative Adversarial Networks

Ian Goodfellow et al. [8] in 2014 introduced the concept of GANs. Mathieu et al. [24] and Denton et al. [25] used GANs for image generation tasks. In [24] they used a Laplacian pyramid structure for generators to produce high-resolution images. Many attempts have been made thereafter to improve the quality of the images generated by GANs. These attempts resulted in many variants of GANs, some of the prominent ones include conditional GANs [26], Invertible Conditional GANs [27], Deep Convolutional GANs (DC-GANs) [28]. Mirza and Osindero et al. proposed Conditional GAN, which used a conditional variable as one-hot encoding to control the output features generated by the GAN. Perarnau et al. extended the C-GAN by adding an encoder to the network to inverse the mapping for image editing applications resulting in Invertible Conditional GAN. Another approach by Radford et al. uses a Conv-Deconv architecture to improve the image generation of GANs, namely DC-GAN. Larsen et al. [29] proposed a method of combining VAE and GAN into an unsupervised generative model for synthetic images. Research in this field got extended to many applications from image inpainting, image editing, representation learning, style transfer, image to image translation, medical image augmentations etc.

2.2 Object Detection

Object detection is a well-researched area in computer vision. It deals with the process of detecting instances of objects from a scene and predicting their bounding boxes along with their corresponding class. There has been a lot of research in this field and some of the prominent and most used approaches are discussed here.

The Viola-Jones framework proposed in the year 2011 by Paul Viona and Michael Joans [30] is one of the simplest initial approaches and achieved near real-time performance. The work was intended for face detection and used Haar Features for generating binary classifiers. Another traditional approach suggested by Dalal and Triggs in 2005 [31] showed that HOG (Histogram of Oriented Gradients) and Linear Support Vector Machine (SVM) could achieve better accuracy

in object detection. However, this approach was much slower when compared to Viola-Joanes. Convolutional neural networks revolutionized this field. One of the first deep learning approaches is OverFeat [32] published in 2013 where they introduced a method of multi-scale sliding window algorithm using CNN. Region-based CNN (R-CNN) [33] was released soon after OverFeat and achieved 50% improvement on the object detection challenge VOC 2012. A region proposal method was employed in this approach to extract possible object regions and CNN for feature extraction followed by SVM for object classification. Although it achieved good results, the training process was slow. The state of the art research methods then started focussing on increasing the accuracy and speed of the neural networks for object detection. Fast-RCNN [34], Faster-RCNN [35], YOLO [36], R-FCN [37], SSD [38], RetinaNet [39] are some of the prominent works among them and are further discussed below.

Fast-RCNN This approach was proposed by Ross Girshick, as an extension of RCNN to achieve better performance. This network was faster and easier to train as compared to RCNN. Instead of extracting and applying classifier independently on the object proposal regions, this method applies a CNN on the complete image and then applies Region of Interest (RoI) pooling [34] on the feature map followed by final feedforward network for classification. This made the entire network end to end differentiable and therefore easy to train.

YOLO (You Only Look Once) The CNN, proposed by Redmon et al. achieved good performance and high speed. A single neural network divides the image into regions and predicts the position and probability of the object being in the particular region. The bounding boxes of the objects are then weighed by these probabilities.

Faster-RCNN The third iteration of RCNN, proposed by Ren et al. is an attempt to improve the performance of Fast-RCNN. The major drawback of Fast-RCNN is the usage of selective search algorithm for object proposals which increased the training time of the network. This method uses Region Proposal Network (RPN) instead of a selective search algorithm, which predicts objects depending on the "objectness" [35] score and these predicted objects are further processed by the RoI pooling and classifier.

R-FCN The idea behind Faster-RCNN to share the computations to enhance speed inspired Dai et al. to propose a new network known as R-FCN (Region based Fully Connected Network). The bounding box predictions are based on the regions of the last feature layer of the base network. In contrast to other approaches like Faster-RCNN this method employs fully convolutional network. The network produces position sensitive score maps which are then trained to detect certain parts of each object. This method could surpass the performance of its counterpart Faster-RCNN on PASCAL-VOC dataset [37].

SSD (Single Shot Detector) proposed by Liu et al. uses a single deep neural network for object detection. Unlike the above mentioned methods it does not use region proposal algorithms, instead produces a set of bounding boxes in different aspect ratios and scales based on feature map locations. The network structure of SSD is shown in Fig 2.1. The feature maps are based on the output of the base network and the final feature layer of the truncated base network which is obtained by progressively feeding it through strided convolution layers. VGG-16 [40] is used as the base network of the architecture and instead of the fully connected layers, auxiliary convolutional layers are used

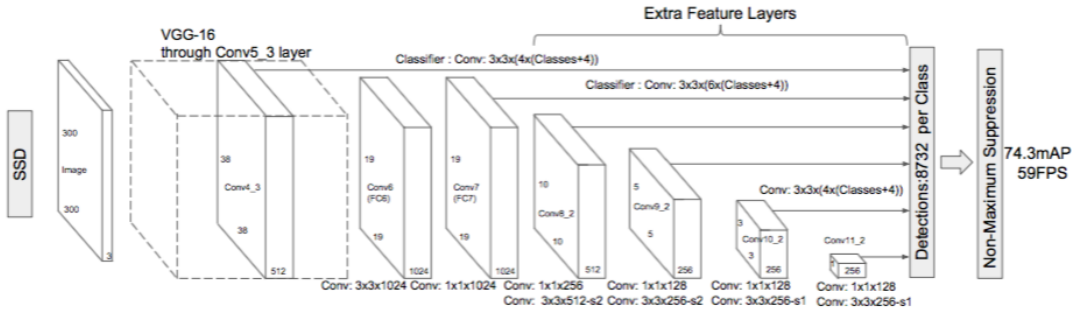


Figure 2.1: SSD architecture (adapted from [7]).

after conv 6, which results in multi-scale feature extraction. Non-maximum suppression in the final layer of the network retains the most accurate bounding boxes among the multiple box outputs and removes the noisier ones. SSD achieved 72.1 mAP (Mean Average Precision) for 300*300 input and 75.1 mAP for 500*500 input on VOC2007 dataset thus outperforming Faster-RCNN networks [38].

RetinaNet RetinaNet proposed by Lin et al. [41] is a single stage dense object detector characterized by loss function termed as Focal loss by the authors. Although the speed and the simplified structure of single stage detectors surpass the two-stage detectors, one of the major drawbacks is their comparatively low accuracy. This paper discusses the reason behind the lower accuracy of single stage detectors and argues that the extreme foreground-background class imbalance faced during the training process is the major reason behind their low accuracy. To overcome this imbalance they reshaped the standard cross-entropy loss such that the well classified and easy examples are down-weighted and the loss is focused on the hard examples. Focal loss thus focuses on the sparse set of difficult examples during training. The RetinaNet architecture is shown in Fig 2.2. The architecture comprises Feature Pyramid Network (FPN) as backbone on top of a feed forward network which is a Resnet architecture. The FPN is basically a standard convolutional network with an additional top down pathway and lateral connections as shown

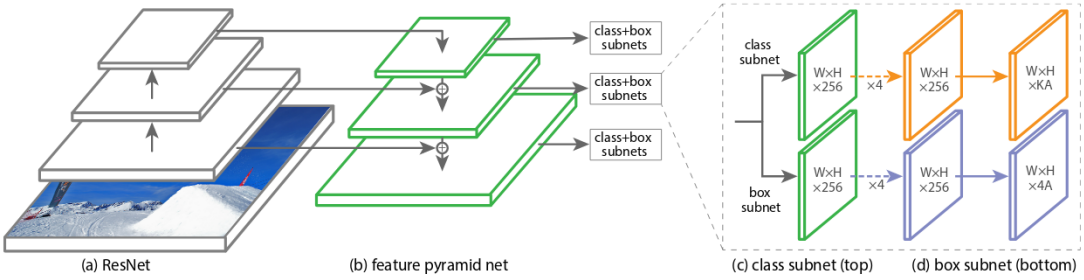


Figure 2.2: Retinanet architecture (adapted from [41]).

in the figure used to produce a rich, multi-scale feature pyramid from a single resolution image. Translation-variant anchor boxes are used in RetinaNet. Two sub networks are attached to the backbone namely a classification and a box regression subnet. The classification subnet predicts the probability of the object classes in each anchor boxes and the box regression subnet is used to regress the offset of the anchor boxes to the nearby ground-truth object. RetinaNet is also the first single stage object detector that matches the state of the art COCO average precision (AP) of the other complex two-stage detectors, such as variants of Faster R-CNN. In order to evaluate the quality of the generated images from GAN, RetinaNet object detector is used in this work.

Chapter 3

Background and Theory

3.1 Generative Adversarial Networks

The concept of GANs was introduced by Ian Goodfellow et al. in the year 2014 [8]. GANs consist of two network structures called generator and discriminator. The generator produces images that look like real images and the discriminator tries to distinguish between the generator output and the real images.

Generator In a basic GAN architecture, the generator is basically a deep neural network which takes latent noise distribution as its input as shown in Fig 3.1. During the training process the generator takes the feedback of the discriminator and updates its weights accordingly during back propagation. Thus the generator gradually produces images similar to the training dataset to fool the discriminator.

Discriminator The discriminator is fundamentally a supervised classifier that tries to classify its input images as either 'real' or 'fake'. The structure is again a neural network with sigmoid as the last layer activation function to output the probability of the images being 'real' or 'fake'. As shown in the Fig 3.2 the discriminator should output 1 when the input image is from dataset and 0 when the input image is the generator output (fake output).

The generator along with the discriminator forms the entire GAN structure as shown in Fig 3.3. In the training process the generator tries to fool the discriminator by generating images

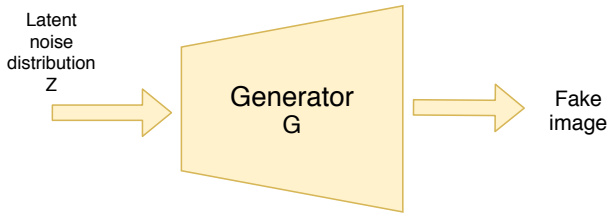


Figure 3.1: Structure of generator.

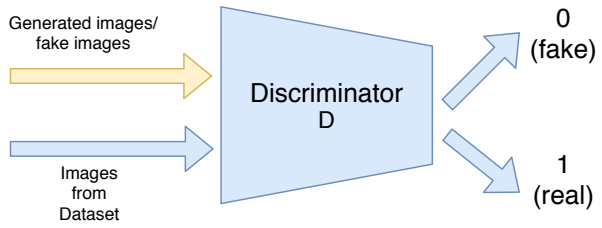


Figure 3.2: Structure of discriminator.

similar to the images in the dataset and the discriminator tries to distinguish between the images from the dataset and the fake images from the generator. Thus the training of GANs can be viewed as a minimax two-player game between the generator and discriminator.

3.1.1 Working Principle of GANs

The generator and discriminator are two functions which are differentiable with respect to their inputs and parameters. The generator function is represented by G with the parameters θ_G and the discriminator function is represented by D with parameters θ_D. If the distribution of data in the dataset (also called real distribution) is x, then the generator objective is to learn a distribution P_{model} over data x. The input to the generator is a latent noise distribution z and the input to the discriminator is either the output from the generator or a sample of the real distribution x. $D(x, \theta_D)$ outputs a single scalar value representing the probability of x being real. The objective of the generator is to produce samples from the given data. The discriminator can be considered as an opponent to the generator. The objective of the discriminator is to observe the samples from the generator and given data and to identify the real data and the fake data(generator samples). The value of the discriminator is closer to 1 when it detects real data and closer to 0 when it detects generated images. The discriminator loss function is thus defined in Equation (3.2).

$$J_D(\theta_D, \theta_G) = -\frac{1}{2}\,\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] - \frac{1}{2}\,\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))], \tag{3.1}$$

where:

$\mathbb{E}$ = Expectation or expected value

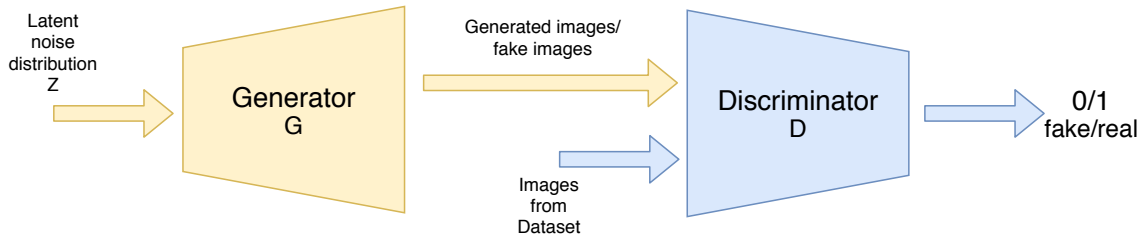


Figure 3.3: GAN architecture.

It is clear from the equation above that the loss of the discriminator will be 0 if the output of the discriminator for real images, $D(x)$ is 1 and the output for the generated images, $D(G(z))$ is 0. In order to define the loss function for generator, we consider the entire scenario as a zero-sum game. In a zero-sum game the sum of the loss function of the generator and discriminator is zero. The loss function of the generator, $J_G(\theta_G, \theta_D)$ is defined as

$$J_G(\theta_G, \theta_D) \; = \; -J_D(\theta_D, \theta_G) \tag{3.2}$$

To summarize, the discriminator is trained to maximize the probability of assigning the correct scalar values for both the images from the dataset and the generator images, whereas the generator is trained to minimize $\log\left(1 - D(G(z))\right)$ so that $D(G(z))$ will be closer to 1. Thus the GANs minimax objective function $V(G, D)$ can be represented as in

$$\min_G \max_D V(G, D) \; = \; \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] \; + \; \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log\left(1 - D(G(\boldsymbol{z}))\right)], \tag{3.3}$$

where:

$\mathbb{E}$ = Expectation or expected value

The cost function J of the generator and the discriminator are defined in terms of both networks parameters. The discriminator is trained to minimize $J_D(\theta_D, \theta_G)$ and can only control its parameter θ_D. Similarly generator tries to minimize $J_G(\theta_D, \theta_G)$ with control only in its parameter θ_G. Since each network's loss function depends on the other network's parameters and does not have control of other network's parameters, the GANs scenario is considered as a game problem rather than as an optimization problem. Therefore the solution to this two mini-max player game is Nash equilibrium which is the optimal point for the mini-max function of GANs [42]. The Nash equilibrium of GAN is a tuple (θ_D, θ_G), local minimum of the cost function of both players (generator and discriminator) with respect to their own parameters. If the discriminator receives a fake output from the generator $G(z)$, it tries to make $D(G(z))$ equal to 0 while the generator tries to make it 1. Then the Nash equilibrium would be $G(z)$ being drawn from the same distribution as the training dataset, and $D(x) = \frac{1}{2}$ for all x. Please refer to Appendix for derivation.

3.1.2 Training stability of GANs

The training procedure of GANs comprises simultaneous updates of both generator and discriminator parameters. On each step of the training process, a batch of m samples from the training dataset x and a batch of z values from the noise prior are sampled. The two gradient steps are made simultaneously here, one to update the discriminator parameters θ_D to reduce the discriminator cost function J_D and the other to update the generator parameters θ_G to reduce the generator cost function J_G. The choice of the gradient-based optimization algorithm for both cases depends on the application; however, Adam [43] is usually considered a good choice. Adam stands for adaptive moment estimation and is a widely used optimization algorithms to update network weights during training. Our method also uses this optimization algorithm and further details will be discussed in the following Chapters. Practically while training the network, instead of training G to minimize $\log\left(1 - D(G(z))\right)$ we train G to maximize $\log(D(G(z))$. This is computationally less expensive, and according to the original paper, it also provides better gradients in learning. That is when the generator samples are not yet accurate or close to the real data samples, the discriminator easily learns to differentiate between real and false data samples, thereby saturating

$\log(1 - D(G(z)))$.
The GANs training algorithm is summarized as follows:

Algorithm 1 GAN algorithm

```
for number of training iterations do
    for k steps do
        Sample batch of m noise samples from noise prior p_g(z).
        Generate m images from the noise prior.
        Sample batch of m samples from the training dataset.
        Update discriminator parameters.
    end for
    Sample batch of m noise samples from noise prior p_g(z).
    Generate m images from the noise prior.
    Update generator parameters.
end for
```

The number of steps k to train the discriminator is a hyper-parameter and the original paper used the least expensive option, $k = 1$. The training of GANs is usually considered as hard because of the difficulty in finding a balance, more specifically to reach the nash equilibrium between its two players: generator and discriminator. The major problem involved in the training scenario is the objective of convergence. The following section discuss the major problems faced while training GANs.

One of the most commonly encountered problem in training GANs is called Mode Collapse or sometimes the Helvetica scenario where the generator collapses producing limited varieties of samples. The reason for this collapse can be related to the game theory approach in GANs. The objective of G is to minimize $\mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$ that is to generate a point x^* such that $x^* = argmax_x D(x)$. It is assumed that discriminator is held constant during this time. Since x^* is fixed regardless of the value of z, it depends only on the discriminator at a given time step. This denote that on expectation, there exists a single point in space that the generator assumes as the optimal point to generate output regardless of the noise input z. There is no specific function in the generator loss function that explicitly forces the generator to produce different samples for the given input. As a result of this, the generator will map all the input values to that same most likely to be the real point. The most extreme condition in mode collapse is the generator producing one image for all possible inputs. The hyperparameters of the GANs must be chosen appropriately since there are high chances of either one of the networks to diverge or stop learning. Since the training involves both networks, it is commonly observed either that one of the networks is stronger than the other meaning the gradient from the loss function could be zero easily. This is referred to as vanishing gradients problem. Since the network is highly sensitive to hyperparameters, failing to determine the optimal values for these parameters results in an unbalance between the generator and discriminator causing overfitting.

In order to stabilize the training of GANs several methods and architectural modifications of the network have been proposed. The methods to use highly depends on the application and all of the existing stabilizing methods may not improve the performance of GANs in certain scenarios. The problems that were faced in training GANs in our approach and the techniques used to

stabilize our GANs will be discussed in detail in Chapter 4 and Chapter 5.

3.2 Image Conditioned Adversarial Networks

Many variants of GANs have been proposed and are fundamentally an extension of the basic GAN framework. Among them, our focus is on the GANs conditioned with images that learn the generative model of image data conditioned on image data. A source image and a target image is provided as input along with the noise for Generator1 and Generator2 respectively. The target image is provided as input to Discriminator1 and source image to Discriminator2. The resulting architecture translate the images from the source domain to the target domain. This facilitates the usage of GANs in domain adaptation applications where images in one domain are translated to another domain. (further discussed in the next section) Some variants of image conditioned GANs are discussed in the next section.

3.2.1 Coupled GAN (CoGAN)

CoGAN learns the joint distribution between the images in both domains without any paired training dataset. CoGAN comprises a pair of GANs, each for producing images in one domain and discriminator corresponding to each generator to identify the real or fake images in each domain. As shown in Fig 3.4, GAN1 and GAN2 are the two generators. During training the generators are forced to share their weights or parameters in the initial stages and the discriminators are forced to share their weights in the last few layers [44]. These layers which are shared via weights are responsible for encoding high-level semantics in the network. The weight sharing concept is introduced with the objective of learning a joint distribution of the images in both domains. The CoGAN thus facilitates generation of a pair of images sharing the same high-level abstraction and different low-level realizations.

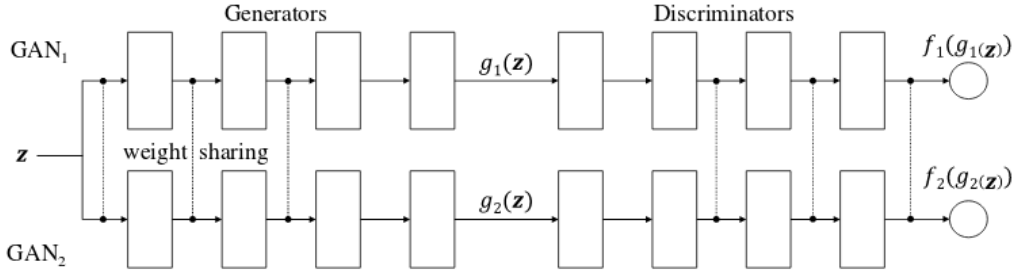


Figure 3.4: CoGAN architecture (adapted from [44]).

3.2.2 PixtoPix GAN

PixtoPix GAN is an image to image translation network using conditional adversarial GANs [45]. This framework employs a conditional generative adversarial network to learn a mapping between source and target images. This network requires paired dataset for its operation and is a generic approach to many traditional image to image translation applications. In addition to learning the mapping between two domains these networks also learn the loss function to train the mapping. The generator architecture is a U-Net architecture [46] featured by skip connections in each layer allowing the low-level information to shortcut across the network. The discriminator architecture is a PatchGAN, which classifies the patches of fixed sizes of images as real/fake. This network learns a loss function according to the task and data at hand, which enhances its wide range of applicability in various image to image translation applications.

3.2.3 CycleGAN

CycleGAN [47], DualGAN [48] and DiscoGAN [49] are networks with similar architectures. These networks share a lot of similarities, with slight variations in their loss functions. Our research is focused on the CycleGAN framework. CycleGAN is built upon the pixtopix GAN but does not require paired dataset to learn the mapping between the source and the target domain. Paired datasets are expensive and not always available for all the applications, which makes this framework much reliable than pix2pix. The architecture comprises two pairs, each consisting of one generator and one discriminator. Each generator is used for mapping the images from one domain to another and their corresponding discriminators to distinguish the real and fake images. The loss function includes an adversarial loss to generate images which is indistinguishable from the target domain and an additional cycle consistency loss to enforce the inverse mapping from the target to the source domain. The intuition behind this additional cyclic loss is that if an image is translated from one domain to another it should translate back to the initial domain when an inverse mapping is performed. This idea is clearly illustrated in the Fig 3.5. Our work is inspired from the architecture of CycleGAN network and we adopt the CycleGAN architecture for further research on domain adaptation task.

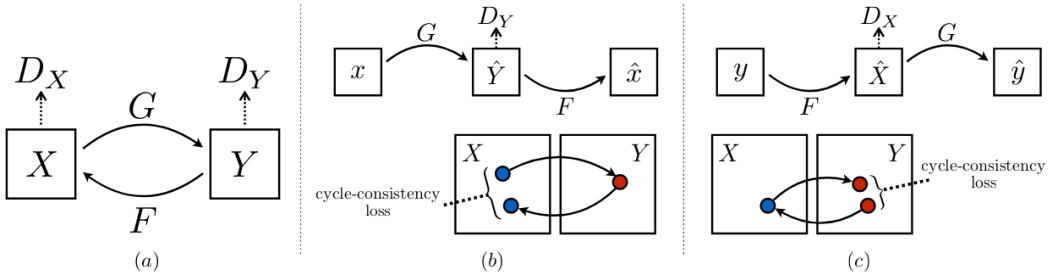


Figure 3.5: CycleGAN illustration (adapted from [47]).

3.3 Domain Adaptation

Domain adaptation as the name suggests refers to an algorithm that transfer between two domains. These two domains are generally called as source and target domain. Domain adaptation can be achieved either by translating source domain to target domain or by finding a common embedding between the two domains. One of the most interesting and potential research area is unsupervised domain adaptation where the source images have labels and target images do not. Unsupervised domain adaptation has advanced incredibly over the years. With the recent emergence of adversarial domain adaptation, the performance and the results of domain adaptation field has improved a lot. Adversarial domain adaptation refers to the training of two networks namely generator and discriminator such that the generator tries to produce target domain looking images from source domain, whereas discriminator tries to distinguish the target domain images and the transformed source domain images from the generator. This is an extension of basic GAN in such a way that instead of providing continuous random distribution to the network we input source domain images to the network. Many variants have been proposed so far with this underlying concept, among which some of the prominent ones are mentioned in the previous sections. In this thesis, we focus on the domain adaptation of synthetic images to real images for object detection.

Chapter 4

CycleGAN for Domain Adaptation

CycleGAN is used to adapt rendered images to real-world images. The architecture works on unpaired dataset which enhances the applicability of the framework. The images from both domains are taken as input to the network. The source domain images are rendered images and the target domain images are real images. This chapter discusses the network architecture, loss functions and the training procedure adopted.

4.1 Network Architecture

As mentioned in the previous chapter, CycleGAN comprises of two pairs of generator and discriminator. Generator1 - Discriminator1 pair transfer the images from synthetic domain to the real domain and Generator2 - Discriminator2 pair transfers the images from real domain to the rendered domain. Although our focus is in transferring the images from synthetic domain to real domain, we implemented both transformations. The architecture of our network is further discussed below.

4.1.1 Generator Architecture

CycleGAN consist of two generator networks. The generator architecture used is shown in Fig 4.1. The generator contains three main modules namely, encoding module, transformation module and decoding module. The input image of size $(b, w, h, 3)$, where b is the batch size of the images, w is the width of the image and h is the height of the image, is provided as input to the encoding module. The encoding module is used to extract the high level features of the input image and comprises of three convolutional layers. The output size after the encoding module is $(b, w/4, h/4, 128)$. Resnet blocks are used for the transformation module. The output size after the transformation module will therefore be the same $(b, w/4, h/4, 128)$. The decoding module is used to reconstruct the images from the input domain to the target domain based on the transformation features. It consists of two deconvolution layers to reconstruct the width and height of the image and a convolution layer in the last to reconstruct the number of channels. Thus the output of the decoding module is same as the input of the encoding module $(b, w, h, 3)$. The filter size, number of filters and stride used in each layer of the generator is summarized in Fig 4.2. ReLU activation and instance normalization is used in the encoding module of the generator along with symmetric

reflection padding of size 3.

Figure 4.3 shows the illustration of a resnet block used in the transformation module of the generator. The major feature of resnet block is the identity shortcut connection between the input and output layers. Resnet block skips one or more hidden layers in the network as specified. In these networks the input is added to the output which concludes that the resnet block will not produce a worse output than an identity mapping (since the input is always fed in to the network). This architecture also helps in reducing the chances of gradient vanishing problems. We use 6 resnet blocks in the transformation module. Symmetric reflecting padding of size 1 is applied to the image dimensions with convolution layer of kernel size 3 and stride 1.

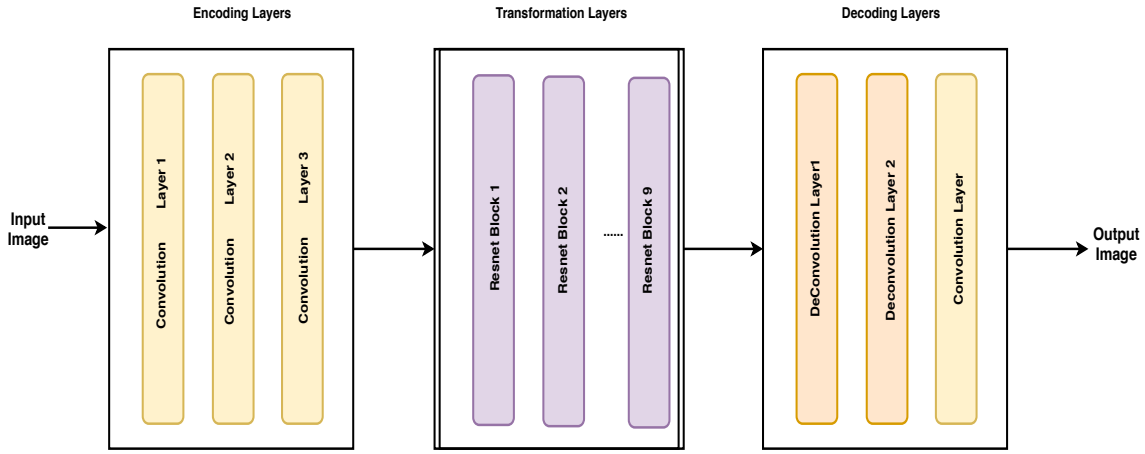


Figure 4.1: Generator architecture.

Layer	Kernel	Stride	Channels	Padding
Conv1	7 x 7	1	32	REFLECT
Conv2	3 x 3	2	64	x
Conv3	3 x 3	2	128	x
Deconv1	3 x 3	2	64	x
Deconv2	3 x 3	2	32	x
Conv4	7 x7	1	3	REFLECT

Figure 4.2: Generator layer specifications.

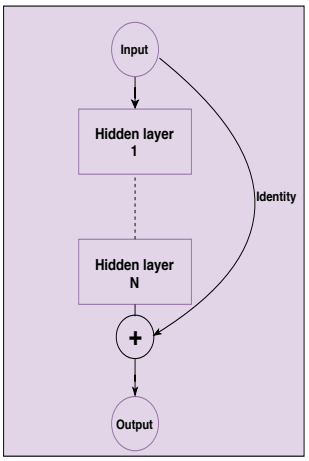


Figure 4.3: Resnet block illustration.

4.1.2 Discriminator Architecture

The Discriminator architecture is shown in Fig 4.4. The discriminator consist of 5 convolution layers and the specification of these layers is shown in Fig 4.5. The slope parameter of the leaky RELU activation used is 0.2. Instance normalization is used in the first four convolution layers. The input to the discriminator is fed through image pooling algorithm which is further explained in the section 4.3.

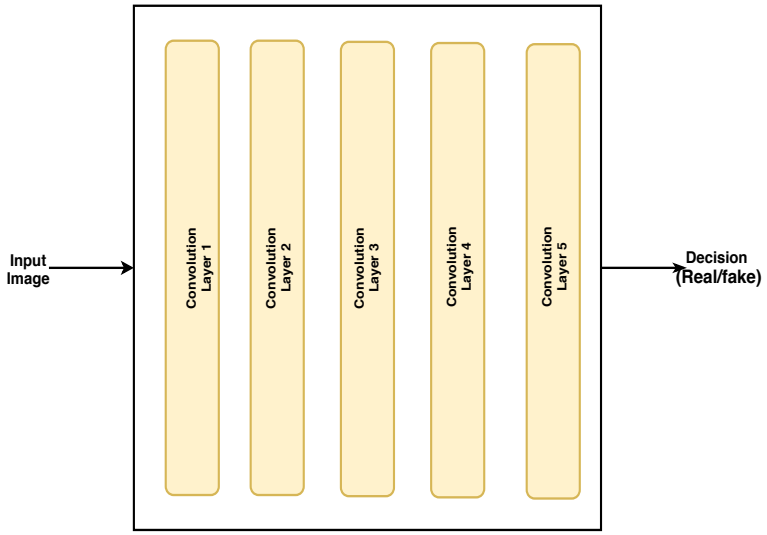


Figure 4.4: Discriminator architecture.

Layer	Kernel	Stride	Channels	Activation
Conv1	4 x 4	2	64	Leaky RELU
Conv2	4 x 4	2	128	Leaky RELU
Conv3	4 x 4	2	256	Leaky RELU
Conv4	4 x 4	1	512	Leaky RELU
Conv5	4 x 4	1	1	x

Figure 4.5: Discriminator layer specifications.

4.1.3 CycleGAN Architecture

The CycleGAN architecture used is shown in Fig 4.6. The architecture comprises of mainly two generator-discriminator pairs. The top portion of the architecture depicts the network structure used for the conversion of the simulated images to the real images and the bottom one for real images to simulated images. Since we are interested in generating real images from simulated images, further sections will be focused on the top part of the architecture. The simulated image X is given to the Generator $X \rightarrow Y$ as input. The output of the Generator is provided to the Discriminator Y along with the real image Y. The Discriminator evaluates the Generator output by comparing it to the real image and predicts whether this output is real or fake. Since we are using unpaired dataset there exist no meaningful input-output relationship information for our network to transform images from simulated domain to real domain. Therefore another generator, Generator $Y \rightarrow X$ is added to reconstruct the original image from the output image of Generator $X \rightarrow Y$. By adding this generator we enforce that there exists some common features between the input image and the output image that can be used by the Generator $Y \rightarrow X$ to reconstruct the original image from the output image. The bottom part of the architecture is similar to the one explained above with real image as input to the Generator $Y \rightarrow X$.

4.2 Loss Function

A suitable loss function has to be formulated for the model in order to accomplish the required goal. The discriminator should approve for all real images (output 1) and reject (output 0) the corresponding generator images. The generator on the other hand should make the corresponding discriminator approve the generated images. The generator should also output images in such a way that when an inverse operation is performed it should reproduce its input image, that is it should satisfy the so called cycle consistency. Taking into consideration the above goals the loss function of our model is formulated as below. As explained in the previous section we have two generators (G_{xy} and G_{yx}) and two discriminators (D_x and D_y). The simulated input image is denoted as I_s and the real image as I_r.

Generator loss In order to fool the corresponding discriminator, the generator should be

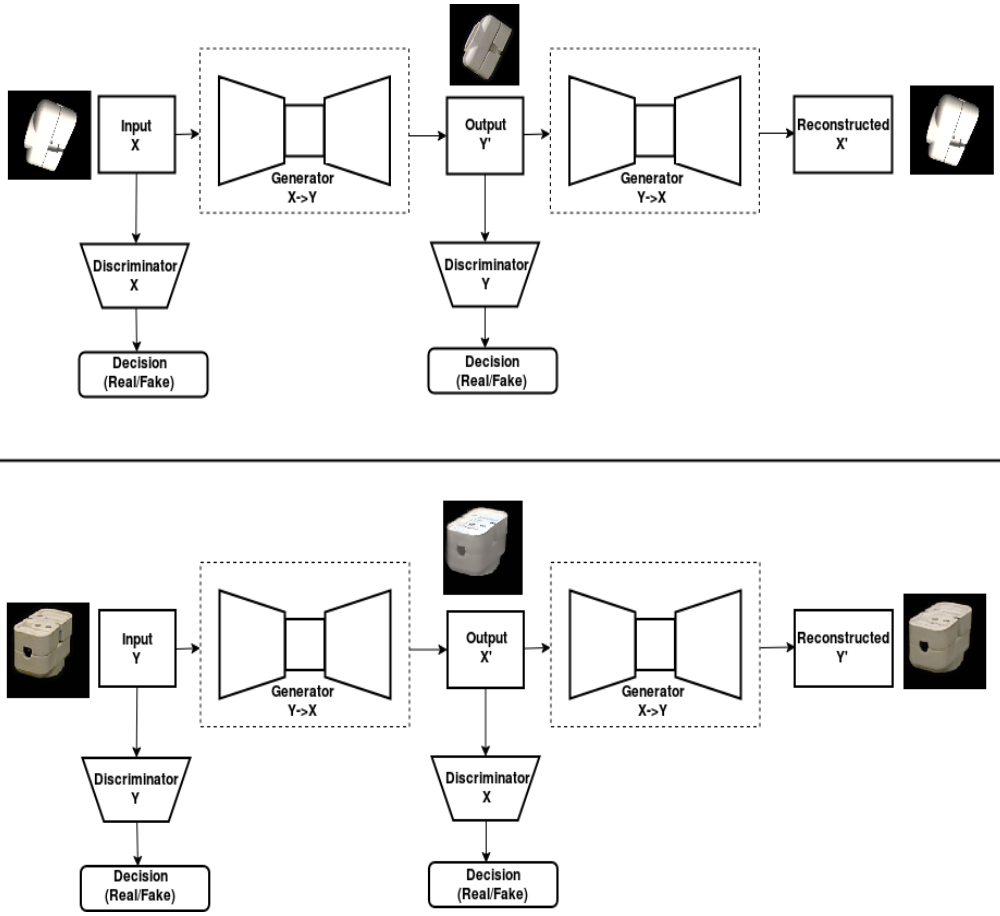


Figure 4.6: CycleGAN architecture.

able to make the discriminator approve the generated images, that is to make the discriminator output 1 for the generated images. The generator should also output images such that when reconstructed produces the original input image. To achieve this, reconstruction loss, also defined as cycle-consistency loss is used and is denoted in Equation (4.1).

$$\mathcal{L}_{cyclic}(I_s, I_r) = \lambda_1 * MAE(I_s - G_{yx}(G_{xy}(I_s))) \; + \; \lambda_2 * MAE(I_r - G_{xy}(G_{yx}(I_r))) \tag{4.1}$$

It is calculated with $L1$ reconstruction error, also known as mean absolute error (MAE) between the original input image and the reconstructed image. The terms λ_1 and λ_2 in the equation is termed as cycle loss coefficients and is considered as hyperparameter during the training process. The adversarial loss (least square loss) combined with the cyclic reconstruction loss constitutes the generator loss of CycleGAN. The loss functions for the two generators $\mathcal{L}_{Gxy}$ and $\mathcal{L}_{Gyx}$ are depicted in Equation (4.2) and Equation (4.3) respectively.

$$\mathcal{L}_{Gxy}(I_s, I_r) = (1 - D_x(G_{xy}(I_s)))^2 \; + \; \mathcal{L}_{cyclic}(I_s, I_r) \tag{4.2}$$

$$\mathcal{L}_{Gyx}(I_s, I_r) = (1 - D_y(G_{yx}(I_r)))^2 \; + \; \mathcal{L}_{cyclic}(I_s, I_r) \tag{4.3}$$

Discriminator loss The discriminator should be able to differentiate the generated images and the input images. The loss functions for the two discriminators $\mathcal{L}_{Dx}$ and $\mathcal{L}_{Dy}$ are depicted in Equation (4.4) and Equation (4.5) respectively.

$$\mathcal{L}_{Dx}(I_s, I_r) = (1 - D_x(I_s))^2 + (D_x(G_{yx}(I_r)))^2 \tag{4.4}$$

$$\mathcal{L}_{Dy}(I_s, I_r) = (1 - D_y(I_r))^2 + (D_y(G_{xy}(I_s)))^2 \tag{4.5}$$

4.2.1 Mask Loss

We propose some modifications to the loss function of the CycleGAN network for improving the quality of the images generated. The generated images should maintain the exact position of the input images for further object detection or pose estimation applications. An additional loss termed as "Mask loss" is added to the generator loss function. As mentioned above since we are focusing on the generation of real images from simulated images this additional loss term is added only to the Generator $X - Y$. In addition to the real image of the object, the mask of the object is also given as input to the generator. The modified architecture is shown in Fig 4.7. In the modified

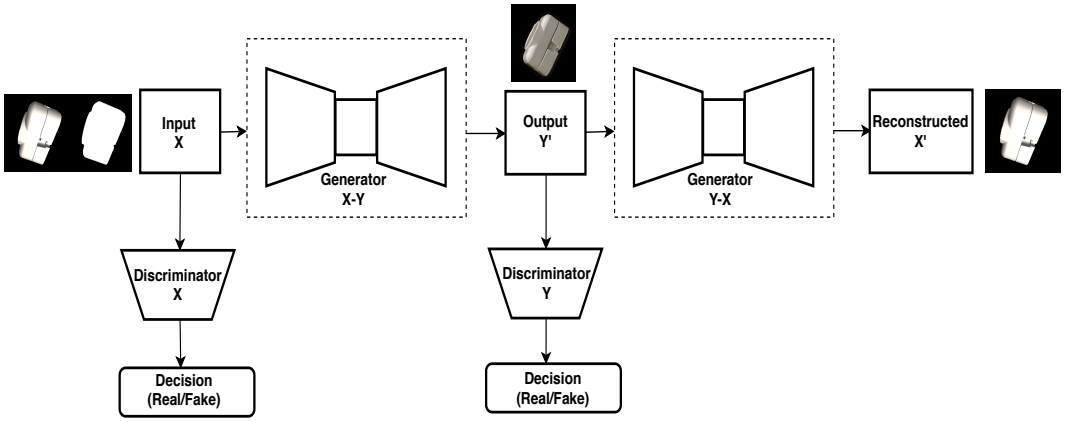


Figure 4.7: CycleGAN architecture with mask loss.

architecture the Generator $X-Y$ produce the output image along with the mask output. The mean absolute error (MAE) of the input mask and the output mask constitutes the mask loss $\mathcal{L}_{mask}$ and is depicted in Equation (4.6) where $mask_{input}$ corresponds to the mask of the real image and $mask_{output}$ corresponds to the mask of the generated output.

$$\mathcal{L}_{mask} = MAE(mask_{input} - mask_{output}) \tag{4.6}$$

In addition to the mask loss we also use another loss function termed as "Identity loss" in the generator loss function. The objective of the identity loss is to ensure that the generator produce an identity mapping when the target domain images are given as its input. This idea was first proposed in [50]. The identity loss function used is given in Equation (4.7).

$$\mathcal{L}_{identity} = (\lambda_1/2) * MAE(G_{yx}(I_s), I_s) + (\lambda_2/2) * MAE(G_{xy}(I_r), I_r) \tag{4.7}$$

Therefore the generator loss function of our CycleGAN model is the combination of the mask loss, identity loss and the loss function specified in previous section. The modified generator loss function used in our model is given in Equation (4.8).

$$\mathcal{L}_{Gxy(new)} = \mathcal{L}_{Gxy} + \mathcal{L}_{mask} + \mathcal{L}_{identity} \tag{4.8}$$

4.3 Training Techniques and Algorithm

After the formulation of suitable loss function for our model the next step is to train the network. As mentioned in Chapter 3, training of GAN is difficult and we should adopt suitable techniques to stabilize the training of the generator and discriminator. This section discuss the methods adopted to train our model efficiently and the training algorithm used.

4.3.1 Image Pool Algorithm

Image pool algorithm was used for the first time in [51] as an effective method for stabilizing the training of GAN. The method involves updating the discriminator using a history of generated images rather than using the current generated image. When discriminator is updated using a current image from the generator, there occurs lack of memory in the discriminator which leads to the divergence during training. This will end up in generator re-producing artifacts in the generated images that the discriminator has failed to memorize. This is a major problem because the generator network should always try to model real image characteristics rather than introducing new artifacts to its output. Image pool algorithm is used to tackle these two major limitations that occur during training process. We update the discriminator using an image buffer that stores 50 previously produced generator images.

4.3.2 Instance Normalization

Instance normalization (IN) [52] is a technique used in deep learning to standardize the internal description of data to enhance the neural network training. Instance normalization can be considered as an instance of batch normalization [53] with batch size 1. Batch normalization (BN) was introduced to resolve one of the most common limitations of deep neural networks termed as internal covariant shift. The Covariant shift is described as the changes in the input value distribution of a learning algorithm. Since deep learning algorithms behave differently for different input distributions, if the train and test set have different input distribution, deep learning model fails to generalize the results. If we consider a dense neural network with many hidden layers, the parameters of each layer changes over the course of training. As a result of this, the activation of each layer also differs. The activations of the earlier layers are input to the following layers and thereby for each hidden layers, the input distribution changes with each step during training. Hence, during training, each layer of the network are forced to adapt to its changing inputs. This becomes problematic when the covariant shift occurs in the dataset of these networks. Batch normalization overcomes this limitation by normalizing the activation of each layer in the network. This enhances the network layers to learn on more stable data distribution. The batch normalization algorithm is shown below. The steps include transforming the inputs to 0 mean and unit variance. However, restricting the input to 0 mean and unit variance, can sometimes limit the expressive power of the network, in practice we add two parameters γ and β so that the network will convert them to

the most desired mean and variance values. Instance normalization is widely used in the domain adaptation and style transfer applications. In [52], its shown that using IN instead of BN can remove instance specific contrast information from source domain images in domain adaptation tasks. We used IN as the normalization method for our CycleGAN architecture.

Algorithm 2 Batch Normalization algorithm

Input : Values of x over a mini-batch: $\mathbf{B} = \{x_{1...m}\}$
Parameters to be learned β,γ
Output : $\{y_i = BN_{\gamma\beta}(x_i)\}$

$\mu_\beta \leftarrow \frac{1}{m}\sum_{i=0}^{m} x_i$ // mini-batch mean

${\sigma_B}^2 \leftarrow \frac{1}{m}\sum_{i=0}^{m}(x_i - \mu_\beta)$ // mini-batch variance

$\hat{x_i} \leftarrow \frac{(x_i - \mu_\beta)}{\sqrt{\sigma_B^2 + \epsilon}}$ // normalize

$y_i \leftarrow \gamma * x_i + \beta \equiv BN_{\gamma\beta}(x_i)$ // scale and shift

4.3.3 One-sided Label Smoothing

Label smoothing is an effective technique used for reducing the over confidence of deep neural networks. This is a method of smoothing the target labels of images to values between 0 and 1 instead of 0 and 1. Over confidence occurs in neural networks when a small set of features is used during learning. If discriminator uses only a small set of features to identify real and fake images, the generator network playing as an opponent to the discriminator learns to produce only these features to fool the discriminator. This will end up in a greedy optimization objective and the learning does not occur as intended. By penalizing the discriminator output, the over confidence of the network is reduced and therefore ensures a better learning. Instead of smoothing both target label values of the discriminator, only the labels of real images are smoothed to a value α instead of 1. Hence the name one sided label smoothing. The reason behind softening the probability of real image target labels can be explained by taking into account the optimal discriminator function of the discriminator. The optimal discriminator function is defined as below (the derivation of this equation is given in Appendix)

$$D_{\text{opt}}(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_{\text{model}}(\boldsymbol{x})} \tag{4.9}$$

If we use α for real image target and β for fake target label then the Equation (4.9) becomes,

$$D(\boldsymbol{x}) = \frac{\alpha p_{\text{data}}(\boldsymbol{x}) + \beta p_{\text{model}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_{\text{model}}(\boldsymbol{x})} \tag{4.10}$$

From the above equation it is clear that in regions where p_{data} is zero, p_{model} will be very large, which causes the fake samples from the generator not moving closer to the real data. In order to avoid this limitation, p_{model} term in the numerator should be removed. Thus β is chosen as 0. We soften the probability of real image output of the discriminator by $\alpha = 0.9$. The modified generator loss functions are given in Equation (4.11) and Equation (4.12).

$$\mathcal{L}_{Gxy}(I_s, I_r) = (\alpha - D_x(G_{xy}(I_s)))^2 \; + \; \mathcal{L}_{cyclic}(I_s, I_r) \tag{4.11}$$

$$\mathcal{L}_{Gyx}(I_s, I_r) = (\alpha - D_y(G_{yx}(I_r)))^2 \; + \; \mathcal{L}_{cyclic}(I_s, I_r) \tag{4.12}$$

The discriminator loss function is modified as in Equation (4.13) and Equation (4.14).

$$\mathcal{L}_{\mathrm{Dx}}(I_s, I_r) = (\alpha - D_x(I_s))^2 + (D_x(G_{\mathrm{yx}}(I_r)))^2 \tag{4.13}$$

$$\mathcal{L}_{\mathrm{Dy}}(I_s, I_r) = (\alpha - D_y(I_r))^2 + (D_y(G_{\mathrm{xy}}(I_s)))^2 \tag{4.14}$$

4.3.4 Training Algorithm

The training algorithm adopted for our network is summarized in Algorithm 2. It was observed that the discriminator network of our CycleGAN architecture was comparatively stronger than the generator network meaning that the loss function of the discriminator converged to 0 quickly when compared to the generator network. In order to overcome this problem we train our network by giving a head start to the generator and by training the generator more number of steps, than the discriminator. A batch of source domain data are sampled and fed to the generator network and trained m number of steps. The discriminator is given a randomly selected image buffer that contains 50 previously generated images by the corresponding generator. Image pool algorithm is used randomly during training. The discriminator also takes the target domain samples as input. The discriminator is trained n number of steps. The entire training is then repeated for the given number of iterations. The values for m, n and the number of iterations are all considered as hyperparameters and are mentioned in the appendix.

Algorithm 3 CycleGAN algorithm for simulated to real domain adaptation ($m > n$)

```
for number of iterations do
    for m steps do
        Sample a batch of k source domain samples
        Generate m images from the samples
        Update the generator parameters
    end for
    for n steps do
        Sample an unit uniform random variable p
        if p > 0.5 then
            Feed the discriminator with the newly generated image
        else
            Randomly selects images from image pool of size 50
            Replace Image pool with the newly generated images
        end if
        Sample image from target domain I_r
        Update discriminator parameters
    end for
end for
```

Chapter 5

Experiments, Results and Discussions

In order to evaluate the quality of the domain adapted images obtained from the CycleGAN network, these images are used for training an object detector. The state of the art object detector RetinaNet is used here. The performance of the object detector trained using the CycleGAN generated images are evaluated and then compared to the performance on using real images and CAD rendered images. The datasets used are the challenging T-LESS dataset, an RGB-D dataset used for object detection and pose estimation of texture-less objects, and YCB dataset for textured objects.

5.1 Evaluation Metric

The evaluation metric used for measuring the accuracy of object detectors is termed as mAP (mean Average Precision).

5.1.1 Mean Average Precision

Mean average precision is calculated by taking the average precision of each class and then taking the mean over the total classes. mAP values range from 0 to 100 and a higher mAP value defines a better model. Average precision is defined as the average of the maximum precision at 11 recall levels, $r \in \{0, 0.1, ..., 1\}$. The term precision measures the percentage of positive predictions of a class and recall measures the number of correct predictions from the existing ground truth predictions of that class. A prediction is correct only if it matches the ground truth with intersection over union also termed as IoU $>= 0.5$. IoU measures the amount of overlap between two regions, in object detection we calculate the amount of overlap between the predicted bounding box and the ground truth bounding box. The mathematical definitions of precision and recall are depicted in Equation (5.1).

$$\begin{aligned} Precision &= \frac{TP}{TP + FP} \\ Recall &= \frac{TP}{TP + FN}, \end{aligned} \tag{5.1}$$

where:

TP = true positive
FP = false positive
FN = false negative

In order to calculate the recall level, a softmax prediction threshold is varied and then the [precision (p), recall (r)] pairs are interpolated. The Average Precision (AP) is defined in Equation (5.2).

$$AP = \frac{1}{11} \sum_{r \in \{0, 0.1, \ldots, 1\}} p_{interp}(r) \tag{5.2}$$

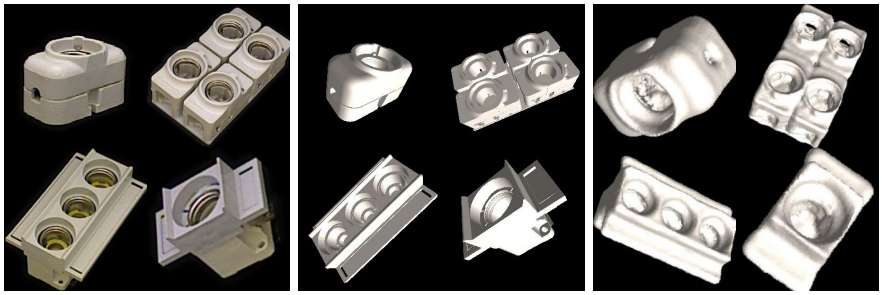
The mAP is then the mean AP over total classes.

5.2 CycleGAN Training Overview

We have two sets of input given to the two generators of the CycleGAN network. The input to the first generator is the rendered CAD model image of the objects. The CAD model of the objects are rendered in random position with augmentations using OpenGL framework. Along with the rendered images, information about the objects in each images such as the object id, bounding box values are also obtained. The input to the second generator is the real images dataset with additional augmentations.

5.2.1 T-LESS Dataset

T-LESS dataset consist of images taken from three image sensors and we use images obtained from Primesense CARMINE 1.09 for our experiments. There are 1296 RGB images in T-LESS dataset with objects centered in a black background. The dataset includes the real images, manually created 3D CAD models and semi-automatically reconstructed (RECONST) models of 30 industry relevant objects. The real images depicts single objects per scene covering a symmetrically sampled full view sphere at constant radius. We use objects with ids 5, 8, 9 and 10. The 4 T-LESS objects used for the evaluation in real, CAD nad RECONST domain are shown in Fig 5.1. To obtain images of objects in all possible 3D orientation each of the images in the dataset is rotated in-plane. These images comprises the target image dataset for CycleGAN. The corresponding 3D CAD models of



(a) Real images (b) CAD images (c) RECONST images

Figure 5.1: T-LESS objects used for evaluation. Clockwise from top, object 5, object 8, object 10 and object 9.

these objects are taken form the T-LESS dataset and then rendered using OpenGL in random positions. The CAD model images of the 4 T-LESS objects are also shown in Fig 5.1. These images constitute the input dataset for the CycleGAN.

5.2.2 YCB Dataset

YCB dataset comprises of 77 objects from 5 different categories. The dataset includes the real images, their corresponding segmentation masks and texture-mapped 3D mesh models. Two objects from the tool items category namely power drill and wood block and one from the kitchen items category, pitcher base, are used for the evaluation purpose. The real images in the dataset consist of single object placed in a table. The objects are cropped out from the scene and then pasted in a black background as shown in Fig 5.2(d). Further the images are rotated in plane to get all possible orientations. These images are used as the target dataset for our CycleGAN. The corresponding textured 3D mesh models of these objects are rendered using OpenGL without the texture information. These images are further used as input images for the CycleGAN. The real, textured and non-textured images from the YCB dataset are shown in Fig 5.2.

(d) Real images

(e) Textured 3D model images

(f) Non textured images

Figure 5.2: YCB objects used for evaluation. From left, power drill, wood block and pitcher base.

5.3 RetinaNet Object Detector

RetinaNet is a state of the art single stage object detection network and is proven to detect objects in a scene quickly and with high accuracy. Since researching on various object detectors is not a

part of this thesis, we use RetinaNet[1] object detection network for our experiments. We train real dataset, CAD rendered images and CycleGAN generated images using RetinaNet and then validate the performance in test dataset. The resulting performance of the detector with different domain images are then analyzed and compared. The hyperparameters used for training RetinaNet are mentioned in Appendix.

5.3.1 Training Dataset

T-LESS We train the RetinaNet on Primesense images, CAD rendered images, RECONST images, CycleGAN generated images and a combination of CAD rendered, RECONST and CycleGAN generated images. Please note that only the above mentioned 4 T-LESS objects are used in all these cases.

YCB We train the RetinanNet on real images, 3D textured 3D mesh model images, CycleGAN images and their combinations. Please note that the above mentioned 3 YCB objects are used for in all these cases.

5.3.2 Test Dataset

T-LESS For validation of the trained model, we use the T-LESS test dataset. The test dataset comprises of 20 scenes each containing 504 views of objects, with more than one object per scene. We use test-scene 11 which contains the objects 5, 8, 9 and 10.

YCB For validation of the YCB trained model we use images containing the above mentioned 3 YCB objects from the YCB video dataset.

5.4 CycleGAN Results

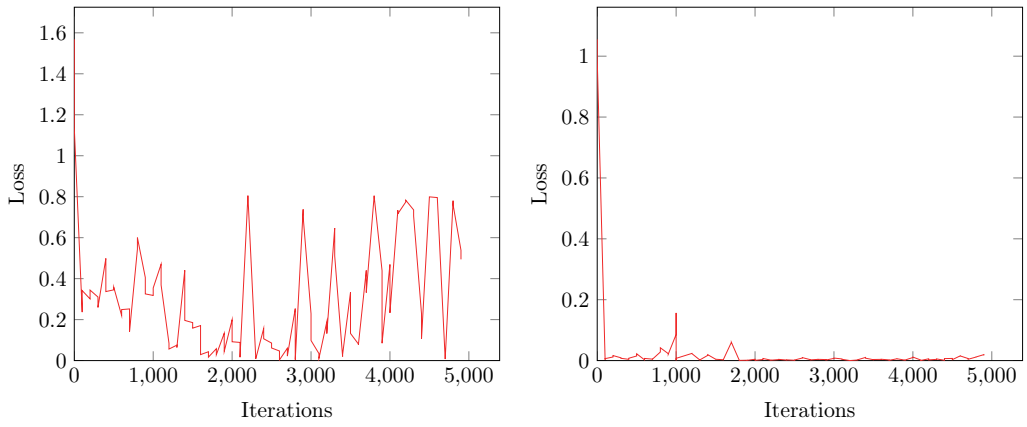
This section discuss the results from the CycleGAN architecture for T-LESS and YCB dataset. Although we use 4 objects in T-LESS and 3 objects in YCB dataset for evaluation, we discuss the results based on one object from both datasets. Object 5 from T-LESS dataset and Power drill from YCB dataset is chosen here. Since we do not have any parameter or metric in the CycleGAN network to analyze the results, in this section, we compare the results based on the loss values of the generator and the discriminator and the visual quality of the fake images and the reconstructed images produced by the Generator. Further in the next section these images are evaluated using an object detection network and the quality of the images are compared based on the mAP of the detector.

5.4.1 Vanilla CycleGAN

The basic CycleGAN architecture is used initially to evaluate the potential of the network to generate images from the T-LESS and YCB dataset. The Generator 1 and Discriminator 1 loss curves during the training process is shown in Fig 5.3. The Generator loss values were increasing and decreasing through out the training process with the maximum value of 0.8 whereas the

[1]The code is adapted from the Keras implementation of RetinaNet object detector at https://github.com/fizyr/keras-retinanet

Discriminator loss collapsed to 0 soon after the start of training. The entire network was trained for 5000 iterations only because of the early collapse of the discriminator. The hyper-parameters used are mentioned in the Appendix A.2.1. The generator stops learning further since the discriminator became strong enough to distinguish the real images from the dataset and the fake images produced from the generator. The images produced from the vanilla CycleGAN for T-LESS object 5 is shown in Fig 5.21a. The fake images produced from the generator failed to produce real looking images and moreover failed to even reproduce the structure and position of the input images. Similarly the results obtained for YCB dataset object power drill is shown in Fig 5.21b. As mentioned before we are trying to produce textured object images from texture-less rendered images for YCB objects. This is considered as an even more difficult problem than T-LESS objects because there exists a wide domain gap between the texture-less input and the expected textured output images. This can be clearly observed in the generated fake images. The generated images are nearly black for all the input images. An interesting observation that can be seen for T-LESS object is that regardless of the poor training performance, the network successfully reconstructed the input images. The reason for this behaviour may be the higher dominance of the cyclic loss through out the training, which in turn caused the generator to concentrate more on reconstructing the input image rather that producing the domain adapted fake images. However the same network failed to reproduce the reconstructed images for YCB object due to the large domain gap of the input and the target image. To conclude, the vanilla CycleGAN could not perform well in domain adaptation for both T-LESS and YCB objects.



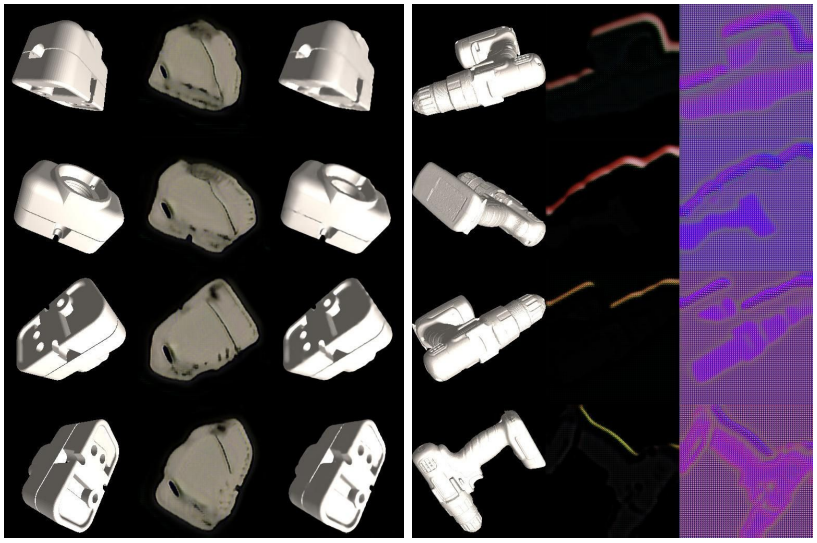
(a) Generator loss curve

(b) Discriminator loss curve

Figure 5.3: Loss curve during Vanilla CycleGAN training. The values represent the corresponding loss function values for each iterations.

5.4.2 Patch GAN Discriminator

The discriminator in the vanilla CycleGAN architecture collapsed in the first few iterations of the training process since the discriminator network was comparatively stronger than its counterpart generator network. In order to make the discriminator network less strong, we used a patch discriminator architecture. In the vanilla CycleGAN architecture we used a regular GAN



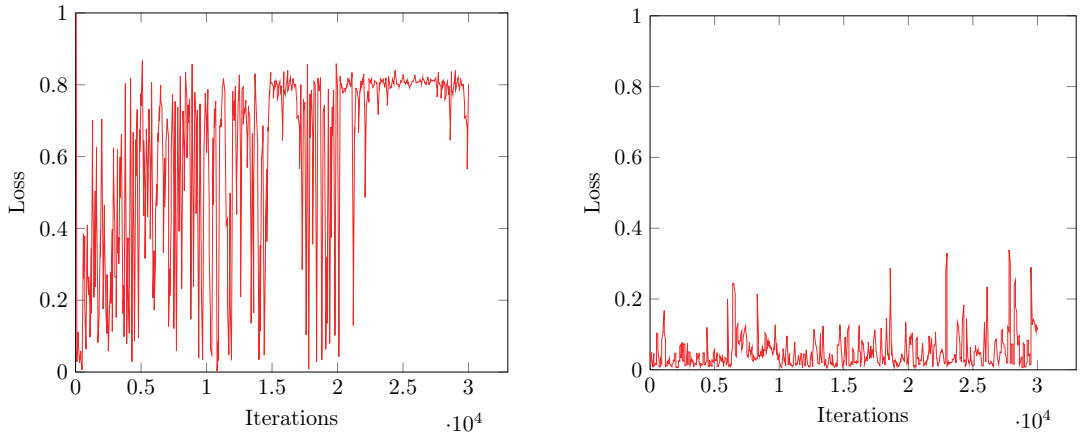
(a) T-LESS Object 5 (b) YCB power drill

Figure 5.4: Results obtained from Vanilla CycleGAN architecture. The rendered CAD models of the object used as input is shown in left column, the corresponding output from the generator is shown in the middle column and the reconstructed image is shown in the right column.

discriminator where an input image of size $N \times N$ is classified as 'fake' or 'real' and outputs a single scalar value 1 or 0. In patch discriminator the input image of size $N \times N$ is mapped to an output array of size $M \times M$ of outputs O, where each output O_{ij} indicates whether the patch ij in the image is 'real' or 'fake'. This is done by tracing back the receptive field of each output values O_{ij} to identify which pixels of the input image it is responsive to. We used a 70×70 patch GAN discriminator with the same hyperparameters as vanilla CycleGAN. This modified discriminator architecture is thus expected to have a high loss function value during the training. The generator and discriminator loss curves during the training is shown in Fig 5.5. From the loss curves it can be observed that the discriminator did not collapse during the training as before. However, the discriminator network got stronger than the generator network through out the training process, making it hard for the generator to produce real looking fake images. As a result, the generated images did not exhibit real image characteristics and are shown in Fig 5.6. For T-LESS and YCB objects the generated images are of poor quality with no defined corners and structure. The reconstructed images also exhibited poor quality. Please note that the cyclic loss coefficient was exponentially reduced (with a start value of 12) during the training in order to reduce the dominance of the cyclic loss as observed in the vanilla CycleGAN architecture. Although Patch GAN discriminator CycleGAN architecture surpassed the vanilla CycleGAN performance in terms of loss and image quality it could not exhibit satisfactory performance.

5.4.3 Training Generator more than Discriminator

Since the discriminator network got always stronger than the generator network in the previous experiments, a new method of training the network was adopted. In this experiment the generator



(a) Generator loss curve

(b) Discriminator loss curve

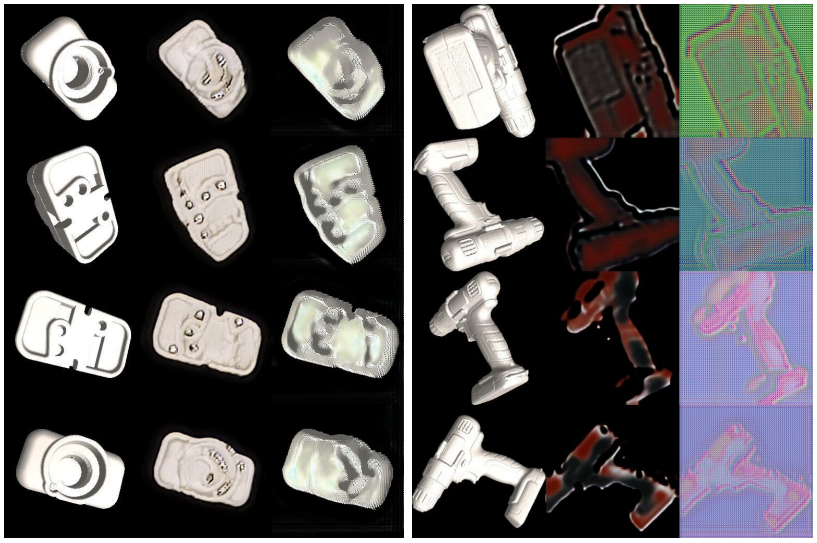
Figure 5.5: Loss curve of CycleGAN with patch discriminator. The values represent the corresponding loss function values for each iterations.

network is given a head start in the training process and trained more number of times than the discriminator network. All other hyperparameters of the network remained unchanged during training except that the generator network was trained 4 times more than the discriminator in each iteration. Since the exponential decay of the cyclic loss coefficient increased the quality of the images in the previous experiment the same condition was used here also. Since the network performance did not improve after few iterations an early stopping was done after 8000 iterations. It is an interesting point to note that the network managed to produce better results even with less number of iterations.

The loss curves of the generator and the discriminator during the training is shown in Fig 5.7. The discriminator loss values randomly increased and decreased within the range 0.1 and 0.4 whereas the generator loss values were in the range 0.1 and 0.8. Although a balance could not be achieved between the two networks a considerable better performance was observed in this experiment in terms of loss function values. Also during the last few iterations of the training both networks loss values reached around 0.4. The analysis of the loss curves shows that, the discriminator network neither collapsed nor became effectively stronger than the generator network. This improved performance can be clearly seen in the quality of the images produced as well and are depicted in Fig 5.8. For T-LESS object, a reasonably good domain adapted images and reconstructed images were generated when compared to our previous results. For YCB power drill, the results got improved from the previous ones but the network could not produce domain adapted images well indicating that for image pairs with large domain gap more modifications are required.

5.4.4 Addition of Identity Loss

In this experiment, an additional loss term called as identity loss was added to the generator loss function which is already defined in the previous chapter. Basically the intuition behind this auxiliary loss term is to preserve the colour composition between input and the output images. The



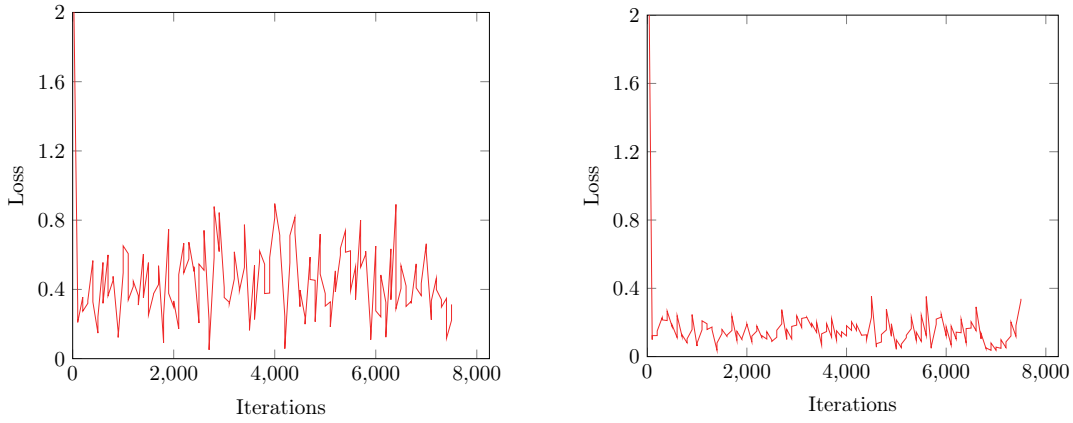
(a) T-LESS Object 5 (b) YCB Power drill

Figure 5.6: Results obtained when patch discriminator architecture is used. The rendered CAD models of the object used as input is shown in left column, the corresponding output from the generator is shown in the middle column and the reconstructed image is shown in the right column.

identity loss term regularize the generator network to be near an identity mapping when the target images are given as input to the network. This also implies that if a particular image already looks similar to the target image, it should not be mapped further into a different image. We used the half value of the cycle loss coefficient parameter as the identity loss co-efficient parameter value. The other hyperparameters used for the training remained unchanged. The resultant loss curves of the generator and discriminator is shown in Fig 5.9. The generator loss curve was seen increasing and decreasing randomly with each iterations. The discriminator loss curve gradually decreased with each iterations. The generated and the reconstructed images of the T-LESS and YCB objects are shown in Fig 5.10. For T-LESS objects, the generated images showed better visual quality than the previous results. For YCB power drill, the results improved for some object poses whereas for image rows 3 and 4 the generated images did not produce domain adapted images properly. It can be observed that some sections of the generated image adapted the colour information of the target image.

5.4.5 Dropout in Discriminator

The term 'Dropout' [54] in deep learning refers to the method of ignoring few units or neurons in a neural network during the training time. This is a widely used regularization technique used to prevent the over-fitting of the network model. Dropout technique forces the neural network to learn more general and robust features with different sets of neurons being activated at each iteration step during the training process. This method is preferred based on the fact that sometimes learning less can make the network learn better to generalize. The effect of adding dropout in a standard



(a) Generator loss curve

(b) Discriminator loss function

Figure 5.7: Loss curves of CycleGAN when generator network is trained more number of times than discriminator. The values represent the corresponding loss function values for each iterations.

fully connected neural network is shown in Fig 5.11. As shown in the figure, the dropout can be added in the hidden layers of the neural network by ignoring random p number of neurons. In this experiment, Dropout is added to discriminator network in order to penalize the learning of the discriminator. During the training phase dropout is added to the hidden layers in the discriminator architecture with dropout value of 0.5. This implies for each iteration during the training phase, half number of neurons along with their activation functions in the hidden layers of the network are randomly chosen and ignored. The loss curves of the generator and the discriminator during the training are shown in Fig 5.12. It can be observed that the discriminator loss curve did not go below 0.2 because of the presence of dropout layers in the network architecture. However the generator loss curves followed a similar kind of behaviour as in the previous experiment but with less number of fluctuations from the peak value 0.8. The generated and the reconstructed images for this network model is shown in Fig 5.11. For T-LESS objects, the generator produced a good quality domain adapted images but the images were accompanied with random noises around the object. For YCB power drill, the generator could not produce a good quality domain adapted image. The generated images were all dark red in colour with no other textural details. Addition of dropout layers made the YCB results comparatively poor than the previous experiments and created random noise like appearance around the T-LESS object. Even though the dropout technique improved the discriminator performance during the training, the generator network could not perform well to produce domain adapted images.

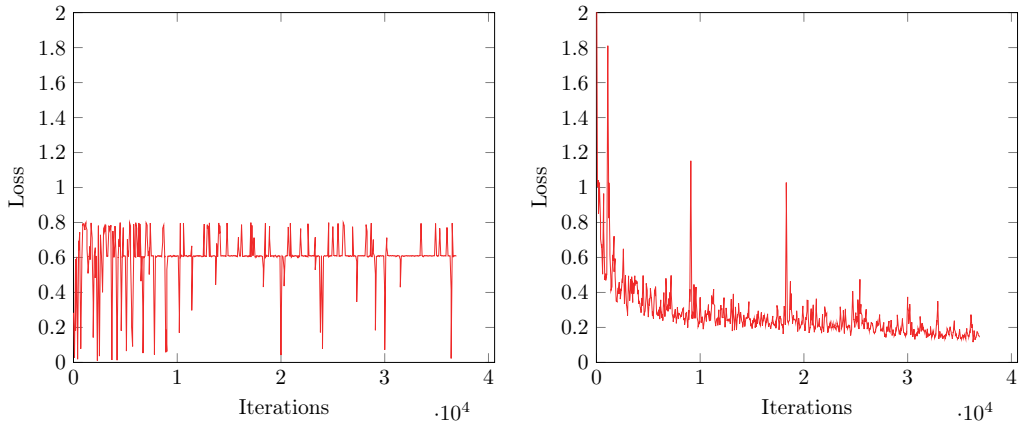
5.4.6 Addition of Mask Loss

The last method used is the usage of Mask loss, an additional loss function used along with the generator loss function of our CycleGAN architecture. The major intuition behind the usage of this additional loss term was to preserve the generated image position same as that of the synthetic input image. It was noted from the close observation of some generated images of previous experiments

(a) T-LESS Object 5 (b) YCB Power drill

Figure 5.8: Results obtained when generator network is trained more number of times than discriminator. The rendered CAD models of the object used as input is shown in left column, the corresponding output from the generator is shown in the middle column and the reconstructed image is shown in the right column.

that the position of the images showed deviation from the input image pose. An example image for T-LESS object is shown in Fig 5.14. In order to preserve the object position in both domains we used another loss function that calculated pixel wise difference between the mask of the input synthetic image and the generated domain adapted image. A detailed explanation of this loss term is already given in the previous chapter. However this additional term slows down the training process and in due time there are high chances for this to dominate the domain adaptation loss. In order to avoid this issue an exponential decay is performed on the mask loss term during the training process. By adopting this technique it is estimated that the network would have an initial knowledge on the pose information during the early steps of training. An exponential decay factor of 1e-3 is used and after few iterations the network is trained with no mask loss term. Please note that dropout layers were not used during training in this section since it did not perform well on YCB object. The loss curves obtained during this training are shown in Fig 5.15. The generator network showed larger fluctuations through out the training whereas the discriminator loss values centered around 0.1 and 0.35. The generated images are shown in Fig 5.16. The network produced comparatively good visual quality domain adapted images in the same position as input images for both T-LESS object and YCB power drill. The reconstructed images also exhibited good and sharp features.
Hence our modified CycleGAN architecture for domain adaptation incorporates the following methods: usage of patch discriminator, training generator more than discriminator, addition of identity loss and mask loss. The hyperparameters used for the training is mentioned in Appendix A.2.2. The domain adapted images obtained from the modified CycleGAN architecture for T-LESS object 8 is shown in Fig 5.17, object 9 is shown in Fig 5.18 and for object 10 in Fig 5.19.



(a) Generator loss curve

(b) Discriminator loss curve

Figure 5.9: Loss curves of CycleGAN when identity loss function is added to the generator loss function. The values represent the corresponding loss function values for each iterations.

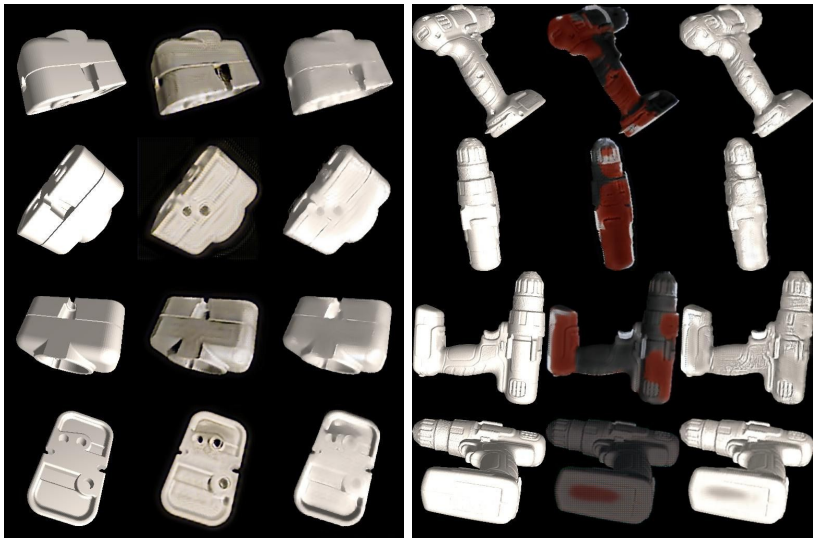
Similarly the domain adapted image results for YCB wood block is shown in Fig 5.20 and for YCB pitcher base is shown in Fig 5.21. In all of the above mentioned methods used for improving the performance of CycleGAN, we discussed the quality of our results based on the loss curves during training and by analyzing the visual appearance of the generated images. An important observation is that the analysis of the loss curves of generator and discriminator did not provide much intuition on the overall performance. The loss curves did not reach the theoretical value of 0.5 in these experiments. It should be noted that even when the images generated from the CycleGAN architecture were visually appealing, the loss curves did not balance well.

5.5 Object Detection Results

In the previous section we evaluated the quality of the GAN generated images in terms of visual quality. The main goal of this thesis is to produce realistic looking images from GAN so that we can use them in place of real images for applications like object detection, pose estimation etc. In order to evaluate the possibility of using the GAN images for deep learning applications we train the RetinaNet object detector with these images and analyze the accuracy of the trained detection model.

5.5.1 Evaluation on T-LESS Dataset

In order to compare the performance of various domain images on the object detection we carried out experiments using these images. The number of input images during the training was fixed to 40,000. For training phase multiple objects of the respective domain were pasted in random images from PASCAL VOC dataset. A sample of the training images used in each domain is shown in Fig 5.22. The hyperparameters used for the training of RetinaNet detector is mentioned in Appendix A.2.3. Initially we trained the detector using real images, GAN images, CAD images and RECONST images as input images. The detector was also trained on combinations of real and

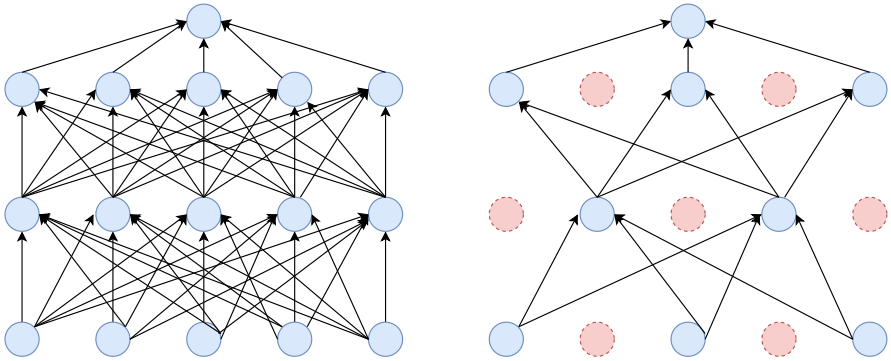


(a) T-LESS Object 5 (b) YCB Power drill

Figure 5.10: Results obtained when identity loss function is added to the generator loss function of the network. The rendered CAD models of the object used as input is shown in left column, the corresponding output from the generator is shown in the middle column and the reconstructed image is shown in the right column.

GAN images, real and CAD images, real and RECONST images. In these combinations 20,000 real images were only used. The trained model was then tested in the test dataset scene 11 of T-LESS dataset. The test results of the model trained with GAN images are shown in Appendix A.3. The mAP obtained for each of the 4 T-LESS objects in the test dataset is given in Table 5.1. It can be inferred from the results that the detector network trained with GAN images alone could not surpass the performance of the network model trained with real images alone. However when we combined 20k real images and 20k GAN images the detector network performed well with an average mAP of 0.9054. Please note that the detector network trained with half number of real images and GAN images performed similar to the network trained with 40k real images alone. The detector network trained using CAD images failed to detect objects in the test dataset whereas the RECONST images exhibited an average performance on the test dataset with an average mAP of 0.2322. The combination of real and RECONST images performed better than RECONST images alone similarly the network trained with the combination of real and CAD images performed significantly well than the network model trained with CAD alone.

Although the detector network model could achieve better performance with the combination of GAN and real images it is important to evaluate the dominance of the real image features in these models. In order to evaluate the dominance of the real images in the network model another set of experiments were conducted. This time the detector network was trained on 1000 real images alone, then we combined 38000 GAN images and trained the network again. Similarly the network was trained on 10,000 real images alone and then with a combination of 10,000 real and 30,000 GAN images followed by 15,000 real images alone and then a combination of 15,000 real and 25,000

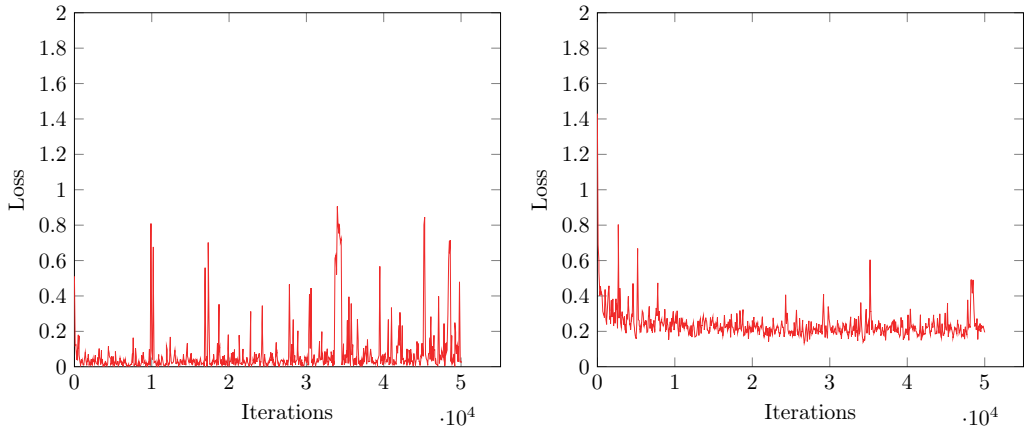


(a) Standard neural network (b) Neural network after dropout

Figure 5.11: Effect of dropout layers in a standard neural network.

GAN images. The mAP values obtained are tabulated in Table 5.2. Addition of 39,000 GAN images improved the average mAP of the detector trained with 1000 real images from 0.7381 to 0.8216. Correspondingly, a significant increase in the performance was observed when 30,000 GAN images were combined with 10,000 real images. The effectiveness of our CycleGAN generated images can be explained with the help of mAP curve as shown in Fig 5.23. The figure shows the mAP values obtained for the various combinations of real and CycleGAN generated images. Please note that the number of input images was fixed to 40k in all these combinations. As shown in the graph, when the detector network was trained with only GAN images that is with 0% of real images the mAP obtained was 0.42. Gradually we increased the amount of real images used as input to the detector network. When 5% real data was added to the detector, the mAP value increased to 0.82. When the real and GAN images were mixed in equal amounts (50%) the mAP of the trained network reached 0.90. When the network was trained with 100% real data the detector network reached mAP value of 0.89. An important inference from this curve is that, if we replace half of the real data with GAN data and train the detector network, we could achieve similar performance as the model trained with 100% real data.

It is also important to compare the individual mAP of the 4 GAN generated T-LESS objects. However, we trained the same CycleGAN architecture for all the objects, the performance of these objects during object detection varied largely. Different set of hyperparameters were used for each objects during the training of RetinaNet architecture. Considering the results obtained for the object detection model trained with 40k GAN images (Table 5.1), Object 5 performed well than the other 3 objects with mAP of 0.7038. The other 3 T-LESS objects could not achieve mAP value greater than 0.5. In order to understand the reason behind the different mAP values of T-LESS objects, we compared the GAN generated images and the real images of the same. The real and GAN generated images of the 4 T-LESS objects used is given in Fig 5.24. The GAN generated image of object 5 resembled well with its real domain image showing that our CycleGAN architecture could successfully produce domain adapted image for object 5. However for objects 8, 9 and 10 the GAN generated images did not resemble well with their respective real domain images. The GAN image of object 10 differed substantially from its real domain image which accounts for its lowest mAP value of 0.3148.



(a) Generator loss curve

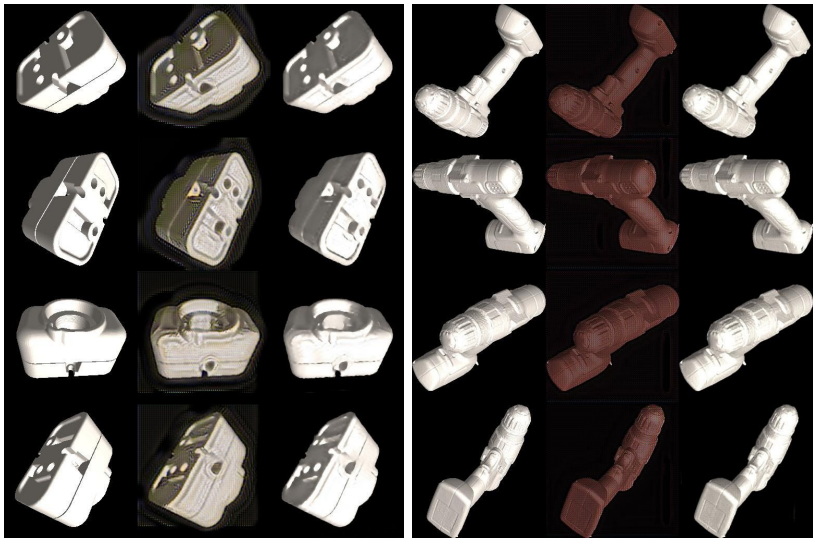
(b) Discriminator loss curve

Figure 5.12: Loss curves of CycleGAN when dropout layers are added to discriminator network. The values represent the corresponding loss function values for each iterations.

5.5.2 Evaluation on YCB Dataset

Similar to the experiments conducted with T-LESS dataset, we evaluated the performance of the YCB generated GAN images by training the RetinaNet object detector. We used 40k images with multiple objects in a single image with random background from PASCAL VOC dataset as input during training. The detector network was trained with GAN images, real images, textured 3D model images and their combinations. The sample input images used for these different domains are given in Fig 5.25. After training, the model was evaluated on test images extracted from the YCB video dataset. The test results of the model trained with GAN images are shown in Appendix A.3. The mAP obtained for the 3 YCB objects in the test dataset is shown in Table 5.3. The real images trained model dominated significantly well in performance than models trained with other domain images with mAP of 0.8994. The performance of the GAN images trained model was comparatively low than the real images with mAP of 0.3742. However GAN images surpassed the performance of the textured 3D images trained model. The combination of real and GAN images did not exceed the performance of the model trained with real images alone. Thus our CycleGAN generated images could surpass the performance of the textured 3D model trained images while it failed to exceed the real domain performance.

When considering the individual performance of the GAN generated images on object detection, YCB wood block performed significantly well than the other two YCB objects with mAP value 0.7010. The YCB power drill and pitcher base could not achieve mAP of 0.5 during testing. The real and GAN images of the three YCB objects used are given in Fig 5.26. For power drill, although the CycleGAN managed to generate texture similar to its real domain image, some of the details were wrongly produced. It can be seen from the image that power drill GAN image could not generate the black colour in the handle and instead produced black base. Wood block GAN image resembled with its real domain image comparatively well. With closer observation it can be seen that the fine line in the real images are not well produced in the GAN images. Pitcher base



(a) T-LESS Object 5 (b) YCB Power drill

Figure 5.13: Results obtained when dropout layers are added to the discriminator architecture of the network. The rendered CAD models of the object used as input is shown in left column, the corresponding output from the generator is shown in the middle column and the reconstructed image is shown in the right column.

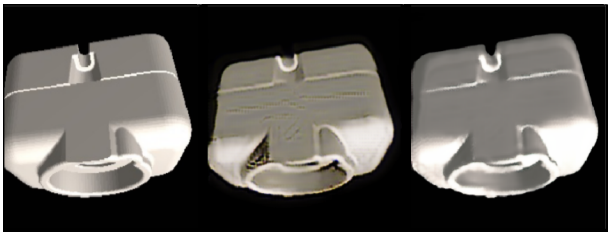
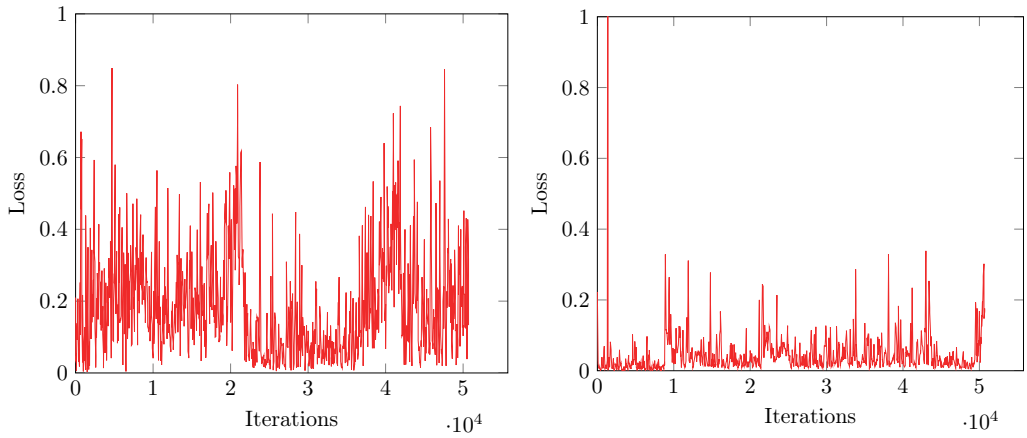


Figure 5.14: An example image of T-LESS object 5 with generated image position deviated from the input image.

GAN image could only resemble the colour from its real domain image. The label information was not at all generated in the GAN image. The above observations conclude that our CycleGAN could generate domain adapted images well for only objects with lesser complexity. For images with rich and fine texture and colour information like pitcher base,the GAN images failed to regenerate those features. This downside of our GAN images justifies the lower mAP values obtained for the power drill and pitcher base during object detection.



(a) Generator loss curve

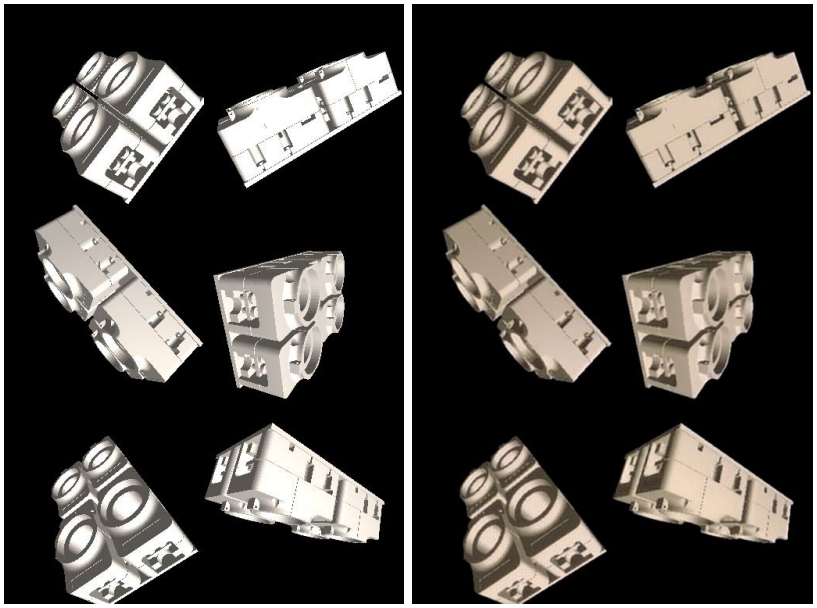
(b) Discriminator loss curve

Figure 5.15: Loss curves of CycleGAN when mask loss function is added to the generator loss function.The values represent the corresponding loss function values for each iterations.

(a) T-LESS Object 5

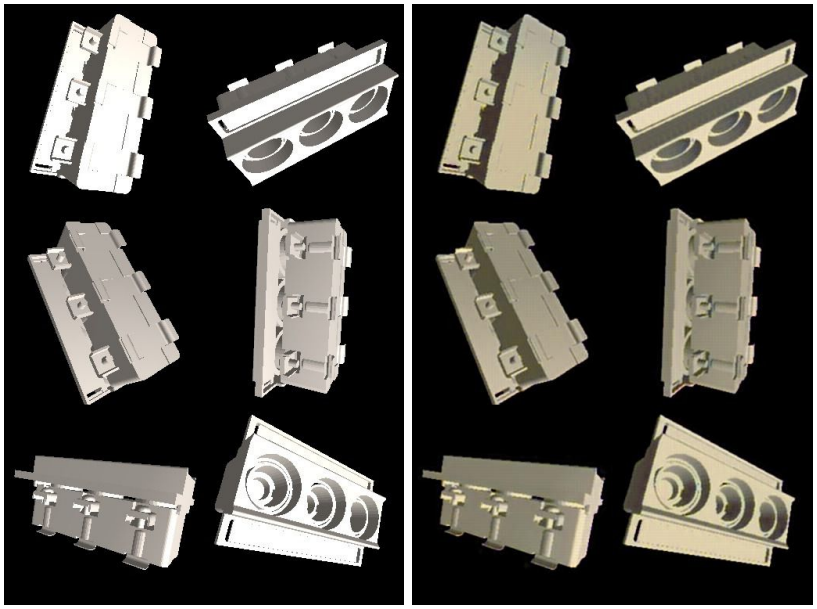
(b) YCB Power drill

Figure 5.16: Results obtained when mask loss function is added to the generator loss function of the network. The rendered CAD models of the object used as input is shown in left column, the corresponding output from the generator is shown in the middle column and the reconstructed image is shown in the right column.



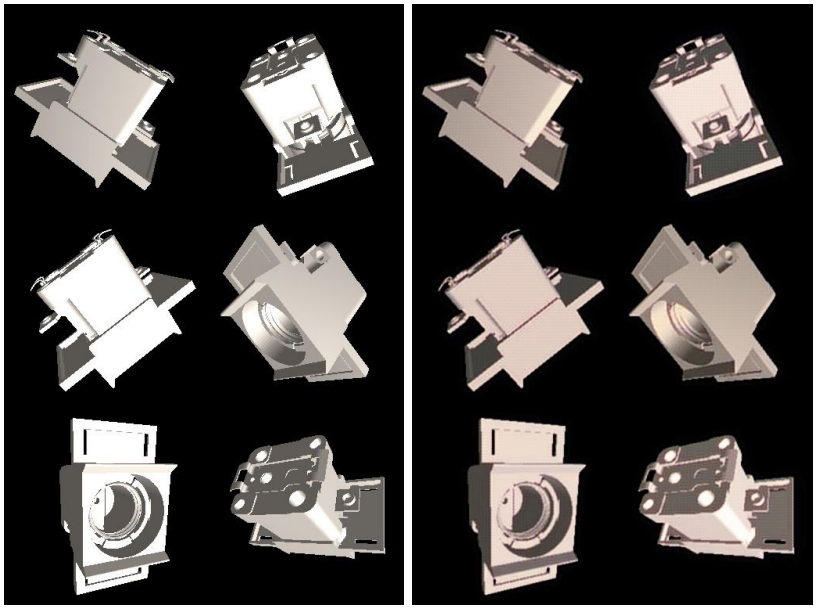
(a) Input images (b) Output images

Figure 5.17: Results obtained for T-LESS object 8 using our CycleGAN architecture.



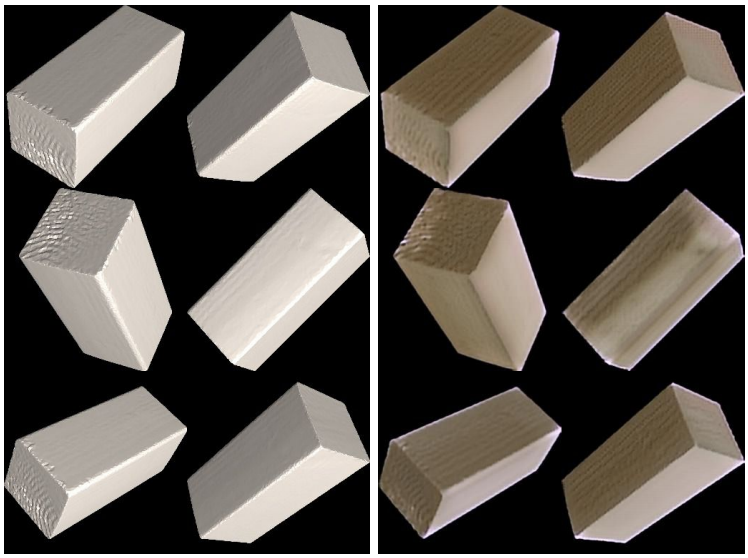
(a) Input images (b) Output images

Figure 5.18: Results obtained for T-LESS object 9 using our CycleGAN architecture.



(a) Input images (b) Output images

Figure 5.19: Results obtained for T-LESS object 10 using our CycleGAN architecture.



(a) Input images (b) Output images

Figure 5.20: Results obtained for YCB object wood block using our CycleGAN architecture.

(a) Input images (b) Output images

Figure 5.21: Results obtained for YCB object pitcher base using our CycleGAN architecture.

Input Images	Object 5 (mAP)	Object 8 (mAP)	Object 9 (mAP)	Object 10 (mAP)	Average (mAP)
40k Real	0.8700	0.9772	0.8693	0.8564	0.8932
40k GAN	0.7038	0.4598	0.4343	0.3148	0.4793
40k CAD	0.0732	0.1111	0.0880	0.0179	0.0725
40k RECONST	0.2350	0.3987	0.1482	0.1470	0.2322
20k Real + 20k GAN	0.8731	0.9739	0.9183	0.8560	0.9054
20k Real + 20k CAD	0.6753	0.8623	0.6627	0.6017	0.7005
20k Real + 20k RECONST	0.7129	0.9056	0.7129	0.7083	0.7599

Table 5.1: Object detection results on objects 5,8,9 and 10 of T-LESS dataset. The mAP values obtained for various domains of input images using RetinaNet detector is tabulated. Higher values of mAP corresponds to better performance.

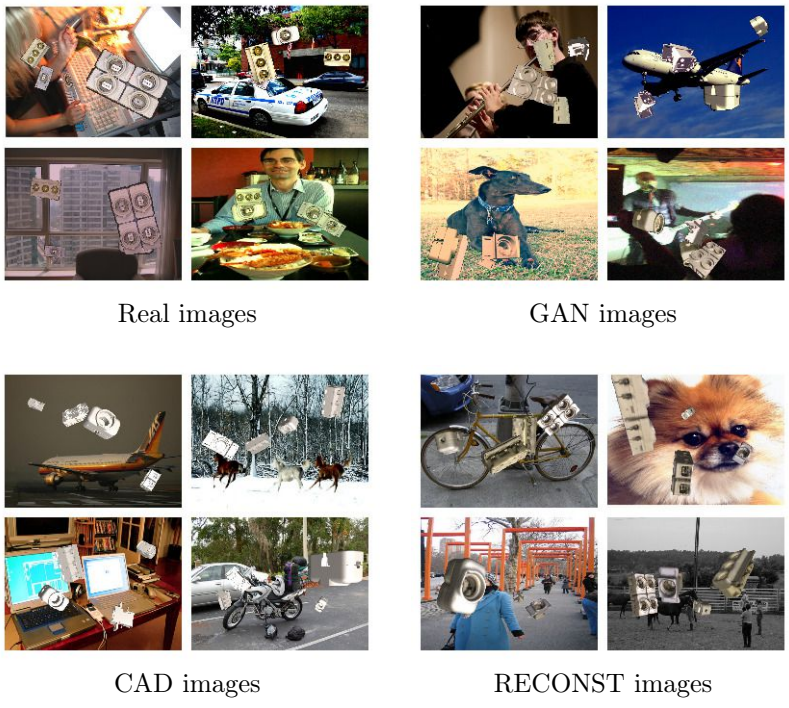


Figure 5.22: Sample T-LESS input images in various domains used for training the RetinaNet object detector.

Input Images	Object 5 (mAP)	Object 8 (mAP)	Object 9 (mAP)	Object 10 (mAP)	Average (mAP)
1k Real	0.7073	0.8571	0.7421	0.6459	0.7381
1k Real + 39k GAN	0.7824	0.9623	0.8210	0.7210	0.8216
10k Real	0.7923	0.8632	0.7820	0.7123	0.7875
10k Real + 30k GAN	0.8230	0.9730	0.8480	0.7880	0.8580
15k Real	0.8220	0.8710	0.8005	0.7489	0.8106
15k Real + 25k GAN	0.8424	0.9735	0.8732	0.8132	0.8755

Table 5.2: Object detection results on objects 5,8,9 and 10 of T-LESS dataset. The mAP values obtained for various combinations of real and GAN images using RetinaNet detector is tabulated. Higher values of mAP corresponds to better performance.

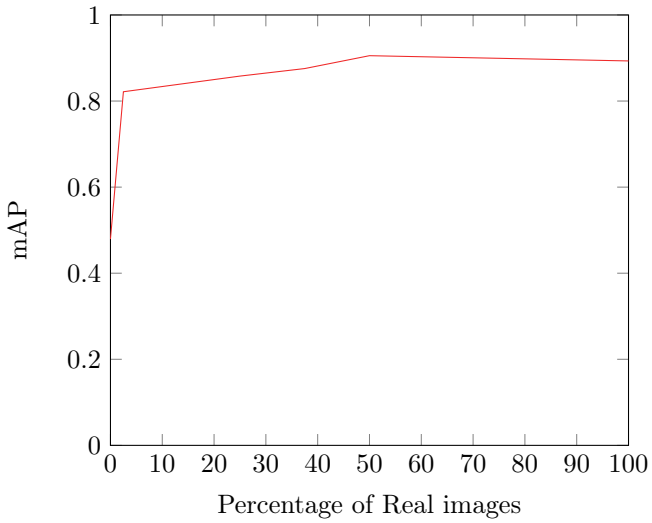


Figure 5.23: Object detection results on different combinations of real and GAN images from table 5.1 and 5.2. The number of real images used for training the detector is expressed in percentage. The mAP values obtained for each percentage of input real images are plotted.

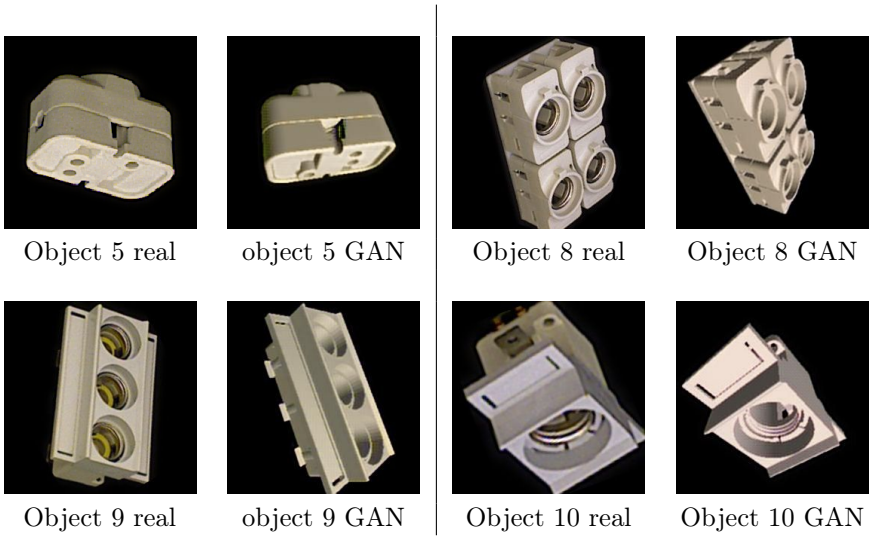


Figure 5.24: Comparison of real and GAN generated images of T-LESS objects.

Input Images	Power drill(mAP)	Wood block (mAP)	Pitcher base (mAP)	Average (mAP)
40k Real	0.9010	0.8963	0.9009	0.8994
40k GAN	0.3010	0.7010	0.1208	0.3742
40k Textured 3D model	0.3783	0.1059	0.0104	0.1648
20k Real + 20k GAN	0.6349	0.6466	0.6051	0.6289
20k Real + 20k Textured 3D model	0.7993	0.3333	0.1208	0.4178

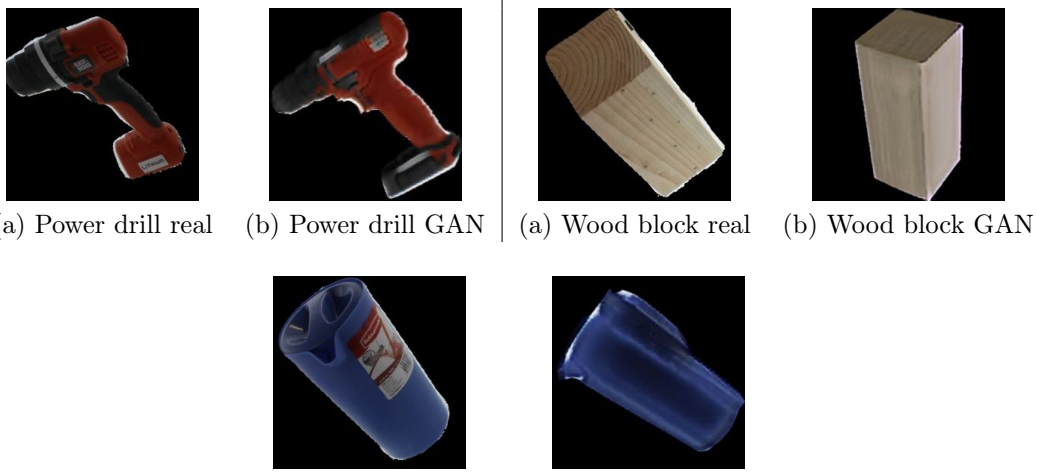
Table 5.3: Object detection results on objects power drill, wood block and pitcher base of YCB dataset. The mAP values obtained for various combinations of input images using RetinaNet detector is tabulated. Higher values of mAP corresponds to better performance.

(a) Real images

(b) CycleGAN images

(c) Textured 3D model images

Figure 5.25: Sample YCB input images from various domains used for training the RetinaNet object detector.



(a) Power drill real (b) Power drill GAN (a) Wood block real (b) Wood block GAN

(a) Pitcher base real (b) Pitcher base GAN

Figure 5.26: Comparison of real and GAN generated images of YCB objects.

Chapter 6

Conclusion

The potentiality of generating realistic images using GANs was investigated in this thesis. Different variants of GANs were researched in the beginning and CycleGAN architecture was chosen since the network worked with the unpaired dataset. The generated image quality was first evaluated based on visual inspection and the loss curves of the generator and the discriminator during training. Initially, Vanilla CycleGAN was trained to check the possibility of the network to produce domain adapted images from simulation images. As the vanilla network failed to generate good quality images, we modified the architecture by utilizing a patch GAN discriminator instead of the normal discriminator architecture to stabilize the loss imbalance between discriminator and generator networks. Although this improved the results, the discriminator network was seen dominating throughout the training resulting in the generator network to stop learning further. The generator network was trained more than the discriminator to resolve this limitation. An additional loss term called identity loss was added to the generator network to regularize the generator and another loss term, mask loss was added to preserve the position of the objects in the generated images. Thus we modified the CycleGAN architecture with techniques to stabilize the network and to produce realistic domain adapted images from the simulated input images.

Two datasets were used for carrying out the experiments. A simple texture-less object dataset called T-LESS and a complex dataset termed YCB. For T-LESS objects, the objective was to produce real domain images from rendered CAD images of the objects and for YCB objects the goal was to generate textured objects from texture-less rendered images. In order to evaluate the quality of our generated images, we used RetinaNet object detector. Four objects from T-LESS dataset and three objects from YCB dataset were used for the evaluation. Images from real, GAN and simulation domain were trained and their corresponding mAP values were tabulated and compared for examining the performance and quality. For T-LESS objects, object detector trained with a combination of of 50% of real and 50% GAN domain images succeeded in achieving similar accuracy as the detector trained with 100% real images alone. However, for YCB objects, where there existed a large domain gap between the source and target domain images, the generated images were not able to adapt the domain gap efficiently. For a simple object with less domain gap, notable performance was observed.

To summarize, our modified CycleGAN architecture produced realistic looking images from

simulated data with less complex domain gaps. We could effectively train an object detector with a combination of GAN images and real images for T-LESS objects. For complex domain gaps, further research is essential.

6.1 Future Scope

This thesis presents a potential future research possibility of using domain adapted images for object detection and pose estimation applications. One of the most important future works would be to evaluate the possibility of producing domain adapted images on more objects. We evaluated the results on 4 T-LESS and 3 YCB objects only. T-LESS contains 30 objects in its dataset and YCB contains a total of 77 objects in different categories. It would be really beneficial to explore how our modified network performs in different objects. We have already seen in our evaluation that some objects performed really well and some did not in domain adaptation. Evaluating more objects would help us in a better understanding of this behavior of our network and would aid in improving the current network.

While evaluating YCB objects it was observed that objects with large domain gap between the source and target domain images could not perform well as compared to the objects with less domain gap. Also, objects with complex and fine texture like YCB pitcher base could not perform well in our experiments. The reason for this limitation of our model would be the network structure. More study would be needed to thoroughly examine this behavior. We did not alter the network structure much in this thesis and hence adapting the network architecture to handle complex domain gaps would be a potential future work. This includes addition or removal of convolutional layers or adding skip connections to the network.

Improving the training stability of GANs has been always a potential research area. Although the research community has come up with many techniques to stabilize the training of GANs, choosing suitable techniques for each application is still a tedious task. A technique that works for one application might fail to perform in another different task. Despite using some techniques to improve the training of GANs, we could not totally stabilize the training of GANs. Exploring newer methods for enhancing training stability is another promising future work to focus on improving the current performance of our network.

It was also observed that T-LESS objects with appealing visual quality failed to perform as expected in object detection. Neural networks are usually criticized as "black box" since its unknown how the hidden layers perform and make decisions internally. A recent method proposed by OpenAI [55] introduces a technique called activation atlas to internally visualize the features generated by each hidden layers in a neural network. With more understanding of what is happening inside a neural network, it might be possible to identify where the network is depending on spurious correlation to identify and classify images. This would also help in understanding when the networks do not work as expected. Analyzing the behavior of the object detector using activation atlas when trained with the T-LESS objects would help us in understanding the above mentioned unanticipated behaviour.

We explored variants of GANs in generating realistic images from the synthetic domain in

this work. Another popular generative model, Variational Autoencoder [9] is also an area to investigate for future research. It basically comprises of an encoder-decoder architecture and has successfully able to generate high-quality images. Despite domain adaptation, another technique called domain randomization could be also used to augment the dataset used for training.

Bibliography

[1] C. Ott, O. Eiberger, W. Friedl, B. Bauml, U. Hillenbrand, C. Borst, A. Albu-Schaffer, B. Brunner, H. Hirschmuller, S. Kielhofer, R. Konietschke, M. Suppa, T. Wimbock, F. Zacharias, and G. Hirzinger. A humanoid two-arm system for dexterous manipulation. In *2006 6th IEEE-RAS International Conference on Humanoid Robots*, pages 276–283, Dec 2006.

[2] Andreas Domel, Simon Kriegel, Michael Kaecker, Manuel Brucker, Tim Bodenmuller, and Michael Suppa. Toward fully autonomous mobile manipulation for industrial environments. *International Journal of Advanced Robotic Systems*, 14(4):1729881417718588, 2017.

[3] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[4] Guilin Liu, Fitsum A. Reda, Kevin J. Shih, Ting-Chun Wang, Andrew Tao, and Bryan Catanzaro. Image inpainting for irregular holes using partial convolutions. *CoRR*, abs/1804.07723, 2018.

[5] Yuxi Li. Deep reinforcement learning: An overview. *CoRR*, abs/1701.07274, 2017.

[6] A. Gelbukh. Natural language processing. In *Fifth International Conference on Hybrid Intelligent Systems (HIS'05)*, pages 1 pp.–, Nov 2005.

[7] Generative Models. Accessed on august 2018. URL:`https://blog.openai.com/generative-models/`.

[8] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, 2014.

[9] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2014.

[10] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *ICML*, 2016.

[11] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016.

[12] Tomáš Hodaň, Pavel Haluza, Štěpán Obdržálek, Jiří Matas, Manolis Lourakis, and Xenophon Zabulis. T-LESS: An RGB-D dataset for 6D pose estimation of texture-less objects. *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2017.

[13] Berk Çalli, Arjun Singh, Aaron Walsman, Siddhartha S. Srinivasa, Pieter Abbeel, and Aaron M. Dollar. The ycb object and model set: Towards common benchmarks for manipulation research. *2015 International Conference on Advanced Robotics (ICAR)*, pages 510–517, 2015.

[14] Ramakant Nevatia and Thomas O. Binford. Description and recognition of curved objects. *Artif. Intell.*, 8(1):77–98, February 1977.

[15] Michael Stark, Michael Goesele, and Bernt Schiele. Back to the future: Learning shape models from 3d cad data. In *Proceedings of the British Machine Vision Conference*, pages 106.1–106.11. BMVA Press, 2010. doi:10.5244/C.24.106.

[16] J. Liebelt and C. Schmid. Multi-view object class detection with a 3d geometric model. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1688–1695, June 2010.

[17] Debidatta Dwibedi, Ishan Misra, and Martial Hebert. Cut, paste and learn: Surprisingly easy synthesis for instance detection. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 1310–1319, 2017.

[18] Hao Su, Charles Ruizhongtai Qi, Yangyan Li, and Leonidas J. Guibas. Render for cnn: Viewpoint estimation in images using cnns trained with rendered 3d model views. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2686–2694, 2015.

[19] Yu Xiang, Roozbeh Mottaghi, and Silvio Savarese. Beyond pascal: A benchmark for 3d object detection in the wild. 2014.

[20] Xingchao Peng, Baochen Sun, Karim Ali, and Kate Saenko. Learning deep object detectors from 3d models. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, pages 1278–1286, Washington, DC, USA, 2015. IEEE Computer Society.

[21] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

[22] Georgios Georgakis, Arsalan Mousavian, Alexander C. Berg, and Jana Kosecka. Synthesizing training data for object detection in indoor scenes. *CoRR*, abs/1702.07836, 2017.

[23] Stefan Hinterstoisser, Vincent Lepetit, Paul Wohlhart, and Kurt Konolige. On pre-trained image features and synthetic images for deep learning. *CoRR*, abs/1710.10710, 2017.

[24] Michael Mathieu, Camille Couprie, and Yann Lecun. Deep multi-scale video prediction beyond mean square error. 11 2015.

[25] Emily L Denton, Soumith Chintala, arthur szlam, and Rob Fergus. Deep generative image models using a laplacian pyramid of adversarial networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 1486–1494. Curran Associates, Inc., 2015.

[26] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *CoRR*, abs/1411.1784, 2014.

[27] Guim Perarnau, Joost van de Weijer, Bogdan Raducanu, and Jose M. Álvarez. Invertible conditional gans for image editing. *CoRR*, abs/1611.06355, 2016.

[28] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015.

[29] Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle, and Ole Winther. Autoencoding beyond pixels using a learned similarity metric. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 1558–1566. JMLR.org, 2016.

[30] Paul Viola and Michael J. Jones. Robust real-time face detection. *Int. J. Comput. Vision*, 57(2):137–154, May 2004.

[31] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1, June 2005.

[32] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann Lecun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *http://arxiv.org/abs/1312.6229*.

[33] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '14, pages 580–587, Washington, DC, USA, 2014. IEEE Computer Society.

[34] Ross Girshick. Fast r-cnn. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, pages 1440–1448, Washington, DC, USA, 2015. IEEE Computer Society.

[35] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, pages 91–99, Cambridge, MA, USA, 2015. MIT Press.

[36] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, June 2016.

[37] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. In *NIPS*, 2016.

[38] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. In *ECCV*, 2016.

[39] T. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2999–3007, Oct 2017.

[40] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2015.

[41] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2999–3007, 2017.

[42] L. J. Ratliff, S. A. Burden, and S. S. Sastry. Characterization and computation of local nash equilibria in continuous games. In *2013 51st Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 917–924, Oct 2013.

[43] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[44] Ming-Yu Liu and Oncel Tuzel. Coupled generative adversarial networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 469–477. Curran Associates, Inc., 2016.

[45] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5967–5976, 2017.

[46] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham, 2015. Springer International Publishing.

[47] Ming-Yu Liu, Thomas Breuel, and Jan Kautz. Unsupervised image-to-image translation networks. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 700–708. Curran Associates, Inc., 2017.

[48] Zili Yi, Hao Zhang, Ping Tan, and Minglun Gong. Dualgan: Unsupervised dual learning for image-to-image translation. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2868–2876, 2017.

[49] Taeksoo Kim, Moonsu Cha, Hyunsoo Kim, Jung Kwon Lee, and Jiwon Kim. Learning to discover cross-domain relations with generative adversarial networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1857–1865, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[50] Yaniv Taigman, Adam Polyak, and Lior Wolf. Unsupervised cross-domain image generation. *CoRR*, abs/1611.02200, 2016.

[51] Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Josh Susskind, Wenda Wang, and Russell Webb. Learning from simulated and unsupervised images through adversarial training. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2242–2251, 2017.

[52] Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *CoRR*, abs/1607.08022, 2016.

[53] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[54] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[55] Shan Carter, Zan Armstrong, Ludwig Schubert, Ian Johnson, and Chris Olah. Activation atlas. *Distill*, 2019. https://distill.pub/2019/activation-atlas.

Appendix A

Appendix

A.1 Nash Equilibrium

Nash equilibrium is defined as the point of convergence of GANs. In order to gain more intuition on Nash equilibrium, lets analyze an optimal discriminator. The discriminator objective is to minimize its loss function,

$$J_D(\theta_D, \theta_G) \ = -\frac{1}{2}\,\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] \ - \frac{1}{2}\,\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log\left(1 - D(G(\boldsymbol{z}))\right)] \tag{A.1}$$

In order to derive an optimal discriminator the following assumptions are made:

- $D(x)$ is optimized for all values of x
- P_{data} and P_{model} are non-zero everywhere

Equation (A.1) is rewritten as

$$J_D = -\frac{1}{2}\int p_{\text{data}}(\boldsymbol{x}) \log D(x) dx - \frac{1}{2}\int p_{\text{model}}(\boldsymbol{x}) \log 1 - D(x) dx$$

Differentiating the above equation with respect to $D(x)$ we get,

$$\frac{J_D}{dD(x)} = -\frac{1}{2}\int p_{\text{data}}(\boldsymbol{x})\frac{1}{D(x)} dx + \frac{1}{2}\int p_{\text{model}}(\boldsymbol{x})\frac{1}{1 - D(x)} dx$$

For an optimal discriminator the above derivative value should be 0.

$$\frac{1}{2}\int p_{\text{data}}(\boldsymbol{x})\frac{1}{D(x)} dx = \frac{1}{2}\int p_{\text{model}}(\boldsymbol{x})\frac{1}{1 - D(x)} dx$$

By rearranging the above equation we get,

$$D_{\text{opt}}(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_{\text{model}}(\boldsymbol{x})} \tag{A.2}$$

Nash equilibrium is the point where the generator generates samples similar to that of real data then, $p_{\text{data}}(\boldsymbol{x}) = p_{\text{model}}(\boldsymbol{x})$. Then Equation (A.3) would be,

$$D(\boldsymbol{x}) = \frac{1}{2} \tag{A.3}$$

Thus in Nash equilibrium the generator output is $p_{\text{data}}(\boldsymbol{x})$ and discriminator output is $\frac{1}{2}$.

A.2 Hyperparameters

A.2.1 Vanilla CycleGAN for Domain Adaptation

Hyperparameters of Vanilla CycleGAN used for generating real images from rendered images and conversely.
All images were scaled at the input with pixel values ranging between 0 and 1
Optimizer : Adam
Adam momentum weights : $\beta_1 = 0.9$, $\beta_2 = 0.999$
Generator learning rate : 1e-3
Discriminator learning rate : 1e-3
Cyclic loss co-efficient : $\lambda_1 = \lambda_2 = 10$
Image pool size : 50
Batch size : 16

A.2.2 Modified CycleGAN for Domain Adaptation

Hyperparameters of our modified CycleGAN used for generating real images from rendered images and conversely.
All images were scaled at the input with pixel values ranging between 0 and 1
Optimizer : Adam
Adam momentum weights : $\beta_1 = 0.9$, $\beta_2 = 0.999$
Generator learning rate : 5e-3
Discriminator learning rate : 5e-3
Cyclic loss co-efficient : $\lambda_1 = \lambda_2 = 12$
Cyclic loss decay : $1e - 4$
Identity loss co-efficient : 6
Mask loss decay : $1e - 3$
Image pool size : 50
Batch size : 16
Epochs : 100
Number of training steps: Generator, $m = 4$, Discriminator, $n = 1$

A.2.3 RetinaNet

Hyperparameters of RetinaNet used for training real, GAN, CAD, RECONST images for T-LESS objects and real, GAN and textured 3D model images for YCB objcets.
All images were scaled at the input with pixel values ranging between 0 and 1. The backbone layers were frozen during training.

A.2.3.1 T-LESS real images

Optimizer : Adam
Adam momentum weights : $\beta_1 = 0.9$, $\beta_2 = 0.999$
learning rate : 1e-5
Batch size : 1
Epochs : 100

A.2.3.2 T-LESS CAD images

Optimizer : Adam
Adam momentum weights : $\beta_1 = 0.9$, $\beta_2 = 0.999$
learning rate : 1e-5
Batch size : 1
Epochs : 50

A.2.3.3 T-LESS RECONST images

Optimizer : Adam
Adam momentum weights : $\beta_1 = 0.9$, $\beta_2 = 0.999$
learning rate : 1e-5
Batch size : 1
Epochs : 70

A.2.3.4 T-LESS GAN object 5

Optimizer : Adam
Adam momentum weights : $\beta_1 = 0.9$, $\beta_2 = 0.999$
learning rate : 2e-5
Batch size : 1
Epochs : 100

A.2.3.5 T-LESS GAN object 8

Optimizer : Adam
Adam momentum weights : $\beta_1 = 0.9$, $\beta_2 = 0.999$
learning rate : 2e-3
Batch size : 10
Epochs : 60

A.2.3.6 T-LESS GAN object 9

Optimizer : Adam
Adam momentum weights : $\beta_1 = 0.9$, $\beta_2 = 0.999$
learning rate : 2e-3
Batch size : 1
Epochs : 130

A.2.3.7 T-LESS GAN object 10

Optimizer : Adam
Adam momentum weights : $\beta_1 = 0.9$, $\beta_2 = 0.999$
learning rate : 2e-5
Batch size : 1
Epochs : 150

A.2.3.8 YCB real

Optimizer : Adam
Adam momentum weights : $\beta_1 = 0.9$, $\beta_2 = 0.999$
learning rate : 1e-3
Batch size : 1
Epochs : 60

A.2.3.9 YCB textured 3D model

Optimizer : Adam
Adam momentum weights : $\beta_1 = 0.9$, $\beta_2 = 0.999$
learning rate : 1e-3
Batch size : 1
Epochs : 80

A.2.3.10 YCB GAN

Optimizer : Adam
Adam momentum weights : $\beta_1 = 0.9$, $\beta_2 = 0.999$
learning rate : 1e-5
Batch size : 10
Epochs : 80

A.3 Object Detection Results

The RetinaNet object detection results for all the objects used in this thesis are given below. T-LESS objects are tested on test-scene 11 of T-LESS test dataset scenes and YCB objects are tested on a images extracted from YCB video dataset.

Figure A.1: Object detection results of T-LESS object 5 shown in red bounding box; tested on T-LESS test scene 11.

Figure A.2: Object detection results of T-LESS object 8 shown in orange bounding box; tested on T-LESS test scene 11.

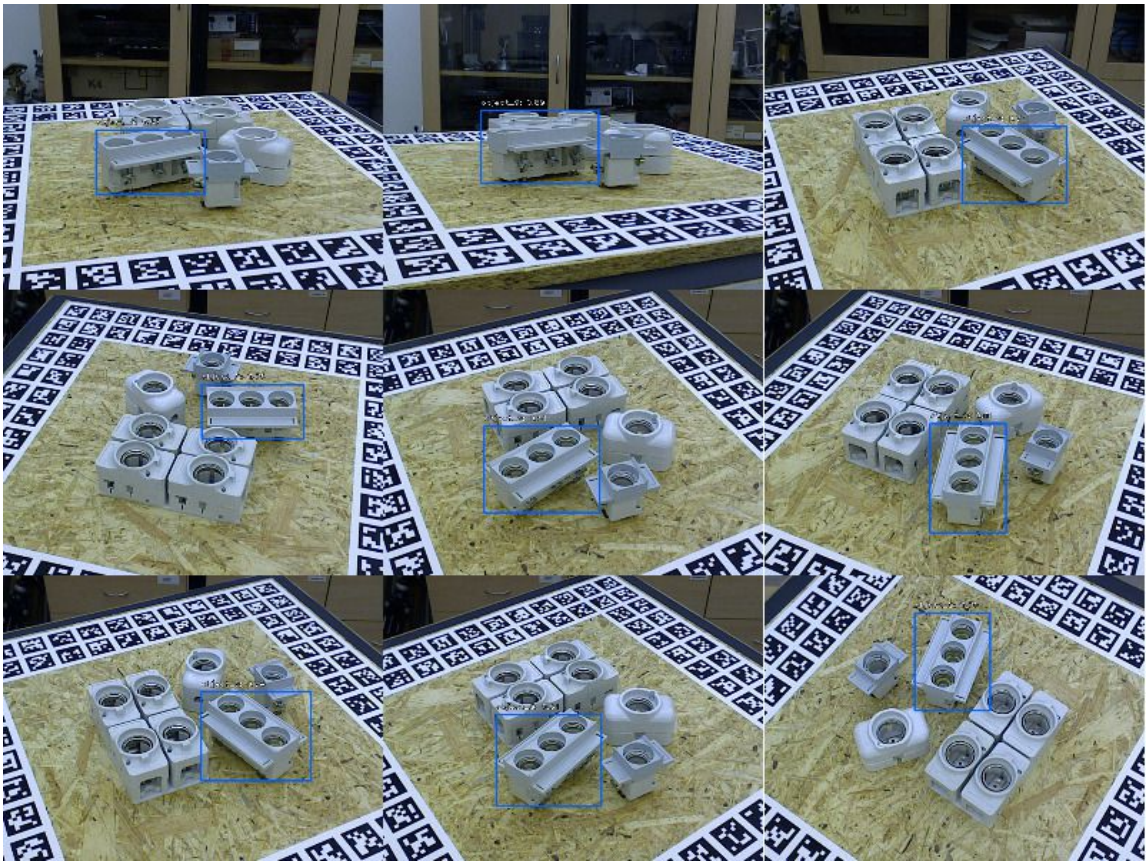


Figure A.3: Object detection results of T-LESS object 9 shown in blue bounding box; tested on T-LESS test scene 11.

Figure A.4: Object detection results of T-LESS object 10 shown in dark blue bounding box; tested on T-LESS test scene 11.

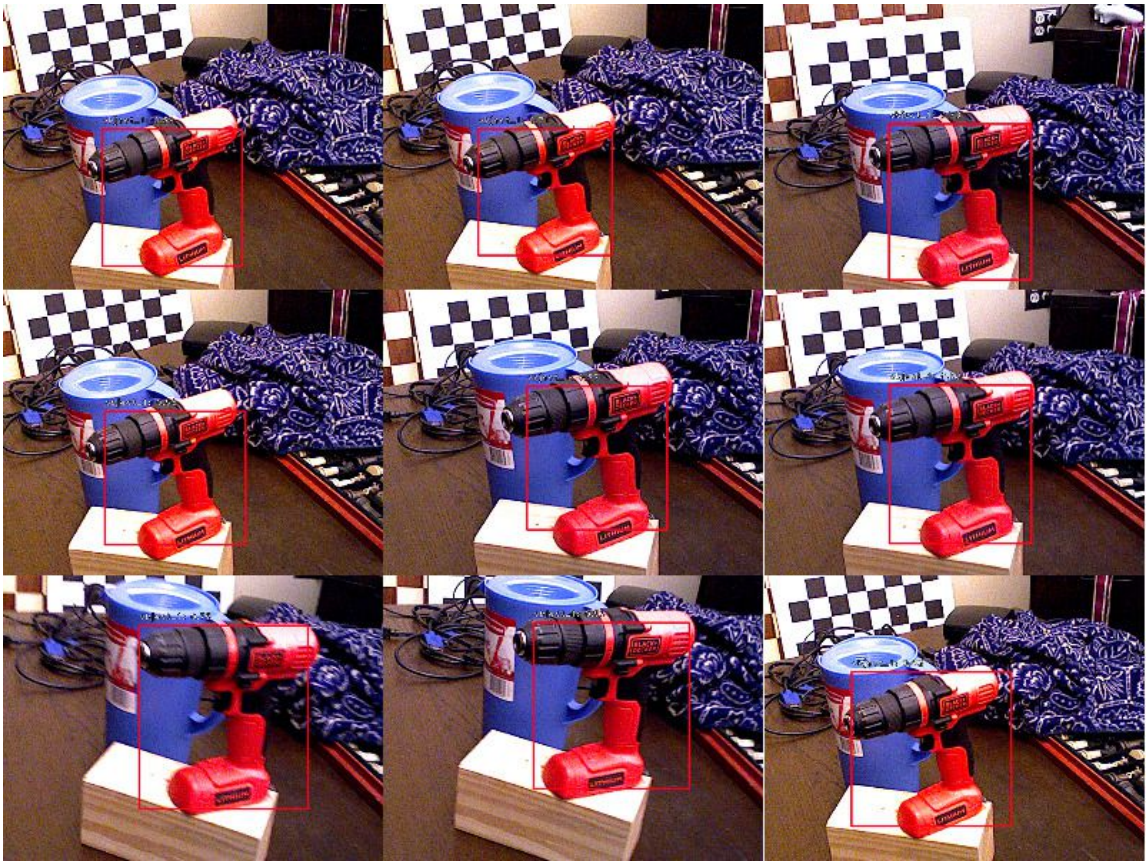


Figure A.5: Object detection results of YCB object power drill shown in red bounding box; tested on YCB video dataset extracted images.

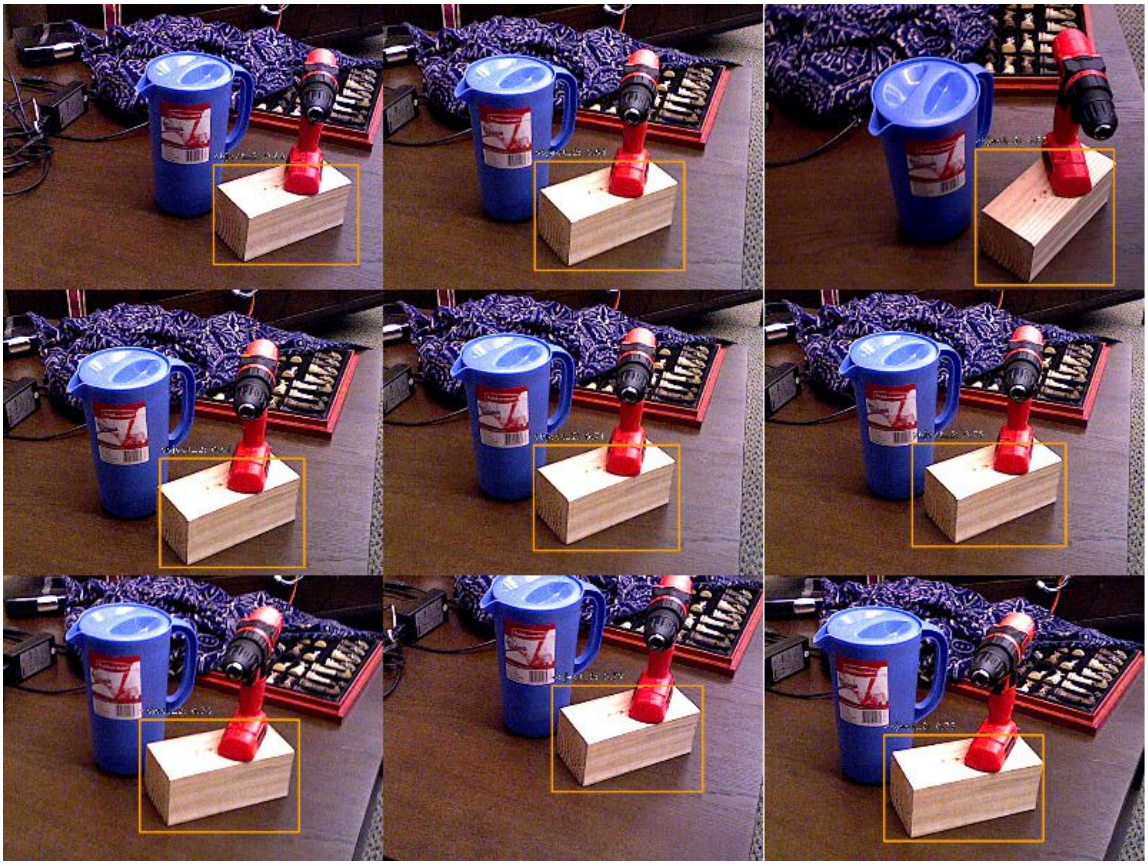


Figure A.6: Object detection results of YCB object wooden block shown in orange bounding box; tested on YCB video dataset extracted images.

Figure A.7: Object detection results of YCB object pitcher base shown in blue bounding box; tested on YCB video dataset extracted images.