TRANSFERRABLE LEARNING FROM SYNTHETIC DATA: NOVEL TEXTURE SYNTHESIS USING DOMAIN RANDOMIZATION FOR VISUAL SCENE UNDERSTANDING

By

MOHAMMAD KH M H M ANI

A thesis submitted to
the University of Birmingham
for the degree of
DOCTOR OF PHILOSOPHY

Intelligent Robotics Lab
School of Computer Science
College of Engineering and Physical Sciences
University of Birmingham
September 2021

ABSTRACT

Modern supervised deep learning-based approaches typically rely on vast quantities of annotated data for training computer vision and robotics tasks. A key challenge is acquiring data that encompasses the diversity encountered in the real world. The use of synthetic or computer-generated data for solving these tasks has recently garnered attention for several reasons. The first being the efficiency of producing large amounts of annotated data at a fraction of the time required in reality, addressing the time expense of manually annotated data. The second addresses the inaccuracies and mistakes arising from the laborious task of manual annotations. Thirdly, it addresses the need for vast amounts of data typically required by data-driven state-of-the-art computer vision and robotics systems. Due to domain shift, models trained on synthetic data typically underperform those trained on real-world data when deployed in the real world. Domain Randomization is a data generation approach for the synthesis of artificial data. The Domain Randomization process can generate diverse synthetic images by randomizing rendering parameters in a simulator, such as the objects, their visual appearance, the lighting, and where they appear in the picture. This synthetic data can be used to train systems capable of performing well in reality. However, it is unclear how to best approach selecting Domain Randomization parameters such as the types of textures, object poses, or types of backgrounds. Furthermore, it is unclear how Domain Randomization generalizes across various vision tasks or whether there are potential improvements to the technique. This thesis explores novel Domain Randomization techniques to solve object localization, detection, and semantic segmentation in cluttered and occluded real-world scenarios. In particular, the four main contributions of this dissertation are:

(i) The first contribution of the thesis proposes a novel method for quantifying the differences between Domain randomized and realistic data distributions using a small number of samples. The approach ranks all commonly applied Domain Randomization texture techniques in the existing literature and finds that the ranking is reflected in the task-based performance of an object localization task.

(ii) The second contribution of this work introduces the SRDR dataset - a large domain randomized dataset containing 291K frames of household objects widely used in robotics and vision benchmarking [23]. SRDR builds on the YCB-M [67] dataset by generating synthetic versions for images in YCB-M using a variety of domain randomized texture types and in 5 unique environments with varying scene complexity. The SRDR dataset is highly beneficial in cross-domain training, evaluation, and comparison investigations.

(iii) The third contribution presents a study evaluating Domain Randomization's generalizability and robustness in sim-to-real in complex scenes for object detection and semantic segmentation. We find that the performance ranking is largely similar across the two tasks when evaluating models trained on Domain Randomized synthetic data and evaluating on real-world data, indicating Domain Randomization performs similarly across multiple tasks.

(iv) Finally, we present a fast, easy to execute, novel approach for conditionally generating domain randomized textures. The textures are generated by randomly sampling patches from real-world images to apply to objects of interest. This approach outperforms the most commonly used Domain Randomization texture method from 13.157 AP to 21.287 AP and 8.950 AP to 19.481 AP in object detection and semantic segmentation tasks. The technique eliminates manually defining texture distributions to sample Domain randomized textures. We propose a further improvement to address low texture diversity when using a small number of real-world images. We propose to use a conditional GAN-based texture generator trained on a few real-world image patches to increase the texture diversity and outperform the most commonly applied Domain Randomization texture method from 13.157 AP to 20.287 AP and 8.950 AP to 17.636 AP in object detection and semantic segmentation tasks.

Contents

Page

Glossary **xxix**

Acronyms **xlii**

1 Introduction **1**

1.1 Motivation 4

1.2 Contributions 6

1.3 Thesis Outline 8

1.4 Publications 10

2 Background **11**

2.1 Synthetic Data Generation 11

2.1.1 The Uncanny Valley - Addressing Realism in Synthetic Data 14

2.2 Importance of Data 16

2.3 Solving Object-Centric Computer Vision Tasks 17

2.3.1 Classical Approaches 19

2.3.2 Deep Learning 20

3 Learning From Synthetic Data **28**

3.1 Motivation 29

3.2 Transfer to Real-World 34

3.2.1 Difficulty in Transfer 34

3.2.2 Domain Shift . . . 34
3.2.3 Notations and Definitions . . . 36
3.3 Applications of Synthetic Data within Computer Vision and Robotics . . . 38
3.3.1 Synthetic Data Only . . . 40
3.3.2 Combining Synthetic and Real Data . . . 42
3.4 Refined Synthetic Data . . . 44
3.5 Procedural Synthetic Data Generation . . . 47
3.6 Conclusion . . . 50

4 Domain Randomization **51**
4.1 Introduction . . . 51
4.2 Algorithm . . . 53
4.3 Applications . . . 54
4.3.1 Variations in Applying DR . . . 57
4.4 Conclusion . . . 59

5 QDRNet - Quantifying the use of Domain Randomization **62**
5.1 Introduction . . . 63
5.2 Method . . . 66
5.2.1 Texture Domain Randomization . . . 67
5.2.2 Quantifying Distances Between Distributions . . . 68
5.2.3 Image Space . . . 70
5.2.4 Feature Space . . . 71
5.2.5 Localization Task . . . 73
5.3 Data Generation . . . 73
5.3.1 Toy Dataset . . . 75
5.3.2 DR Datasets . . . 75
5.3.3 Texture Randomization Routine . . . 77
5.3.4 Static Background . . . 79

5.3.5 Real-world Backgrounds . . . 79
5.4 Experiments . . . 80
5.4.1 Image Space . . . 81
5.4.2 Feature Space . . . 82
5.5 Results and Discussion . . . 83
5.5.1 Image Space . . . 83
5.5.2 Feature Space . . . 89
5.6 Conclusion . . . 93

6 SRDR Dataset: Sim-to-Real Domain Randomized dataset for Benchmarking Tasks in Visual Sim-To-Real Transfer 96

6.1 Introduction . . . 97
6.2 Related Work . . . 101
6.2.1 Datasets . . . 101
6.2.2 Simulators . . . 103
6.3 The SRDR Dataset . . . 105
6.3.1 Data Generation . . . 107
6.3.2 Testing . . . 115
6.3.3 Dataset Statistics . . . 115
6.4 Conclusion . . . 118

7 Generalizability of DR for Multi-Tasks 121

7.1 Method . . . 124
7.1.1 Problem Definition . . . 124
7.1.2 Network Architecture . . . 125
7.1.3 Evaluation Metrics . . . 125
7.1.4 Dataset Generation . . . 127
7.2 Sensitivity to Weight Initialization . . . 130
7.2.1 Experimental Setup . . . 131

7.2.2 Results . . . 133
7.3 Object Poses . . . 136
7.3.1 Experimental Setup . . . 136
7.3.2 Evaluation on Synthetic Images . . . 137
7.3.3 Evaluation on Real Images . . . 141
7.4 Image Backgrounds . . . 145
7.4.1 Experimental Setup . . . 147
7.4.2 Photorealistic Background . . . 147
7.4.3 Active-Vision Background . . . 150
7.5 Scene Replication . . . 155
7.5.1 Experimental Setup . . . 157
7.5.2 Real-Texture Equivalent . . . 159
7.5.3 Varying Backgrounds . . . 161
7.5.4 DR Textures . . . 164
7.6 Discussion . . . 176
7.6.1 Poses . . . 176
7.6.2 Backgrounds . . . 177
7.6.3 Textures . . . 178
7.6.4 Illumination . . . 180
7.7 Conclusion . . . 180

8 Conditional Domain Randomization: Synthesizing Textures via Image Patches 182

8.1 Introduction . . . 183
8.2 Related Work . . . 187
8.2.1 Domain Randomization . . . 187
8.2.2 Image Synthesis . . . 189
8.2.3 Texture Generation . . . 190
8.3 Method . . . 191

8.3.1 Approach Overview . . . 191
8.3.2 Dataset . . . 193
8.3.3 Training Implementation . . . 198
8.4 Experiments . . . 199
8.4.1 Unconditional Real-World Image Patches . . . 199
8.4.2 Conditional Real-World Image Patches . . . 202
8.4.3 Size of Image Patches . . . 204
8.4.4 GAN-Based Image Patches . . . 208
8.5 Conclusion . . . 217

9 Conclusions and Future Work **218**
9.1 Summary . . . 221
9.1.1 Key Contributions . . . 224
9.2 Limitations and Future Work . . . 225

A Additional Image Samples For Conditional Domain Randomization **228**

References **231**

List of Figures

2.1 The illustration is taken from [27], demonstrating the application of textures on a 3D object, given a 3D object with variable name cMesh, and texture map with variable name cTexture2D. The 3D can is composed of several thousands of vertices and triangles, as shown in the bottom left. A texture map is an image of fixed size that contains information about an object's visual appearance. This texture is stretched over the 3D mesh object to produce a texturized object, as shown in the top right. 12

2.2 Figure illustrating a sample mesh model, texture map, and textured mesh model on top, and the process for generating a synthetic scene using a renderer on the bottom. Each mesh is defined as a set of vertices, edges, and faces which describe the 3D shape of an object. A vertex is a point in 3D space (X, Y, Z). Each connection between two vertices forms an edge, and each set of closed edges comprises a face as shown on top. A collection of texture coordinates defines a mapping from a 2D texture map to the 3D shape, where each texture coordinate is associated with a vertex on the 3D model. The textured model is seen in the top right. Rendering a synthetic scene requires a description of where the camera is oriented, the lighting, and the position of the object. The object is transformed to the camera coordinate system via Rotation and Translation. The final rendered scene is in the bottom right. 13

2.3 Figure illustrating a synthetic image with a real-world background on the left and the real-world version on the right from the YCB-M dataset [67]. To generate the scene on the left, we use information regarding the position and orientation of the camera, positions of illumination sources, 3D object meshes, their textures, and their positions are needed to match the real-world scene on the right. 14

2.4 Ray-traced illumination used to generate two photorealistic human faces using Unreal Engine's MetaHuman [50]. While the process of approximating illumination is more computationally expensive, the results are more photorealistic. Figure taken from [50]. 15

2.5 A robotic arm that can be used to interact with household objects in an environment. This setup is typical for object-centric tabletop scenes in visual recognition tasks such as object localization, detection, or segmentation. The objects of interest are the camera's main focal point which is attached to the robotic arm. 18

2.6 SIFT features extracted from the training image on the left, and evaluated on the test image on the right [123]. 20

2.7 Image taken from Zhu et al. [247] showing a street sign classification task, where each sign contains a label describing the visible sign. For example, pl20 denotes a speeding limit sign of 20 Mph [247]. 21

2.8 Example of an object detection task, where a model was tasked to detect certain classes from a given image. Image taken from Redmon et al. [165]. The bounding boxes define the positions of the objects of interest, while the labels "person" and "cat" describe what appears within the region. 24

2.9 Example of a basic feed-forward neural network containing five layers. The input layer is the first layer in the network where the data is first passed through the network. The output layer is the final layer in the network. In this example, each subsequent layer's nodes (represented as circles) are connected to the previous layer's nodes. This structure is called a fully connected neural network. 26

2.10 An illustration showing intermediate layer representations taken from [86]. Training a network end-to-end using a convolutional neural network, resulting in a trained model capable of producing probabilities of what is visible in the image. The visualizations are the activations of the network, providing some insights into what the network is learning. 27

3.1 Inconsistent labeling between similar frames in the MS-COCO dataset. Figure 3.1d does not contain the class keyboard. 30

3.2 Missing annotations for the book category in the MS-COCO dataset. Instances of the books are not labeled consistently in Figures 3.2b and 3.2d. 31

3.3 Sample data from the SRDR data described in Chapter 6 showing clean semantic segmentation annotations for a synthetic RGB scene. 32

3.4 The figure shows an example of domain mismatch in autonomous driving and robotic manipulation scenarios. We see differences in visual appearance between the training images on the left and the test images on the right. The autonomous vehicle scenario shows a training image in bright daylight, while the test image is at night in the snow. The robotics scenario shows differences in illumination, shadows, object shape, and textures. These differences demonstrate a domain mismatch between the source (left) and target (right) domains. Images taken from the BDD100k dataset [234] and from Bousmalis et al. [15]. 35

3.5 T-SNE visualization of the activations from the second to last layer of a network [55, 181]. The training and test data differ in how the points are distributed and clustered. Image taken from [181]. 36

3.6 Images were taken from Gaidon et al. [54]. The figures show five real-world scenes on the right from the KITTI dataset [57], and five matched synthetic scenes from the VKITTI dataset [54] on the left. The matched artificial scenes differ visually from the real-world scenes, such as the lighting (global illumination appears to light up the entire scene compared to the real-world images), textures and material properties used on the cars, or softer shadows in reality. 39

3.7 Sample training images from Su et al. [192], showing an approach to synthesize synthetic images by placing the 3D models on top of real-world images. A renderer generates synthetic images by overlaying the 3D models on the left from different viewpoints with random real-world backgrounds for solving viewpoint estimation. 42

3.8 A Process for training a CNN for object instance detection taken from Dwibedi, Misra, and Hebert [45]. The idea is to place 3D models on real-world background scenes in more natural positions, such as on the surface of a table, to generate a new synthetic dataset. This synthetic dataset is used to train an object detection network. 43

3.9 Sample images showing style consistency between real-world images at the top and semantic label on the left. The generated synthetic scenes at the bottom use the semantic label on the left and match the style from the real-world images. The generated synthetic images are style consistent by maintaining similar illumination conditions such as the time of day, the visual appearance of the roads, and the sky. Image is taken from Wang et al. [215]. 46

3.10 Differentiable rendering overview as presented by Kato et al. [92]. With differentiable rendering, we may compute gradients of some objective function with respect to the scene parameters and ground-truth in the image on top. Traditional rendering does not allow for the computation of gradients which is required when using neural networks. 49

4.1 A visual representation of a synthetic and real-world data distribution on top, for example, a training dataset of synthetic household tabletop scenes and a test dataset of real-world household tabletop scenes. When using domain randomization (DR), the synthetic data distribution is expanded by including some combination of variations in textures, poses, illumination, or backgrounds. This technique would broaden the data distribution such that a real-world sample may appear as another variation in the training distribution. 52

4.2 Samples of DR data from the original DR work by Tobin et al. [202]. Simple geometric objects are randomized and used as part of the training set for the localization of an object of interest. Object textures, positions, camera positions, and backgrounds are randomized. 54

4.3 Sample images taken from Brendel and Bethge [16]. The figure shows original images to the left and scrambled images to the right. An off-the-shelf VGG-16 [187] model achieves 90% accuracy on an image classification task evaluated on the original unscrambled images and 79.4% accuracy on scrambled images. Despite a breakdown in global shape, the model was capable of yielding good accuracy, indicating that local image features are important. 60

5.1 Sample textures used to generate domain randomized (DR) data. A combination of non-complex textures (Flat RGB, Gradient) and complex textures (Checkerboard, Striped, Zig-Zag, and using Perlin noise [152]) are used to create the data. The textures were selected to cover the types used in existing DR literature from Table 4.3. 65

5.2 The localization network using a ResNet-50 backbone to predict the position x, y, z of the object of interest [71]. 72

5.3 The flow of data for QDRNet. Using two data distributions, real-equivalent images and domain randomized synthetic images, we extract the features using a ResNet-50 network [71]. The extracted feature vectors are passed through a WGAN-GP critic to compute the Wasserstein distance using the standard loss function. An additional step of dimensionality reduction is used to reduce computational cost before computing FID. 72

5.4 Real-world objects from the YCB dataset taken from [23]. Each of the real-world objects has a corresponding mesh and high resolution texture associated with it. . . 74

5.5 Toy dataset sample distributions. The toy dataset is generated by uniformly sampling Flat RGB colors from two known Gaussian distributions in the HSV color space. The shades of blue have a fixed hue of 220 and a value of 1, while uniformly sampling for saturation. In this dataset sample, the saturation for our first distribution *P* has a mean of 0.3 and std of 0.05, while our second distribution *Q* has a mean of 0.65 and std of 0.05. The difference between means in the two distributions is 0.35. 74

5.6 Sample Images from the dataset generation routine. Synthetic images of the object of interest are positioned around the center of the table. 76

5.7 2D-Histogram showing the x, y positions of the object of interest around the center of the table. The histogram corresponding to the dataset consists of 5000 samples. . 77

5.8 Sample data from the black background dataset. The dataset consists of the real texture applied to the object of interest on the left and the domain randomized (DR) version on the right. The black background, table, and illumination are fixed. Object poses and textures are randomized. 79

5.9 Sample data from the real-world backgrounds dataset. The previous black backgrounds were replaced with real-world images from the NYU Depth V2 dataset [142]. The real texture versions appear on the left, and the domain randomized (DR) versions are on the right. 80

5.10 Comparison of JSD and WD estimates using a toy dataset shows that WD estimate provides a more practical way of quantifying separations between distributions when they are far apart. 85
5.11 Figure showing the WD estimate using commonly used texture randomization techniques. We compute the estimate between real-equivalent synthetic and DR synthetic RGB images with black backgrounds. There are three distinct groupings between patterned (Checkerboard, Striped, Zig-Zag), non-patterned (Flat RGB and Gradient RGB) and dominant noise (Perlin). 86
5.12 Figure showing results from a localization task where the model was trained on DR synthetic images, and evaluated on real-equivalent synthetic images with black backgrounds. The MSE is between the predicted and ground truth positions of the object on the table. 87
5.13 Figure showing WD estimate for various texture randomization techniques when operating in the image space. There are no clear separations in randomization techniques in the image space using real-world backgrounds. We are only able to differentiate between methods involving non-Perlin and Perlin noise. 88
5.14 WD between real-equivalent and DR synthetic data with backgrounds from NYU V2 dataset [142] when operating in the feature space. The distance is measured using feature vectors extracted from a ResNet-50 backbone [71]. When working in the feature space, we can more clearly distinguish between the various texture randomization techniques. 90
5.15 FID estimates using the real-world background dataset. 91
5.16 Figure illustrating the comparison of localization task, WD, and FID estimate in feature space. The values are normalized and sorted by the lowest mean error in the localization task. The effects of additional noise (Perlin) increase the difficulty in obtaining a clear ranking. In general, the addition of dominant Perlin noise appears to aid performance, in addition to using patterned textures. 92

6.1 We created the Sim-to-Real Domain Randomized (SRDR) dataset by taking real-world images from the YCB-M dataset [67] (top left) and matching 3D household models [23] (e.g., gelatin box, cracker box, meat can, and tuna can) to their positions in the real-world. We generated matched synthetic (top middle) and DR synthetic (top right) versions of the real-world, against five unique environments. Each scene (real-world, synthetic, or DR synthetic) contains pixel-wise segmentation of objects of interest (bottom left), 2D/3D bounding box coordinates (bottom middle), and depth images (bottom right). The camera positions and 3D positions of each object of interest are also provided. 99

6.2 We show sample images from the SRDR dataset displaying synthetic (left column), DR synthetic (middle column), and real-world images (right column). The DR images use all the most commonly applied texture randomization techniques in the existing literature and were using real-world backgrounds from the Active Vision dataset [4]. 100

6.3 Sample Image taken from NVIDIA Isaac Sim [144]. Parameters to render the scene are randomized and non-reproducible. 105

6.4 Sample training images using backgrounds from the Active-Vision dataset [4]. . . 108

6.5 Sample training images using backgrounds from the Structured3D dataset [244]. . 109

6.6 Sample training images using backgrounds from the IRLab. 110

6.7 Sample training images using backgrounds from the Photorealistic dataset [74]. . . 111

6.8 Sample training images using backgrounds from the Scenenet dataset [132]. 112

6.9 Sample annotations from the YCB-M dataset highlighting misaligned segmentation masks and bounding boxes. Grenzdörffer, Günther, and Hertzberg [67] use ArUco markers and generated initial guesses of object poses by using PoseCNN [228], then manually refining the guesses to remove false positives and missing objects. Despite the manual cleanup, there are still some imperfections such as Figure 6.9c, which has the segmentation mask slightly rotated relative to the original position of the object. Similarly, Figure 6.9b shows the large clamp bounding box and segmentation mask shifted to the left. 114

6.10 Figure showing the distribution of the number of objects per frame across the SRDR dataset. SRDR most commonly contains scenes with four to six objects. 116

6.11 Figure showing the total number of object instances across the SRDR dataset. The most transparent bars highlight the number of objects greater than 25% visible, while the second most transparent bar shows objects that are more than 75% visible. Objects that are highly occluded (less than 25% visible) are solid bars. 117

6.12 Visibility across all frames for a subset of four objects varying in shape and size. . 119

6.13 Position of object centroids across all frames for a subset of four objects varying in shape and size. 120

7.1 Figure showing how IoU is computed using bounding boxes. The RGB image [67] on the left with a green bounding box represents the ground truth, and the red boxes represent a model's prediction. To compute the model's accuracy, IoU is used by using the ratio between the overlapping area of the ground truth bounding box and the predicted bounding box and the total area from the ground truth and model predictions as shown on the right. 126

7.2 Sample images from each of the non-replicated scene datasets using realistic household objects from the YCB dataset [23]. The positions and orientation for each of the object is sampled from a uniform distribution for each frame in each dataset. Illumination and camera position remain fixed, and a different background is applied to each image in the datasets. The backgrounds show varying degrees of realism and background clutter, which acts as distractor objects. 129

7.3 Figure showing object models and real-equivalent textures from the YCB dataset [23]. The 20 objects shown are used in all experiments involving replicated scenes from the YCB-M dataset [67]. 130

7.4 Sample images from several object-centric real-world scenes from the YCB-M dataset [67] containing the power drill, banana, mustard bottle, bleach cleanser. Ground truth annotations are overlaid with the RGB images. 132

7.5 Sample images from the real-world test dataset consisting of 169 images from the YCB-M dataset [67]. The scene contains four objects found in the synthetic training set, with no clutter or occlusion. 138

7.6 Using weights from the highest performing network (striped), we visualize some of the predictions from the network. Color has been removed from the images, apart from where an object has been detected and segmented. These are some examples where the network trained on Striped synthetic images, is able to do well. 142

7.7 Visualization of network predictions on a real-world dataset when using Striped weights and the IRLab table background dataset from the SRDR dataset described in Chapter 6. The model has many false positives and commonly mistakes the ArUco markers as an object of interest. 146

7.8 Visualization of training images from the SRDR dataset described in Chapter 6 using replaced backgrounds from the Active-Vision dataset [4]. The set of backgrounds are from a real-world dataset and contains background clutter in the form of additional household objects. Note that there are some instances where real bananas appear in the background. These are not labeled as a sample in a dataset. . 148

7.9 Visualization of training images from the SRDR dataset described in Chapter 6 using replaced backgrounds from the Photorealistic dataset [74]. The backgrounds used are from a photorealistic synthetic dataset containing a high degree of clutter from household objects. None of the objects in the backgrounds are included in the training dataset. 149

7.10 Visualization of the network predictions on a real-world dataset when using Checkerboard weights and using replaced backgrounds from the Active-Vision dataset [4]. Using a unique background per scene in the training set eliminates the false positives previously seen in Figure 7.7. While there are correct, true positive and true negative predictions, there are still some false negatives with missed detections of objects and misclassifications of them. 156

7.11 Figure showing several samples of the scissors class from the training set. The two scenes depicted on the left and right are the only scenes containing the scissors class. Note that the object is partially occluded in most instances and would be challenging to learn from these samples. 162

8.1 Comparison between CDR (top left) and traditional DR approaches (top right using Flat RGB). The CDR approach applies textures visually more similar to object classes in the target dataset. Bottom left is the synthetic real-texture versions, and bottom right is the real-world sample from the YCB-M dataset [67]. 186

8.2 We present an approach for synthesizing synthetic images based on real-world image patches for solving detection and segmentation tasks. Natural images provides the desired texture complexity when using DR, while conditionally applying textures based on the objects of interest adds visually relevant information. 192

8.3 Figure of all object classes used to generate image patches for Texture A. Image taken from [134]. 194

8.4 Samples real-world objects used to generate image patches for Texture B. 195

8.5 Texture A: textures generated from image patches. Each patch is of size 128x128, and uniformly sampled from a set of real-world images. 195
8.6 Texture B: textures generated from image patches. Each patch is of size 512x512 then resized to 128x128, and uniformly sampled from a set of real-world images. . 197
8.7 Samples of patch-based textures generated from Texture B. Each of the initial crops of varying sizes is resized to 128x128, to ensure the same spatial dimension is preserved across all experiments. Larger objects contain more visual information in larger crop sizes. However, smaller and thinner objects such as the board marker contains more of the background. 207
8.8 Samples generated from the unconditional GAN model from Texture A. 211
8.9 Samples generated from the unconditional GAN model from Texture B. 212
8.10 Samples generated from the conditional GAN model. 215

A.1 Additional CDR samples using the proposed method presented in Chapter 8. The left column shows the CDR samples, while the right column shows the real-world images from the YCB-M dataset [67]. 229
A.2 Additional conditional GAN-based samples using the proposed method presented in Chapter 8. The left column shows the conditional GAN-based approach samples, while the right column shows the real-world images from the YCB-M dataset [67]. 230

List of Tables

4.1 Table showing the types of tasks, object complexity, and scene complexity in the current literature using domain randomization (DR). 55

4.2 Table showing the selection of parameters when applying domain randomization (DR) in the existing literature. The most commonly used techniques are randomizing textures and object poses. [1] Gaussian noise added as a post-processing step once the images were generated. [2] Applied to the background only. 58

4.3 Table showing texture randomization techniques applied in current literature. The heavily favored approach is to use flat RGB textures, in which each texture is a single RGB color sampled from a predetermined distribution. 59

6.1 Table showing related object-centric synthetic datasets. The SRDR dataset is highlighted in bold. [1] Only distractor objects are texture randomized from a selection of 7 textures. [2] 15 objects are from LineMod dataset, and 14 from Rutgers APC (RU-APC) [169]. [3] Distractors are in the form of background objects from different categories. [4] SRDR dataset replicates scenes from the YCB-M [67] dataset using RealSense camera. 102

6.2 Overview of existing simulators allowing generating synthetic data. [1] Denoting built-in tools for generating the data. 104

7.1 Instances per class for a training set of size 2320 and a test set of 647 real-world images at a resolution of 640x480 from the YCB-M dataset [67]. This dataset is used for determining a set of pre-trained weights for Mask-RCNN for object detection and segmentation tasks. 131

7.2 Bounding box AP using both pre-trained COCO [116] and pre-trained ImageNet (MSRA) weights [71] with a Mask-RCNN network [72] and a ResNet-50 backbone. Pre-trained COCO outperforms pre-trained ImageNet across the board. . . . 134

7.3 Per-Category bounding box AP using both pre-trained COCO [116] and pre-trained ImageNet (MSRA) weights [71] with a Mask-RCNN network [72] and a ResNet-50 backbone. Pre-trained COCO outperforms pre-trained ImageNet for all classes. 134

7.4 Semantic segmentation mask AP using both pre-trained COCO [116] and pre-trained ImageNet (MSRA) weights [71] with a Mask-RCNN network [72] and a ResNet-50 backbone. Pre-trained COCO outperforms pre-trained ImageNet across the board. 135

7.5 Per-Category semantic segmentation mask AP using both pre-trained COCO [116] and pre-trained ImageNet (MSRA) weights [71] with a Mask-RCNN network [72] and a ResNet-50 backbone. Pre-trained COCO outperforms pre-trained ImageNet for all classes apart from Power Drill. 135

7.6 Instances per class for the real-world training set of size 2320 and an unoccluded test set of size 169 real-world images at a resolution of 640x480 from the YCB-M dataset [67]. The test set is a single scene shown in Figure 7.5 containing the four objects and is not visible in the real-world training set. 139

7.7 Per-Category object detection (bounding box) AP using COCO weights [116]. The network was fine-tuned using synthetic images with the original object textures defined as Real-Texture. The remaining datasets are the ten texture DR techniques used in the current literature. Each model was evaluated on a synthetic Real-Texture test set of size 2700 for the four objects of interest. 140

7.8 Per-Category object segmentation AP using COCO weights [116]. The network was fine-tuned using synthetic images with the original object textures defined as Real-Texture. The remaining datasets are the ten texture DR techniques used in the current literature. Each model was evaluated on a synthetic Real-Texture test set of size 2700 for the four objects of interest. 141

7.9 Object detection AP scores evaluating several models on a real-world test set from the YCB-M dataset [67] shown in Figure 7.5. The weights used are the highest performing texture DR method (striped images), the real-texture synthetic images, and real-world images. 144

7.10 Object semantic segmentation AP scores evaluating several models on a real-world test set from the YCB-M dataset [67] shown in Figure 7.5. The weights used are the highest performing texture DR method (striped images), the real-texture synthetic images, and real-world images. 144

7.11 Per-Category object detection (bounding box) AP using COCO weights [116]. The network was fine-tuned using synthetic images with the original object textures defined as Real-Texture. The remaining datasets are the ten texture DR techniques used in the current literature. Each synthetic dataset uses a unique background per frame from a synthetic photorealistic dataset [74]. The dataset is described in detail in Chapter 6. Each model was evaluated on a real-world test set from a single scene shown in Figure 7.5 containing the four objects from the YCB-M dataset [67]. 151

7.12 Per-Category object semantic segmentation AP using COCO weights [116]. The network was fine-tuned using synthetic images with the original object textures defined as Real-Texture. The remaining datasets are the ten texture DR techniques used in the current literature. Each synthetic dataset uses a unique background per frame from a synthetic photorealistic dataset [74]. The dataset is described in detail in Chapter 6. Each model was evaluated on a real-world test set from a single scene shown in Figure 7.5 containing the four objects from the YCB-M dataset [67]. 152

7.13 Per-Category object detection (bounding box) AP using COCO weights [116]. The network was fine-tuned using synthetic images with the original object textures defined as Real-Texture. The remaining datasets are the ten texture DR techniques used in the current literature. Each synthetic dataset uses a unique background per frame from the real-world Active-Vision dataset [4]. The dataset is described in detail in Chapter 6. Each model was evaluated on a real-world test set from a single scene shown in Figure 7.5 containing the four objects from the YCB-M dataset [67]. 153

7.14 Per-Category object semantic segmentation AP using COCO weights [116]. The network was fine-tuned using synthetic images with the original object textures defined as Real-Texture. The remaining datasets are the ten texture DR techniques used in the current literature. Each synthetic dataset uses a unique background per frame from the real-world Active-Vision dataset [4]. The dataset is described in detail in Chapter 6. Each model was evaluated on a real-world test set from a single scene shown in Figure 7.5 containing the four objects from the YCB-M dataset [67]. 154

7.15 Instances per class for a synthetic dataset replicating real-world scenes from the YCB-M dataset [67]. The scene replication process is described in more detail in Chapter 6 From the 31 scenes in the YCB-M dataset, 26 were separated for training and validation, and 5 were used as the test set. The split ensures that all objects are represented in the test set, and no frames from the test set appear in the training sets. 158

7.16 Per-Category object detection (bounding box) and semantic segmentation AP scores using models trained with the real-texture synthetic images from the SRDR dataset with the IRLab background described in Chapter 6. The models are evaluated using five of the replicated synthetic real-texture scenes (defined as “Synthetic”) and their real-world equivalents (defined as “Real-World”) from the YCB-M dataset [67]. A breakdown of the instances in the test set is in Table 7.15. 160

7.17 Comparative results on a real-world test set of 837 images. All models were trained using Mask R-CNN using the ResNet-50-C4 backbone. Backgrounds of the synthetic training sets were replaced with images from synthetic and real datasets. Scores shown are for object detection (bounding box). 163

7.18 Comparative results on a real-world test set of 837 images. All models were trained using Mask R-CNN using the ResNet-50-C4 backbone. Backgrounds of the synthetic training sets were replaced with images from synthetic and real datasets. Scores shown are for the segementation task. 163

7.19 Table showing comparative results on a real-world test set of 837 images from the YCB-M dataset [67] for the object detection task. All models were trained using Mask R-CNN using the ResNet-50-C4 backbone and each of the texture DR datasets from the SRDR dataset described in Chapter 6. The Synthetic Real Textured dataset is the original object textures, and the Real World dataset is a model trained on the matched real-world dataset from YCB-M [67]. The results shown are from the models trained with synthetic backgrounds from the photorealistic dataset [74] from the SRDR dataset. 165

7.20 Table showing comparative results on a real-world test set of 837 images from the YCB-M dataset [67] for the object segmentation task. All models were trained using Mask R-CNN using the ResNet-50-C4 backbone and each of the texture DR datasets from the SRDR dataset described in Chapter 6. The Synthetic Real Textured dataset is the original object textures, and the Real World dataset is a model trained on the matched real-world dataset from YCB-M [67]. The results shown are from the models trained with synthetic backgrounds from the photorealistic dataset [74] from the SRDR dataset. 166

7.21 Table showing comparative results on a real-world test set of 837 images from the YCB-M dataset [67] for the object detection task. All models were trained using Mask R-CNN using the ResNet-50-C4 backbone and each of the texture DR datasets from the SRDR dataset described in Chapter 6. The Synthetic Real Textured dataset is the original object textures, and the Real World dataset is a model trained on the matched real-world dataset from YCB-M [67]. The results shown are from the models trained with synthetic backgrounds from the Scenenet dataset [132] from the SRDR dataset. 168

7.22 Table showing comparative results on a real-world test set of 837 images from the YCB-M dataset [67] for the object segmentation task. All models were trained using Mask R-CNN using the ResNet-50-C4 backbone and each of the texture DR datasets from the SRDR dataset described in Chapter 6. The Synthetic Real Textured dataset is the original object textures, and the Real World dataset is a model trained on the matched real-world dataset from YCB-M [67]. The results shown are from the models trained with synthetic backgrounds from the Scenenet dataset [132] from the SRDR dataset. 169

7.23 Table showing comparative results on a real-world test set of 837 images from the YCB-M dataset [67] for the object detection task. All models were trained using Mask R-CNN using the ResNet-50-C4 backbone and each of the texture DR datasets from the SRDR dataset described in Chapter 6. The Synthetic Real Textured dataset is the original object textures, and the Real World dataset is a model trained on the matched real-world dataset from YCB-M [67]. The results shown are from the models trained with synthetic backgrounds from the Structured3D dataset [244] from the SRDR dataset. 170

7.24 Table showing comparative results on a real-world test set of 837 images from the YCB-M dataset [67] for the object segmentation task. All models were trained using Mask R-CNN using the ResNet-50-C4 backbone and each of the texture DR datasets from the SRDR dataset described in Chapter 6. The Synthetic Real Textured dataset is the original object textures, and the Real World dataset is a model trained on the matched real-world dataset from YCB-M [67]. The results shown are from the models trained with synthetic backgrounds from the Structured3D dataset [244] from the SRDR dataset. 171

7.25 Table showing comparative results on a real-world test set of 837 images from the YCB-M dataset [67] for the object detection task. All models were trained using Mask R-CNN using the ResNet-50-C4 backbone and each of the texture DR datasets from the SRDR dataset described in Chapter 6. The Synthetic Real Textured dataset is the original object textures, and the Real World dataset is a model trained on the matched real-world dataset from YCB-M [67]. The results shown are from the models trained with real-world backgrounds from the Active-Vision dataset [4] from the SRDR dataset. 173

7.26 Table showing comparative results on a real-world test set of 837 images from the YCB-M dataset [67] for the object segmentation task. All models were trained using Mask R-CNN using the ResNet-50-C4 backbone and each of the texture DR datasets from the SRDR dataset described in Chapter 6. The Synthetic Real Textured dataset is the original object textures, and the Real World dataset is a model trained on the matched real-world dataset from YCB-M [67]. The results shown are from the models trained with real-world backgrounds from the Active-Vision dataset [4] from the SRDR dataset. 174

7.27 Table showing the rankings for object detection from Table 7.25, semantic segmentation from Table 7.26, and localization task from Figure 5.16. A rank of 1 indicates the highest performing texture used when implementing DR. Rankings for the detection and segmentation tasks are using matched real-world scenes and real-world backgrounds [4] from the SRDR dataset described in Chapter 6. Rankings for the localization task are from the experiments conducted in Chapter 5. Despite solving different tasks, using different networks, and different datasets, it is more favorable selecting more complex patterned textures when using DR. . . . 179

8.1 Results from an object detection task using unconditional patch-based textures generated from real-world images. . . . 200

8.2 Results from an object segmentation task using unconditional patch-based textures generated from real-world images. . . . 201

8.3 Results from an object detection task using conditional patch-based textures generated from real-world images from Texture B. . . . 203

8.4 Results from an object segmentation task using conditional patch-based textures generated from real-world images from Texture B. . . . 203

8.5 Results from an object detection task across all object classes comparing baseline DR methods, Unconditional, and Conditional application of textures using our method. . . . 205

8.6 Results from an object segmentation task across all object classes comparing baseline DR methods, Unconditional, and Conditional application of textures using our method. . . . 206

8.7 Results for the object detection task when varying the patch size. . . . 208

8.8 Results for the object segmentation task when varying the patch size. . . . 208

8.9 Results for the object detection task when varying the patch size across all object classes. . . . 209

8.10 Results for the object segmentation task when varying the patch size across all object classes. 210
8.11 Results from the object detection task using the unconditional GAN-based texture application. 212
8.12 Results from the object segmentation task using the unconditional GAN-based texture application. 213
8.13 Results from the object detection task using the conditional GAN-based texture application. 216
8.14 Results from the object segmentation task using the conditional GAN-based texture application. 216

Glossary

Accuracy	Accuracy is defined as how well a given algorithm performs. For example in a classification task, accuracy is the number of correct predictions over the total number of examples.
Adam	A commonly used type of optimizer based on gradient descent.
Backpropagation	The leading algorithm used to train neural networks.
Baseline	An approach or model used as a reference point in comparisons.
Batch Size	The number of examples provided as the input to a model in a single iteration or one update of the model's weights during the training process.
Bounding Box	A set of coordinates (x, y) in a 2D image that forms a rectangle surrounding an object of interest. Bounding boxes are commonly used in object detection tasks.

Clustering	Clustering is the process of grouping a set of related examples. This approach is typically used in unsupervised learning.
Convolutional Neural Network	A particular type of neural network mainly used in visual recognition.
Deep Learning	A subset of Machine Learning that is based on methods utilizing neural networks composed of many layers for solving various tasks. The learning process can involve supervised, unsupervised, semi-supervised, or reinforcement learning.
Denoising	An approach in which a model removes noise from a dataset.
Dimensionality Reduction	Reducing the number of dimensions used to represent a particular feature in a feature vector.
Discriminator	A component in a Generative Adversarial Network that attempts to distinguish between real (ground truth) and fake samples.
Domain Adaptation	Domain adaptation is the study concerned with a mismatch between data distributions used to train and evaluate algorithms. An example of a mismatch in data distributions is using synthetic data to train an algorithm and using the algorithm on real-world data.

Domain Randomization	Domain Randomization is the process of generating data using a renderer or simulator, whereby each rendering or simulator parameter is randomized using a statistical data distribution. The randomization parameters such as textures of objects, their positions, lighting, or position of a camera control the appearance of the generated data. The process helps models trained on domain randomized data to transfer to real-world images.
Domain Shift	A divergence in an algorithm's training data distribution and the test data distribution. For example, if our training data distribution consisted of digits with a black background, and our test data distribution consisted of digits with coloured backgrounds.
Environment	Related to Reinforcement Learning: it is the the world that an agent interacts and observes the current state. For example, the environment can be a physical room in the real world, or a virtual world such as the game Go.
Example	A single sample from a dataset which may contain an associated label depending on the dataset.

False Negative	A false negative is when a model incorrectly predicts the negative class. For example, in an object detection task, the model fails to predict the presence of an object of interest, despite the object being visible.
False Positive	A false positive is when the model incorrectly predicts the positive class. For example, in an object detection task, the model predicts that an object is detected, despite the object not being visible.
Feature	A feature is an input variable used for making predictions in machine learning and deep learning. For example, features could be objects, edges, or shapes; typically a numerical representation.
Feature Extraction	The process of accessing intermediate layers in a neural network to access feature representations of a given input.
Feature Space	A feature space is the space associated with a set of feature vectors.
Feature Vector	A feature vector is a numerical representation in the form of a list which contains information from an input to a neural network.

Fine-Tuning	The process of reusing an existing pre-trained model and re-training using a different training dataset. The process of modifying the weights of the pre-trained model is referred to as fine-tuning.
Fully Connected Neural Network	A fully connected neural network is where every node in the current layer of a neural network, is connected to the subsequent layer.
Generalizability	How accurately a particular algorithm performs when applied to unseen test data distribution.
Generative Adversarial Network	A type of neural network which can be used to generate novel data. The Generative Adversarial Network typically uses two main components: a generator and a discriminator. The generator is tasked with creating new data, while the discriminator determines whether the data is from the generator or the ground truth.
Generative Model	A type of model that typically generates new data from a given training dataset. An example of a generative model is a Generative Adversarial Network.
Generator	A component of a Generative Adversarial Network, which is tasked with creating new data.
Ground Truth	The true value, label, or class. The ground truth can be defined as the correct answer or truth.

Hyperparameter	The parameters used to control the learning process. Some examples of hyperparameters are the learning rate, the batch size, the loss function, or the optimizer.
Illumination	Related to generating synthetic scenes, it is approximating or simulating lighting in a scene.
Image Classification	A visual recognition task to label a single object class in an image. For example, predicting the object class 'mug' when a mug is visible in an image.
Input Layer	The first layer in a neural network.
Interpretability	The ability to understand the reasoning for some behavior using a machine learning or deep learning model.
Intersection-over-Union	A metric typically used in object detection and segmentation tasks for measuring the accuracy for some given model. For example, using bounding boxes, it is the ratio between the overlapping area and the total area from the ground truth and model predictions.
Label	A label is the ground truth or correct result for some sample in a dataset. For example, in a dataset containing images of handwritten digits from $0-9$, an image containing a handwritten number 1 has the label 1.

Layer	A collection of connected nodes which takes multiple inputs and performs a computation to produce an output.
Learning Rate	The learning rate is used in the learning process to train a model using gradient descent. The learning rate determines the step size during each iteration of the learning process.
Loss	The loss quantifies how far a model's prediction is from the ground truth label. The loss is expressed as a function, for example using mean squred error for a loss.
Machine Learning	Machine learning is the study of learning from experience using data-driven algorithms.
Mean Squared Error	Mean Squared Error is a computation that measures the average of the squares of the difference between a model's predicted output and ground truth label.
Mesh	Related to generating synthetic scenes, it is the definition of the shape of a 3D object comprising of vertices, edges, and faces.
Model	A model is a machine learning or deep learning system that can be trained to solve a given task. For example, a trained object detection system to detect boxes is referred to as a trained object detection model.

Model Capacity	A model's capacity is related to the complexity of problems the model can solve. A high capacity model that can solve complex problems contains a larger number of paramters.
Neural Network	A model composed of multiple layers which serves as the backbone for many deep learning approaches.
Noise	Noise is typically any signal or information in the dataset that alters the original data. For example, static noise in images or incorrect labels for objects in images.
Non-Complex Textures	A non-complex texture can be viewed as textures that contain a flat shade of color such as the color red, green, or blue. Non-complex textures are devoid of any patterns or additional noise.
Object Class	A name or label from a set of possible values that describes a particular sample. For example, mugs, bowls, cereal boxes are possible object classes for an image classification task.
Object Detection	A visual recognition task to locate a particular object(s) class(es) in images or videos and classify them. Locating an object of interest involves predicting corners of a rectangle called a bounding box where the object lies and a class label for each bounding box.

Object Localization	A visual recognition task to locate a particular object class by predicting their position in Cartesian space (x, y, z) from images or videos.
Object Segmentation	A visual recognition task to locate a particular object(s) class(es) in images or videos and classify them at a pixel level. Contrary to object detection, which uses a bounding box, segmentation locates and labels individual pixels of an object of interest.
Object Textures	Object Textures are the visual appearance of an object. A texture map is an image that contains information about the visual appearance of a particular object, subsequently applied to the surface of a 3D model.
Optimizer	An optimizer is some implementation of a gradient descent algorithm which is used to tune the parameters of a model.
Output Layer	The final layer in a neural network.
Overfitting	Overfitting is the event of creating a model that closely or exactly matching a set of training data. Closely matching the training set means a model will be less effective or fail to generalize to unseen data leading to lower accuracy.
Photorealistic Images	Synthetic images that closely resemble the fidelity or are indistinguishable from reality.

Policy	Related to Reinforcement Learning: A policy is the mapping from the observable state of an environment to possible actions that may be taken.
Pose Estimation	A task that involves locating particular objects of interest and predicting their position and orientation relative to some coordinate system in images or videos (also referred to as 6D pose estimation).
Primitive Objects	Primitive objects are simple geometric shapes such as a sphere, cube, cylinder, pyramid, or cone.
Real-world Data	Real-world data is any data obtained by direct measurement in reality. An example is using a camera to capture images in the real world.
Reinforcement Learning	Reinforcement learning is an approach to training algorithms using an agent to interact in an environment and maximize a reward signal. An example of this can be a robotic arm (the agent) gripping an object of interest, where a successful grip is rewarded. Over time the agent will learn to grip the object of interest successfully.
Renderer	A renderer is used in the process to generate or synthesize images from a set of parameters. The parameters defining a virtual scene include a camera viewpoint, object models, positions, textures, and lighting to create a synthetic image.

Semi-Supervised Learning	Semi-supervised learning is the process of learning with a combination of labeled and unlabeled data.
Sim-to-Real	Leveraging techniques to transfer knowledge from algorithms trained in simulation to generalize to the real world. An example would be training to detect household objects in simulation and deploying the algorithm in the real world.
Simulator	A simulator is a computer program that we may use to design an environment that accurately represents or simulates reality. We can model specific properties such as visual appearance or physics. Once an environment is set up, synthetic data can be gathered using the simulator.
Source Domain	The data distribution that an algorithm learns from.
Supervised Learning	Supervised learning uses labeled or annotated examples to train an algorithm. For example, training an algorithm to segment boxes utilizing a dataset containing images of boxes and pixel-level annotations for the box's positions in the pictures.
Synthetic Data	Synthetic data is artificial data typically computer generated and produced using a pre-defined data distribution. Synthetic data is oftentimes used to simulate data obtained in the real-world.

Target Domain	The data distribution that an algorithm is evaluated on.
Test Dataset	The data used to evaluate the performance of a particular algorithm. Unlike the validation set, this dataset is unseen during the training and design stages, meaning it is not used to train an algorithm.
Training Dataset	The data used to train a particular algorithm.
Transfer Learning	Transfer learning is re-purposing algorithms trained for a particular task and set of data to improve generalization on a different task and set of data. An example would be using an image classifier to classify food and using the learned knowledge to classify bottles.
True Negative	A true negative is when a model correctly predicts the negative class. For example, in an object detection task, the model correctly predicts that an object of interest is not visible.
True Positive	A true positive is when a model correctly predicts the positive class. For example, in an object detection task, the model correctly predicts that an object of interest is visible.
Unsupervised Learning	Unsupervised learning is the process of learning without explicit labels or annotations to train an algorithm. An example of this would be deblurring or denoising algorithms.

Visual Recognition	The ability to identify or locate particular object(s) of interest in an image. Examples of visual recognition tasks are image classification, localization, detection, or segmentation.
Wasserstein Distance	A distance measure used to measure the distance between two probability distributions.

Acronyms

BRIEF Binary Robust Independent Elementary Features.

CGAN Conditional Generative Adversarial Network.

CNN Convolutional Neural Network.

DA Domain Adaptation.

DL Deep Learning.

DNN Deep Neural Network.

FID Fréchet Inception Distance.

GPU Graphics Processing Unit.

JSD Jensen-Shannon Divergence.

KL Kullback–Leibler.

ML Machine Learning.

MoCap Motion-Capture.

ORB Oriented FAST and rotated BRIEF.

RL Reinforcement Learning.

SGD Stochastic Gradient Descent.

SIFT Scale-Invariant Feature Transform.

SURF Speeded-Up Robust Features.

SVM Support Vector Machine.

TL Transfer Learning.

VAE Variational Auto-Encoder.

VOC Visual Object Classes.

WD Wasserstein Distance.

WGAN Wasserstein Generative Adversarial Network.

WGAN-GP Wasserstein Generative Adversarial Network - Gradient Penalty.

Chapter 1

Introduction

Visual recognition plays a vital role in how we perceive and understand the world around us. Traditional advances in computer vision typically relied on hand-crafted techniques to extract important features that algorithms could operate on. More recently, a combination of computational power, access to vast amounts of data, and advances in algorithms led to solving tasks that once seemed unachievable. Such tasks that have seen extraordinary progress towards solving include object recognition, natural language processing, and robotic perception and control [108, 111, 182].

These advances have been made possible through the use of Deep Neural Networks (DNNs) or Deep Learning (DL), a subset of machine learning (ML). Like ML, this multi-stage process attempts to predict an outcome given an appropriate selection of a loss function and optimizing the parameters of a given DL model on a set of inputs using an optimization algorithm such as Adam [97].

DNNs, particularly Convolutional Neural Networks (CNNs) [109], are powerful tools for solving computer vision tasks such as Object Detection [71, 165], semantic/instance Object Segmentation [72, 174], Image Classification [103, 109], or Pose Estimation [205, 228]. In 2012, Krizhevsky, Sutskever, and Hinton [103] used a CNN to classify images from a large-scale dataset,

that is, to predict what is in a given image, and outperformed the previous state-of-the-art approach by nearly halving the error rate. CNNs usage continued to grow within visual recognition problems and started nearing near-human performance accuracy on image recognition and scene understanding benchmarks [38, 71, 72, 178, 196].

These advances in visual scene understanding using DL solidified the tool for solving such complex problems. However, DL is not without its drawbacks. This thesis aims to address three key points in which using DL is challenging [126]:

1. Reliance on an enormous amount of labeled data
2. Solving problems where data is limited, not easily obtainable, or practical to collect
3. Difficulty in generalizing to unseen data

The first is that these approaches typically rely on enormous annotated data to solve vision tasks. Take, for example, ImageNet [178], which is an image-based dataset comprising of over 14 million real-world photos to help solve classification problems. In 2012, AlexNet [40] managed to win the ImageNet Challenge with a top-5 error rate of 15.3%. More recent models such as ResNet [71], brought that error down to sub 4%. Despite the performance of such models on ImageNet, the models still required a large amount of real-world data to solve the classification task. Using complex DL models with a small amount of data remains an open challenge. It is yet to achieve similar performance when comparing the same classification task on high-capacity models with vast quantities of data [17].

The second point is that access to such a large amount of data to solve computer vision or robotics tasks may not be easily accessible or practical to gather. Take, for example, a kitchen countertop in a typical household, where we would like to localize, detect or segment a mug from a given image. In the localization problem, we are concerned with predicting coordinates for where a particular mug is in a given image; for example, the center of the mug is located at pixel

position (x, y). A variation of this problem is the detection task. We want to draw a rectangular box surrounding the mug, where the rectangle's corners specify the object's position in a given image. Finally, the segmentation task involves 'painting' over the image region that contains our mug at a pixel level, meaning assigning a label of 'mug' to each pixel in a photo containing the object.

Several challenges arise when attempting to solve these problems. For example, is the mug hidden behind a cereal box? Cluttered countertops with many objects can lead to an increase in occlusion, meaning parts of our mug may be partially visible in a given image. Does our mug have a photo of our beloved pet on it? In which case, our system may misclassify the mug as a cat. Is the mug what we conceive shaped like a cylinder with a handle, or could it be heart-shaped? These are some of the examples in which a model may fail to detect our mug. Indeed, different times of the day or different viewpoints would also change the visual appearance of what we are trying to detect, making it increasingly more challenging to capture all possibilities of what our object would look like in different environments.

A typical DL solution would be to include hundreds of thousands of images of our mug under various illumination conditions, camera angles, different mug textures, mug shapes, and varying degrees of clutter to have a robust and generalizable model capable of detecting our desired object. These scenarios are challenging to acquire in reality. However, they are typically the cases in which such systems fail to perform as anticipated. Our reality is long-tailed, and it is the tail-end of our training dataset distribution that matters, leading to the third pitfall of generalizability [93, 210].

Ideally, our DL model would perform well across various environments and conditions and not only function in our specific kitchen. As discussed in the challenges above, this may not always be the case, and we must devise techniques to overcome these potential difficulties. Our desire to predict future inputs for the above tasks when our new inputs are from a different environment is the study of generalization; given a new set of data that is visually different from our initial kitchen

environment, how well will our model perform? Limited training data in only our kitchen can lead to what is referred to as overfitting on the training set, where our model 'memorizes' the training samples provided and is unable to transfer that knowledge to unseen kitchen environments [93].

1.1 Motivation

Using synthetic data, or computer-generated data is one possible approach to addressing the dataset size, difficulty in gathering, and generalizability concerns when using DL techniques. Synthetic Data is the production of data generated using computer simulators based on a set of parameters. Going back to our kitchen example, we could use a simulator to define our household objects' positions around a kitchen countertop, the position of light sources and their intensity, our viewpoint, and how the objects appear visually - such as a shiny red mug. The simulator would handle properties such as physical interactions between objects. If we wanted to generate photos or images of what our defined scene in the simulator looks like, we would use a renderer to create an image based on the above information. We can think of the simulator as handling all the scene modeling, and the renderer processes the information to visualize.

The initial time-sink of using the simulator for creating an environment, designing the types of scenes and object layouts we would like, and running the simulator, dramatically outweighs the cost when collecting the data in the real world. Because we define our scenes using a simulator, we already have access to labeled information regarding where and what the objects are, among other helpful information that can be automatically generated alongside our images of kitchen scenes. This data generation approach helps tackle the data constraint, as we can generate millions of labeled data at a fraction of the time it would take in the real world. We would also be able to more easily create scenes at the tail-end of the distribution, as we have greater control of what we produce using the simulator.

The above reasons make using synthetic data appealing to train DL models to solve Ob-

ject Localization, object detection, or semantic segmentation to solve scene understanding in the real world. However, despite the increasingly more realistic rendering techniques to generate images, these are often imperfect approximations of the real world [29, 30, 101, 163, 167]. In some scenarios, computer-generated images can differ visually due to the textures used. For example, very rarely does a real-world environment look spotless and would most likely include dirt, dust, or scratches over time [121]. Imperfect approximations of the illumination in a scene may cast unnaturally looking strong shadows that would not reflect reality [163]. Creating highly photo-realistic images requires significant effort from professional artists towards creating assets such as accurate object geometries, believable textures, and state-of-the-art illumination approximation techniques to generate more compelling synthetic images [74, 172, 244].

The differences in geometry, object textures, and illumination culminate in a data distribution that does not accurately represent what is present in the real world. This difference between synthetic and real-world data is referred to as a Domain Shift in data distributions. Our synthetically generated data differs from our real-world data distribution.

Due to the greater control of data generation and the time it takes to generate large quantities of data, it would be ideal to use a given DNN model trained using synthetic images and function when we deploy our model to perform our task on real-world images. In this thesis, we are interested in addressing the difficulties of transferring learning from synthetic data to be applied to real-world data for visual scene understanding.

One approach to transferring from synthetic to real that has recently gained traction is a method called Domain Randomization (DR) [180, 202]. Domain Randomization is an approach to synthesizing synthetic data that can generalize to the real world. The intuition behind the approach is to generate variability in the synthetic dataset, such that the real-world would appear as some variation from the synthetic data. To produce this data, we must randomize several rendering or simulator parameters. For example, if we wanted to generate several images of household objects on a table, we would randomize the number of objects and their positions on the table. We would

also randomize what the objects look like, such as different colored mugs or different patterns for the table. We would also randomize the camera's position to see the objects from various angles and locations and change the position and intensity of the lights. Putting this together, we have a system that can generate plenty of randomized synthetic data.

The approach seems straightforward to use; however, it is not clear how we would approach selecting and randomizing the parameters. Would using patterns work better on the table or the mug? Would a particular set of random backgrounds result in higher performance? How should we randomize the poses of the objects on the table? Furthermore, it is ambiguous if the approach would yield similar results across different tasks. For example, would a particular set of randomization techniques lead to higher object localization performance than a segmentation task? Lastly, can we improve the Domain randomization process to increase performance for some given task? This thesis addresses the difficulties of transferring learning from synthetic data to real-world data via novel Domain Randomization techniques for solving tasks in visual scene understanding.

1.2 Contributions

This thesis presents several novel approaches to transfer learning from synthetic data to real-world applications for solving object localization, detection, and segmentation tasks. The first work of the thesis introduces a novel approach to measuring the difference between two different data distributions when using realistic and Domain Randomized synthetic images. The proposed method allows us to rank commonly used domain randomization texture techniques and find that the ranking is reflected in the performance of an object localization task. This chapter presents a measurable way of selecting more appropriate textures for synthesizing DR images and bridges the gap between synthetic-to-real transfer without the need for task-based networks.

The second work presents a large domain randomized dataset containing 291K frames using realistic household objects that are widely used in robotics and vision benchmarking [23]. We

expand upon the dataset by [67] by taking images from the dataset and generating DR versions for each texture type in current literature and in 5 unique environments [4, 74, 132, 244] with varying scene complexity. To our knowledge, this is the first dataset to contain Domain Randomized data using the most commonly applied texture randomization techniques, matched real-world (when combined with the YCB-M dataset) and real-textured synthetic data. The SRDR dataset enables researchers to perform exhaustive comparisons, evaluation, and training using DR techniques in current literature, particularly in cross-domain DR synthetic-to-real settings. Finally, we build upon the tools from To et al. [200] to enable DR for existing labeled real-world datasets and provide new tools for researchers to create DR versions of their own real-world datasets.

The third work is a comprehensive study to evaluate DR's generalizability and robustness in Sim-to-Real settings by randomizing poses, textures, and backgrounds in cluttered and occluded scenes [67] using realistic household objects [23] for object detection and semantic segmentation. We find that the performance ranking is largely similar across the two tasks when evaluating models trained on DR synthetic data and evaluating on real-world data, indicating DR performs similarly across multiple tasks. Based on our findings, we propose researchers generating new DR data to focus on the diversity of object of interest poses that would more likely appear in the target dataset. We also recommend using textures containing complex patterns such as Checkerboard or Zig Zag and using diverse backgrounds from the real-world or photorealistic synthetic backgrounds.

Our final work presents a novel method for conditionally generating and applying DR textures by using patches from real-world images that outperform the most commonly used DR texture randomization method from 13.157 AP to 21.287 AP and 8.950 AP to 19.481 AP in object detection and semantic segmentation tasks, respectively. We also outperform the best DR texture randomization method from 16.354 AP to 21.287 AP and 12.771 AP to 19.481 AP on the same tasks. Readily available real-world images [134] means the patch-based approach is fast, easy to execute, and does not require decisions on manually defined texture generation routines to artificially create complex textures to produce DR images. We also propose a further improvement to

address low texture diversity when using a limited number of real-world images. We do so using a conditional GAN-based texture generator trained on image patches to increase the texture diversity and outperform the most commonly applied DR texture randomization method from 13.157 AP to 20.287 AP and 8.950 AP to 17.636 AP in object detection and semantic segmentation tasks, respectively. This approach also outperforms the best DR texture randomization method for object detection and segmentation tasks from 16.354 AP to 20.287 AP and 12.771 AP to 17.636 AP, respectively.

1.3 Thesis Outline

The thesis links the four core works above to further our understanding of generating more applicable texture synthesis techniques for domain randomization. The improved understanding of textures within Domain Randomization leads to novel methods towards the improved application and synthesis of textures for domain randomized synthetic data to solve visual scene understanding tasks.

- Chapter 2 provides essential knowledge regarding synthetic data generation, the importance of data within ML, and the differences between classical and DL methods for solving visual scene understanding tasks.
- Chapter 3 is a general literature review for using synthetic data for visual scene understanding. This chapter presents more detail regarding the motivation and challenges for using synthetic data when using DL techniques. Transfer learning, domain adaptation, and details regarding domain shift is discussed. An overview of existing literature and methods for transferring the synthetic to the real world is covered, presenting approaches using solely synthetic data, a combination of synthetic and real-world data, refined, and procedurally generated synthetic data.

- Chapter 4 is a literature review of domain randomization, which presents the general algorithm for synthetic image synthesis, its applications, and the drawbacks of using this approach when transferring to the real world.

- Chapter 5 is the first body of work that opens with the motivation for the proposed approach. The method and implementation details follow, with details regarding the data generation process and experiments following that. The chapter concludes with a discussion of the results and the conclusion of our findings.

- Chapter 6 presents our second body of work, the SRDR dataset. The chapter opens with motivations for creating the dataset, comparisons with existing domain randomized datasets, and the data generation methods. The chapter includes dataset statistics and concludes with possible use cases for training, testing, and evaluating transfer from synthetic to real, particularly in cross-domain settings.

- Chapter 7 is the third body of work, which presents the comprehensive study to evaluate DR's generalizability and robustness in sim-to-real settings. The chapter presents the problem definition, followed by detailed experiments surrounding randomization of poses, textures, and backgrounds in cluttered and occluded scenes with realistic household objects. The chapter concludes with a discussion of our findings and proposals for researchers generating new domain randomized data.

- Chapter 8 is our final body of work, which presents a novel method for conditionally generating and applying domain randomized textures. The chapter opens with the motivation for the work, followed by the existing techniques currently used. The method section details implementational and data generation details for the upcoming experiments. Following this, we detail the experiments conducted and conclude the chapter with our key findings.

- We conclude the thesis with a summary of the work presented, the shortcomings, and highlighting the proposed work's key contributions and future directions.

1.4 Publications

- M. Ani, H. Basevi and A. Leonardis, "Quantifying the Use of Domain Randomization," 2020 25th International Conference on Pattern Recognition (ICPR), 2021, pp. 6128-6135. – Related to Chapter 5 [5].

Chapter 2

Background

2.1 Synthetic Data Generation

We navigate the world around us in three-dimensions without paying close attention to the intricacies of how light passes through complex materials around us. Elaborate patterns on uniquely shaped objects and indirect shadows cast leave us oblivious to the intricate physics that occurs when we observe the world around us. For decades, scientists have made progress in developing methods for re-creating the world around us using computers. Utilizing a combination of mathematical techniques and computer programming, we can represent our three-dimensional world on our machines. We see these techniques used in various media such as films, video games, or digital art. Computer graphics has allowed us to create objects that appear, move, and reflect light off their surfaces, similarly to how we would observe them using our eyes and project them onto an image plane for us to see on our displays.

To create the synthetic image in Figure 2.3, we must first describe the scene in the form of parameters passed through the rendering process. These parameters include the 3D geometry of the objects in the scene, the objects' positions, lighting information (number of lights and properties),

textures of the objects (what the object looks like), material properties (describing how a surface would reflect light), and camera information (where we view the scene). Figure 2.1 depicts the formation of a recognizable object, starting with 3D coordinates.

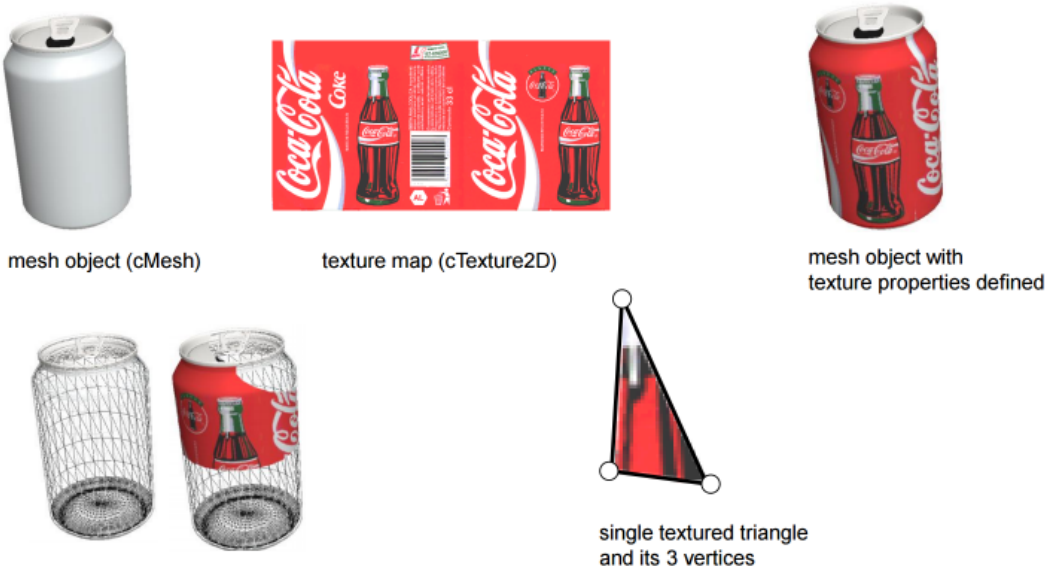


Figure 2.1: The illustration is taken from [27], demonstrating the application of textures on a 3D object, given a 3D object with variable name cMesh, and texture map with variable name cTexture2D. The 3D can is composed of several thousands of vertices and triangles, as shown in the bottom left. A texture map is an image of fixed size that contains information about an object's visual appearance. This texture is stretched over the 3D mesh object to produce a texturized object, as shown in the top right.

Figure 2.2 is a simple illustration of the process to move from a set of vertices or 3D points that define the geometry of an object, placement of the camera, and the final textured and illuminated scene. We have all the information required to automatically label each part of the generated image because of this process, meaning we now have a synthetic image and an associated label defining what and where objects are in the image.

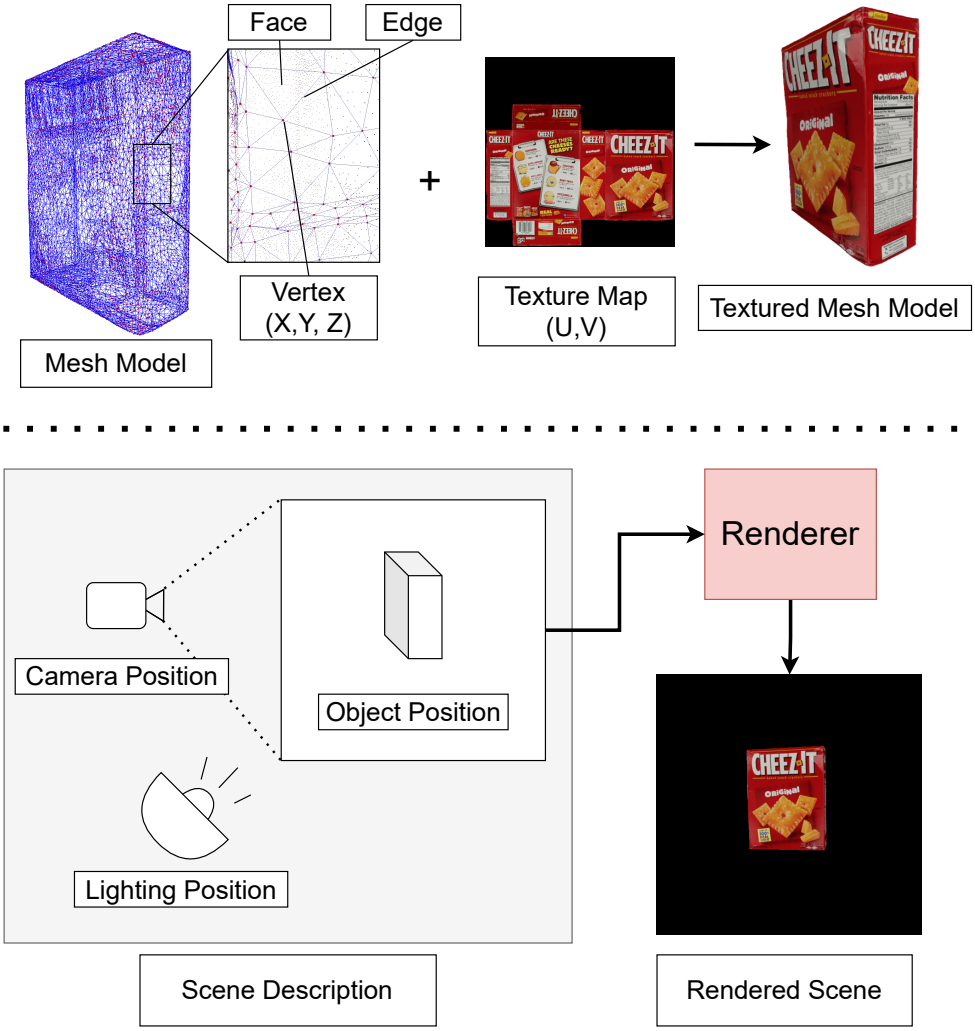
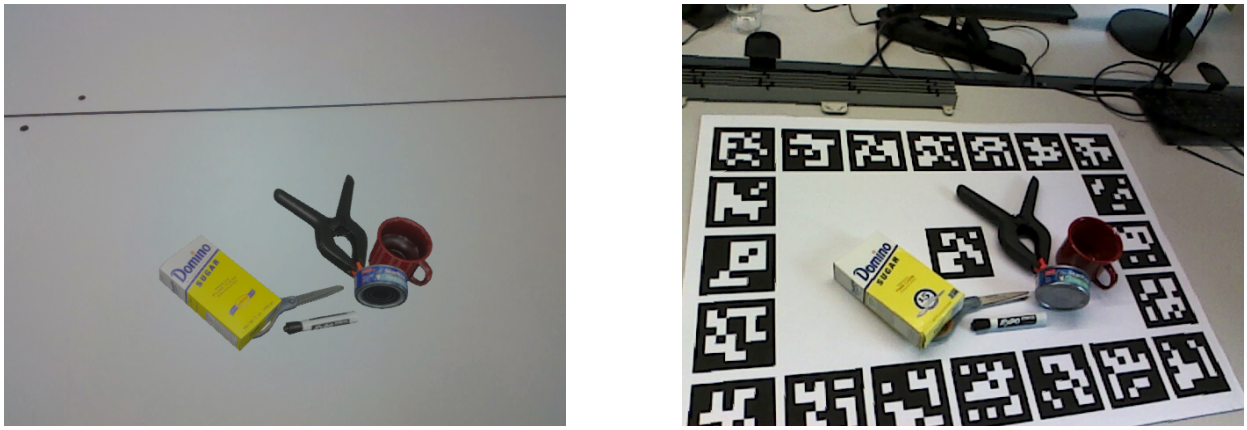


Figure 2.2: Figure illustrating a sample mesh model, texture map, and textured mesh model on top, and the process for generating a synthetic scene using a renderer on the bottom. Each mesh is defined as a set of vertices, edges, and faces which describe the 3D shape of an object. A vertex is a point in 3D space (X, Y, Z). Each connection between two vertices forms an edge, and each set of closed edges comprises a face as shown on top. A collection of texture coordinates defines a mapping from a 2D texture map to the 3D shape, where each texture coordinate is associated with a vertex on the 3D model. The textured model is seen in the top right. Rendering a synthetic scene requires a description of where the camera is oriented, the lighting, and the position of the object. The object is transformed to the camera coordinate system via Rotation and Translation. The final rendered scene is in the bottom right.

2.1.1 The Uncanny Valley - Addressing Realism in Synthetic Data

Despite the incredible progress in advancing the state of computer graphics, we can still discern the difference between real and synthetic (computer-generated) images. Looking at Figure 2.3, we can see differences in how the textures appear compared to the real-world version. Lighting is still quite challenging to approximate, and we observe that in Figure 2.3. For example, the mug appears more brightly lit from the top, or the tuna can seems duller than the real-world version. Several factors such as the material properties used to describe the object, the number and intensity of the lights in the scene, and the shading or illumination models - which control how the lighting behaves when reflecting off of surfaces - can influence the final synthetic image.



(a) Synthetic Image

(b) Real-World Image

Figure 2.3: Figure illustrating a synthetic image with a real-world background on the left and the real-world version on the right from the YCB-M dataset [67]. To generate the scene on the left, we use information regarding the position and orientation of the camera, positions of illumination sources, 3D object meshes, their textures, and their positions are needed to match the real-world scene on the right.

The innate complexity of modeling lighting means utilizing several different methods of approximating how light interacts with objects in a scene. The simplest of these models operate at the local level, meaning once a light source reaches the surface of an object and reflects off of

it, no additional estimates are made to account for the light interacting with objects around it. Examples include Gouraud, Phong, or Blinn-Phong shading, which are computationally inexpensive methods for approximating local illumination from a single point source to an object [12, 65, 153]. More complex models for defining illumination exists, for example, ray tracing is a method for estimating global illumination, where computations are made for light bouncing throughout the entire scene. Due to an increase in the number of light rays bouncing around the whole scene, the computational cost is considerably more significant than the simpler local illumination models.

Synthetic images can appear more realistic when deploying more complex lighting approximations; however, the trade-off for more realistic scenes such as ray-traced face models in Figure 2.4 is an increase in the time needed to generate a scene. The more precise the approximation, the longer it takes to create a single image.

Figure 2.4: Ray-traced illumination used to generate two photorealistic human faces using Unreal Engine's MetaHuman [50]. While the process of approximating illumination is more computationally expensive, the results are more photorealistic. Figure taken from [50].

2.2 Importance of Data

At the heart of a significant amount of deep learning literature lies data. In computer vision, large-scale image datasets are pivotal to advancing the field, alongside algorithmic development and increased computational power in the forms of Graphics Processing Units (GPUs) [179]. The ideal scenario entails gathering a large-scale dataset that contains a range of data that is well-represented and reflected in our world. In reality, it is often challenging to put together the desired dataset.

Imagine a scenario where we would like our self-driving car to autonomously navigate from point A to point B without human intervention. Scene understanding plays a significant role in determining what actions the machine should take. Recognizing stop signs, traffic lights, lane markings, or pedestrians and vehicles is essential to predicting a safe maneuver. However, a system trained on images captured from multiple angles in a car in sunny California will not behave similarly to a system trained in the rainy UK. It would be extremely time-consuming to capture images from various parts of the world under different lighting conditions and during separate seasons [42, 235]. Additionally, these images have to be manually labeled by humans, denoting different categories for what is visible in the image at the pixel level. The tremendous financial cost and time requirements to accomplish this task leave us questioning if there is a more suitable alternative [151, 193].

We have previously seen that we can use computers to generate scenes such as Figure 2.3, where we see a synthetic approximation of what the real-world objects appear. Also mentioned was the bonus of "free" labels due to the method the scenes are generated, meaning in several seconds, hundreds of images can be generated with annotations of what and where objects are in the scene. This idea can be exploited to train machines to recognize objects using computer-generated or synthetic images, as opposed to real-world images. Millions of images and annotations can be generated at a fraction of the cost, both financially and time, while designing scenes that would more accurately reflect the world around us. In our autonomous car scenario, a single computer can

generate the dataset under the different conditions required to function in different environments, allowing our machine to generalize to unseen conditions.

2.3 Solving Object-Centric Computer Vision Tasks

Computer vision remains an integral part of modern AI systems. We may solve tasks such as pose estimation, segmentation, or detection to locate objects of interest in a given image. For example, Figure 2.5 shows a robotic arm that we could use to interact with several household items on a table. Say we would like to pick up an object and place it to the side; to accomplish this task, the robot must use the camera images that display the scene. These images are piped through a model to provide predictions including what objects are visible and an estimate of their pose in 3D space. The information can be processed to construct a motion plan to move the robotic arm and pick up the desired object [104].

Traditionally, a multi-step approach which involves feature engineering to extract and select relevant features in an image manually before using machine learning (ML) or mathematical models to locate and estimate an object's pose [146]. The most commonly used algorithms for identification in traditional computer vision are Scale Invariant Feature Transform (SIFT), Speeded Up Robust Feature (SURF), Features from Accelerated Segment Test, or Oriented FAST and Rotated BRIEF (ORB) [10, 123, 176]. These algorithms help identify key points in images that are used to define each visible object. More traditional machine learning techniques are used, such as Support Vector Machines (SVM) [35], to make the final prediction of an object. It is vital the features selected best describe a particular object class; this highlights the tedious approach in traditional methods, where an expert computer vision engineer conducts a demanding trial-and-error process to ensure the most appropriate features are selected. This difficulty increases as the number of classes we would like to detect grows, and the manual process must be repeated and verified for each one.

Figure 2.5: A robotic arm that can be used to interact with household objects in an environment. This setup is typical for object-centric tabletop scenes in visual recognition tasks such as object localization, detection, or segmentation. The objects of interest are the camera's main focal point which is attached to the robotic arm.

Deep learning (DL), a subset of machine learning that has more recently used Neural Networks with backpropagation, has existed for several years [69, 119, 182]. While the current state-of-art methods build upon years of extensive research, it is a combination of access to large amounts of labeled data, algorithmic advances in the field, and increased computational power propelled into what it is today. In a typical deep learning approach, the process involves gathering labeled data, forming a hypothesis about possible patterns in a scene, and validating the hypothesis by comparing the algorithm's prediction with the real solution [63, 182].

Unlike traditional methods where we have tangible reasons for selecting features, DL's black-box nature makes it difficult to interpret why a network used a particular feature. A subfield exists, which explores interpretability in DL methods, which shows some promising strides taken towards understanding why a particular outcome occurred [28, 239].

2.3.1 Classical Approaches

Classical approaches to solving object-centric computer vision tasks have relied mainly on expert experience from a computer vision engineer to use hand-crafted techniques [145]. The pipeline typically involves using established computer vision techniques to encode various features in an image. These features are what the practitioner would deem interesting for a particular problem, such as edges or corners. This step of feature extraction may involve using feature descriptors such as Scale-Invariant Feature Transform (SIFT) [123], Speeded-Up Robust Features (SURF) [10], or Binary Robust Independent Elementary Features (BRIEF) [24], which help extract descriptive keypoints from an image [145].

For example, in solving an object detection task, our goal is to detect where an object appears in a photo. We first extract keypoints from training images using feature descriptors, representing relevant features in an image. An example of this is in Figure 2.6. Keypoints alone would not solve the problem; these methods usually couple with traditional machine learning (ML)

techniques such as nearest neighbors algorithms or Support-Vector-Machines. During test time, the keypoints from the training set would be matched against those from the test images using the nearest neighbor algorithm.

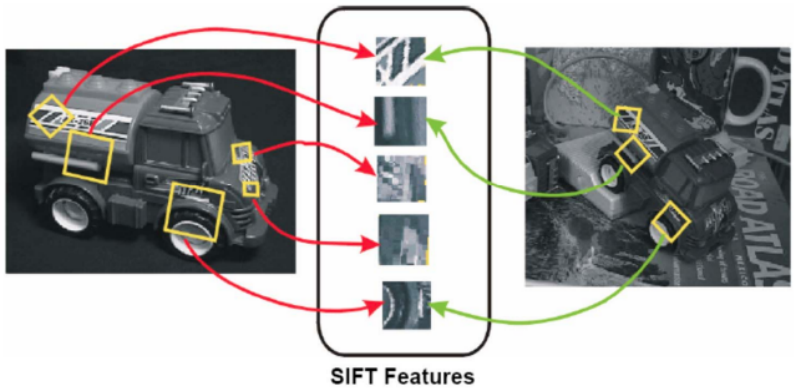


Figure 2.6: SIFT features extracted from the training image on the left, and evaluated on the test image on the right [123].

The process can become increasingly more involved when certain features are not deemed to be suitable, the size of the dataset increases, or the number of classes to detect increases. Increasing manual intervention is needed to discard features that do not appear to match well during evaluation [145].

2.3.2 Deep Learning

Deep learning is a subset of machine learning and has recently been established as the dominant approach for modern AI advancements. The ML or DL algorithm is concerned with the ability to learn from data. The first step to the process is to ensure we have a "well-posed learning problem" [139]. A formal definition of what constitutes as a learning problem is defined by Mitchell [139]:

> "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P,

improves with experience E" - Mitchell [139].

Deep learning follows the same principles and approach as traditional ML. For example, say we would like to recognize street signs from a set of c possible signs:

1. Starting with a task T that we would like to solve. In this case, our task is classifying what street sign is visible in a photo as seen in Figure 2.7.
2. Using experience E, where our experience is defined as a collection of images that are labels containing information about the street sign in the picture.
3. Evaluating the performance P expressed as a percentage of street signs correctly classified.

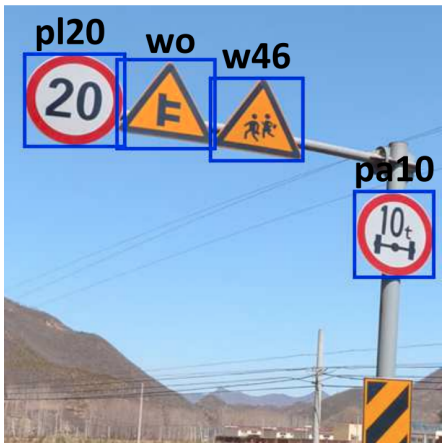


Figure 2.7: Image taken from Zhu et al. [247] showing a street sign classification task, where each sign contains a label describing the visible sign. For example, pl20 denotes a speeding limit sign of 20 Mph [247].

This combination of solving tasks based on some given experience and evaluating its performance is the backbone of modern AI. These algorithms are not restricted to images, and many tasks involving text or real-valued numbers benefit from using these techniques. Tasks involving machine translation is commonly used to translate text from one language to another [19]. Regression is extensively used in robotics for predicting joint positions for robotic manipulation tasks [104].

The experience E, can be seen as the data a model can learn from. Broadly speaking, these set of problems are broken down into 4 main categories:

- Supervised
- Unsupervised
- Semi-Supervised
- Reinforcement Learning

Supervised Learning

Supervised learning is typically a dataset that contains labeled or annotated example. This can be defined as $\{(x_i, y_i)\}_{i=1}^{n}$. Where x_i is a sample from the dataset and y_i is the label of the dataset for n possible examples. In the previous street signs example, this would be a label describing what the street sign is from c possible cases.

Unsupervised Learning

Unsupervised Learning is defined as learning from data without explicit labels. This can be defined as $\{(x_i)\}_{i=1}^{n}$. Where x_i is a sample from the dataset. This type of learning algorithm is typically more suited towards tasks that involve learning features or structure of a given dataset [63]. Examples include image synthesis, or denoising, where estimating the probability distribution of a given dataset of experience E is necessary. We also see this employed in tasks where clustering of data is involved, which is important in several recommendation system tasks [188, 240].

Semi-Supervised Learning

Semi-Supervised Learning involves learning from a combination of labeled and unlabeled examples. Typically, these algorithms rely on a small amount of labeled examples and a much larger set of unlabeled data. The intuition behind this is that despite the large amount of unlabeled examples provided, there still exists some merit in helping the algorithm produce a better estimate or prediction.

Reinforcement Learning (RL)

Reinforcement Learning or RL is an atypical approach, where the algorithms exist within an environment that can perceive states at any given time. For example, assume we would like to build a machine that could play the game Tetris. The machine can take actions such as moving a piece across the screen or rotating the piece. At the start, the machine has no knowledge of what the game is. Different actions can produce different rewards that we assign, such as positive rewards for clearing a line in Tetris or negative rewards for reaching the top of the screen. The goal of a reinforcement learning algorithm is to learn a policy. A policy is a function (similar to a model in supervised learning), that executes an action such as move block and rotate to this position. The policy will try to maximize the expected average reward.

The final piece of the pipeline is selecting the performance metrics used to evaluate how well the algorithm is performing at solving a specific task. It is important to note that depending on the task being solved, the performance measure will differ. For example, when solving an object detection task, our goal is to detect the region of an image that a particular object exists by typically drawing a bounding box around the detected object as seen in Figure 2.8. Our performance criteria would be computing the intersection between our algorithm's predicted bounding box and the actual region of interest. This approach differs from a classification task, where we are interested in how accurate the algorithm is at classifying examples correctly. In this case, the algorithm

measures error as the proportion of examples where a model produces the wrong prediction [63].

Figure 2.8: Example of an object detection task, where a model was tasked to detect certain classes from a given image. Image taken from Redmon et al. [165]. The bounding boxes define the positions of the objects of interest, while the labels “person" and “cat" describe what appears within the region.

There is an emphasis on generalizing to unseen data, as there is a high degree of certainty that the model would not have encountered that exact scenario in the training set. For this reason, we use a separate set of data to evaluate the model’s performance, referred to as the test set. The test dataset is typically representative of scenarios we would like to solve a particular task but is not included in the training set as it would bias the accuracy of the model.

Central to most deep learning algorithms is the structure used to learn from examples, called neural networks. The design lends itself to learn to solve a wide range of tasks dating back to understanding handwritten digits from [109] in the late 1980s. We see these networks applied to many problems from visual scene understanding, machine translation, robotic manipulation, or navigation [139].

When using neural networks to learn from data, we are essentially attempting to approximate some function f. A classic example would be in a classification task, where our optimal classifier $k = f(x)$, mapping an input x to function f to a class k. A neural network, in this case, would learn parameters θ to best approximate the function $k = f(x;\theta)$ [63]. Figure 2.9 shows a neural network with several layers, composing a chain of functions forming $f(x) = f^{(n)}(f^{(n-1)}...(f^{(1)}(x)))$, where n is the number of layers in the network. Deep learning derives from the structures of modern applications of these techniques, where we see the depth of the models increasing from the early days of LeNet-5's five-layer architecture [110], and AlexNet's eight layers [103], to more recent models such as VGG-16/19 [187] and ResNet-50/152 [71] layers.

In summary, it is learning the parameters θ that best approximate a function to solve a task using examples, including discrete or real-valued attributes or a vector containing these. The ability to generalize to unseen examples is core to solving these tasks. In real-world applications, it is implausible that we would encounter the same scenario during our training phase, where we learn to approximate the function.

The approach to deep learning vastly differs from the classical techniques, where additional manual intervention is needed to oversee the training process. Deep learning eliminates the manual feature selection and performs the training in an end-to-end fashion, extracting useful features and patterns as the training process evolves. An example of this is seen in Figure 2.10. Solving computer vision and robotics tasks using NNs resulted in a widespread increase in accuracy and lower error rates, although this comes at the cost of training time and computational resources.

When training a VGG-16 model [187], there are approximately 138 million learnable parameters, which need to be tweaked to approximate a function capable of solving a task. The sheer size of learnable parameters results in the need for graphical-processing units (GPUs) with large amounts of RAM depending on the task at hand [187]. The number of learnable parameters also yields longer training times and consequently results in significant energy use. One of the largest

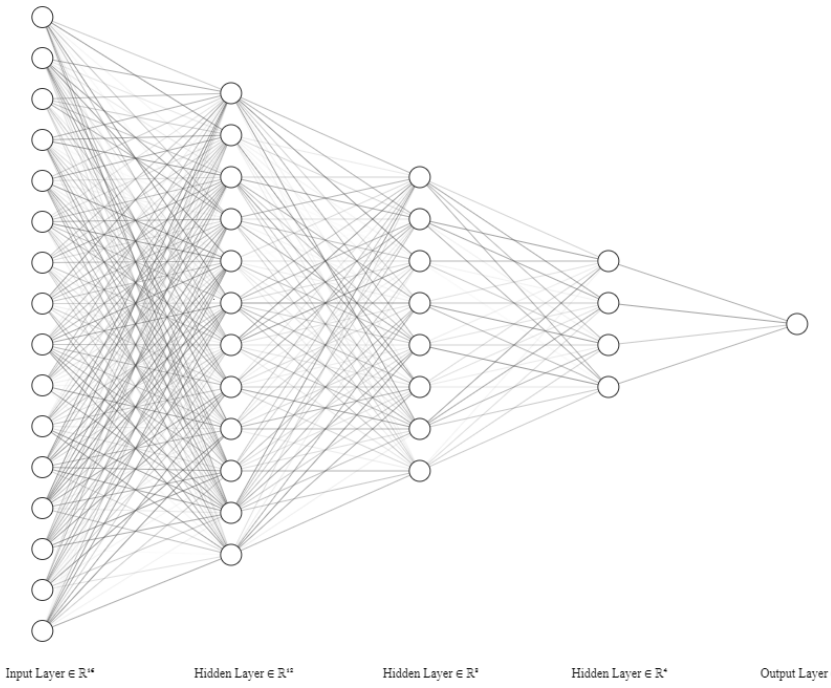


Figure 2.9: Example of a basic feed-forward neural network containing five layers. The input layer is the first layer in the network where the data is first passed through the network. The output layer is the final layer in the network. In this example, each subsequent layer's nodes (represented as circles) are connected to the previous layer's nodes. This structure is called a fully connected neural network.

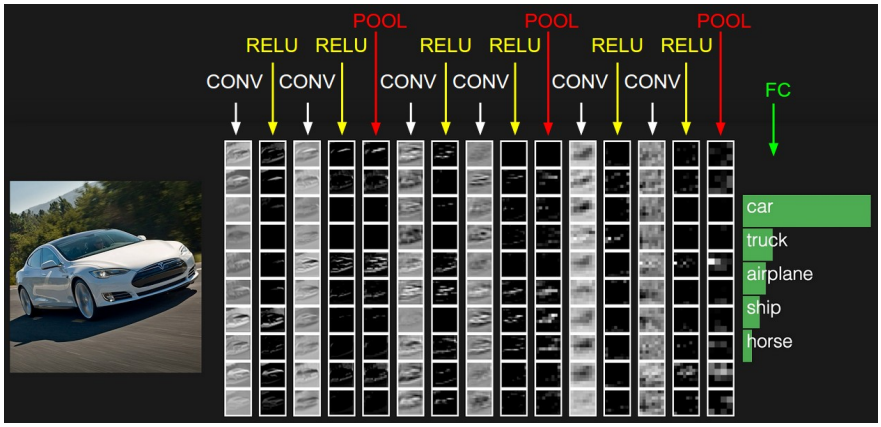


Figure 2.10: An illustration showing intermediate layer representations taken from [86]. Training a network end-to-end using a convolutional neural network, resulting in a trained model capable of producing probabilities of what is visible in the image. The visualizations are the activations of the network, providing some insights into what the network is learning.

language models, GPT-3, contains 175 billion parameters, costing $4.6 million to train and 355 years on a single cloud instance GPU [19, 113]. While the authors of the work are cognizant of the energy use, it is worth highlighting the potential impacts of large-scale single-use models while researching this field [19].

Chapter 3

Learning From Synthetic Data

Given the immense potential of using synthetic data to solve computer vision and robotics tasks, it is crucial to understand the current state. This chapter investigates the existing tools for producing popular synthetic datasets, where these methods are adopted, and future directions in the field. This chapter also highlights key challenges when using synthetic data and how current strategies are attempting to solve this. This chapter opens with a common segmentation task to motivate the importance of synthetic data, in addition to the increase of privacy and reduction in dataset bias (section 3.1). The challenges with using synthetic data are further elaborated on in Section 3.2.1, which focuses on potential drawbacks when relying on this methodology. Broadly, the use of synthetic data is split into four categories: learning purely from synthetic data (both offline and online), using a combination of synthetic and real data, refining synthetic data to appear more realistic, and procedural synthetic data generation. The state-of-the-art methods for solving the above are reviewed in section 3.3.

3.1 Motivation

Task-Based Motivation

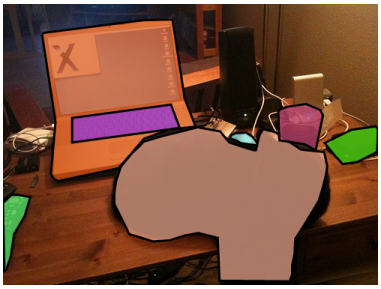
The impetus for advancements in the use of synthetic data stems from a real need for large quantities of high-quality annotated data. One notoriously difficult task to solve that typically requires pixel-level annotations is an instance or semantic segmentation task. Our goal is to individually label each pixel in an image belonging to a specific class of objects or a specific instance of an object. Naturally, the task of manually labeling each pixel can seem daunting, especially if there are many classes in an image and several hundreds of thousands of photos to go through. Despite using image annotation tools to help speed up the labeling process [136, 148, 189, 209, 212], this still takes a tremendous amount of time and resources to accomplish.

As humans, we can make mistakes, which manifests in the annotation process. Figure 3.1 and 3.2 shows several examples from the Microsoft COCO: Common Objects in Context dataset [116]. In Figures 3.1a and 3.1c we see a very similar photo several frames apart, with inconsistent annotations. In Figure 3.1b, we see the keyboard on the laptop correctly annotated, whereas in Figure 3.1d, the annotation for the keyboard class is entirely missing. There is a high chance that these frames were not given to the same person to label, hence the discrepancy. Yet, it does indicate that this type of issue may arise from tackling the monumental task of annotating 123,287 images and 886,284 instances for the train/valid dataset in COCO 2017.

Similarly, Figures 3.2b and 3.2d show instances of books that are completely missing, in addition to the coarse polygonal labels masking multiple instances of the books. This method of labeling may have a non-trivial influence on the training process of Supervised Learning algorithms [190, 236]. This dataset is already widely adopted in the industry, and amending these annotations would likely take a similar amount of time to producing an entirely new one.

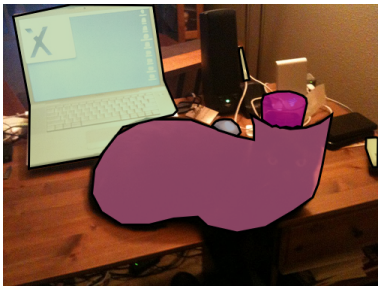
A possible solution to generating higher quality annotations for solving this task would

(a) Image id: 119828



(b) Pixel-level labels

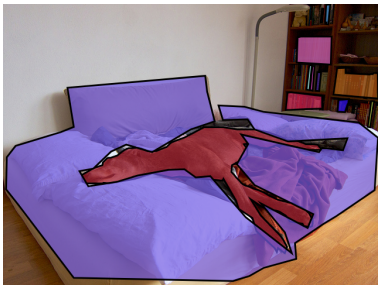
(c) Image id: 209299



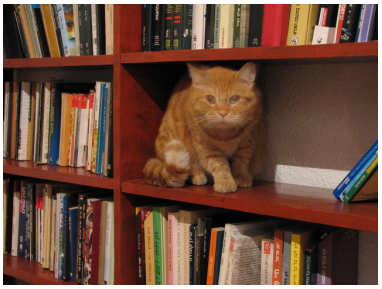
(d) Pixel-level labels

Figure 3.1: Inconsistent labeling between similar frames in the MS-COCO dataset. Figure 3.1d does not contain the class keyboard.

(a) Image id: 117722



(b) Pixel-level labels



(c) Image id: 537213

(d) Pixel-level labels

Figure 3.2: Missing annotations for the book category in the MS-COCO dataset. Instances of the books are not labeled consistently in Figures 3.2b and 3.2d.

be to use synthetic or computer-generated datasets. Doing so allows us to produce millions of images with their labels at a fraction of the time. Because of the image generation process, we may include additional information such as depth maps or stereo vision for free. For example, a sample from the SRDR dataset in Figure 3.3 (presented in chapter 6) displays high-quality dense pixel-level annotations using this methodology. The initial cost of developing and designing the datasets required would greatly outweigh the time spent gathering and annotating real-world data. In contrast, we may never go back to the original RGB images used in the COCO dataset to produce high-quality depth maps or amend the viewpoints. This makes a strong case for leveraging this technique, especially in the age of modern AI where additional data generally aids performance [8].

(a) Synthetic RGB Image

(b) Fine, high-quality, dense pixel annotations for semantic segmentation for the RGB scene

Figure 3.3: Sample data from the SRDR data described in Chapter 6 showing clean semantic segmentation annotations for a synthetic RGB scene.

Privacy and Security

A lesser-discussed theme for the use of synthetic data is the increase in privacy and security of the datasets used. Exceptionally few people may be aware of how their information is being used to train modern AI algorithms. This data ranges from automatic facial recognition systems used in

social media platforms [25], to public datasets used for crowd-counting [77, 242]. The people are generally unaware of their inclusion in such datasets or inadvertently given consent when using social media platforms. Synthetic data allows us to ensure the privacy of individuals by creating data that does not rely on people. Increased awareness of the subject should open up the route to exploring synthetically generated options to preserve privacy. Alternatives to real-world data are actively being researched, such as using synthetic humans for both crowd counting and human pose estimation. The synthetically generated datasets used to train the models are achieving state-of-the-art performance on real data [211, 216].

Bias and Fairness

Synthetic data also addresses the issue of dataset bias from using datasets that are not diverse and indicative of modern society. For example, public face datasets are strongly biased towards White faces, with some nearing 80% being lighter-skinned [76, 105, 120, 130, 135, 213], with the highest being approximately 95% White in the AgeDB dataset [135, 141]. These datasets also have skewed distributions for gender and age, resulting in those that do not fall within the dataset distribution treated unfairly. While researchers are starting to become more aware of the topic, it is unfortunately evident that some damage has been done by companies not considering dataset bias. Several works have investigated the commercial use of products involving facial recognition and concluded additional work is required to address this [20, 164]. Synthetic data would allow careful consideration into the design and creation of these datasets, ensuring the data we would use to train our models would be ethical, fair, and not put anyone at a disadvantage.

3.2 Transfer to Real-World

3.2.1 Difficulty in Transfer

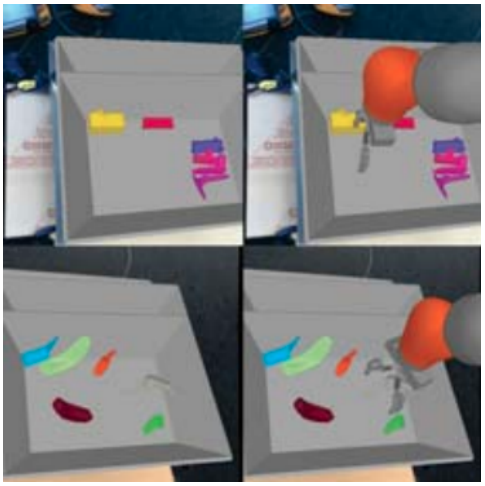
Now that we have discussed some of the benefits of using synthetic data in our training regime, we turn to investigate some of the potential challenges that arise from using it. To begin, when we think about training DL algorithms, we usually assume that our training, validation, and test set would all appear visually similar. In reality, this is oftentimes not the case. For example, Figure 3.4 shows several scenarios where our training samples on the left may appear visually different from the test samples on the right. Despite sharing similarities or attributes such as a city in Figures 3.4a and 3.4b, or a robotic arm in Figures 3.4c and 3.4d, they still visually differ. If our model never learns from a city with snow in the image, it may lead to lower performance. This mismatch between our source domain, the training images, and our target domain, the test images, is referred to as domain shift.

3.2.2 Domain Shift

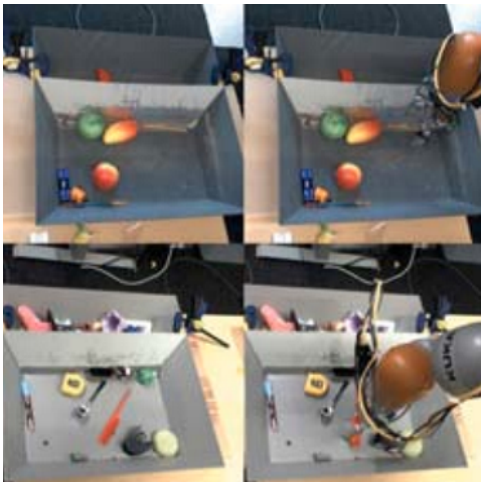
There are two factors that relate to the domain shift, the first relating to the differences in the data distributions themselves. Figure 3.5 shows a t-SNE [124] plot of the second to last layer activations in a network to visualize features in 2D space. Here we see separations between the source data distributions in blue and target data distributions in red. This separation highlights the difference in how the underlying data is different. The second point is that the network could learn and extract discriminative features from the source domain, leading to the clusters in the plot for the distinct blue classes. Yet, those learned features from the source domain were not relevant to the target domain, resulting in a sparse distribution of the data points in the plot.

(a) Image from the BDD100k dataset in daylight and clear skies [234]

(b) Image from the BDD100k dataset at night in snow [234]



(c) Synthetically generated images showing a robotic arm taken from Bousmalis et al. [15]



(d) Real-world scenarios where the robot must pick up objects taken from Bousmalis et al. [15]

Figure 3.4: The figure shows an example of domain mismatch in autonomous driving and robotic manipulation scenarios. We see differences in visual appearance between the training images on the left and the test images on the right. The autonomous vehicle scenario shows a training image in bright daylight, while the test image is at night in the snow. The robotics scenario shows differences in illumination, shadows, object shape, and textures. These differences demonstrate a domain mismatch between the source (left) and target (right) domains. Images taken from the BDD100k dataset [234] and from Bousmalis et al. [15].

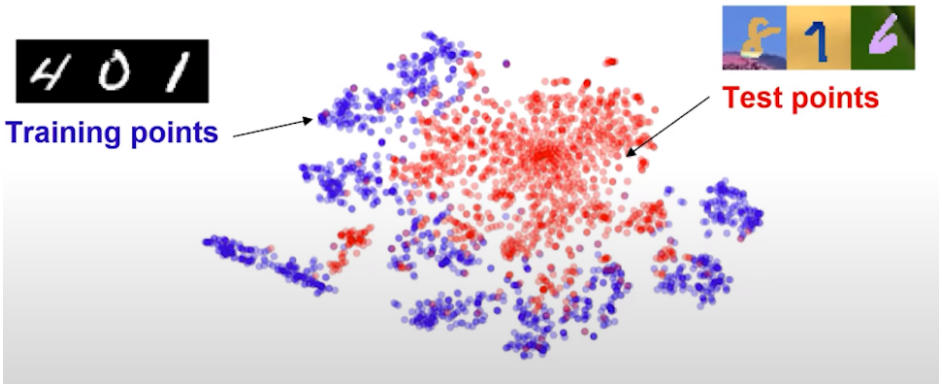


Figure 3.5: T-SNE visualization of the activations from the second to last layer of a network [55, 181]. The training and test data differ in how the points are distributed and clustered. Image taken from [181].

3.2.3 Notations and Definitions

Let us formalize this concept, assuming the notations and definitions used to match the existing literature [37, 149, 194, 214]. We assume a domain $\mathcal{D}$ comprising a feature space $\mathcal{X} \subset R^d$ for d-dimensions and marginal probability distribution $P(X)$, where $X = \{x_1, ..., x_n\} \in \mathcal{X}$. For a traditional binary classification problem, this means that $\mathcal{X}$ is the space of all term vectors, and X is a specific learning sample. For a task $\mathcal{T}$ and feature label space $\mathcal{Y}$, we can view this as a supervised machine learning task, with an objective function $f(\cdot)$. For a given domain $\mathcal{D} = \{\mathcal{X}, P(X)\}$ we may view the objective function $f(\cdot)$ in a probabilistic viewpoint with conditional probability distribution $P(Y|X)$. Take a general supervised binary classification machine learning task $\mathcal{T}$, where we have a sample set $X = \{x_1, ..., x_n\}$ from feature space $\mathcal{X}$ with associated labels $Y = \{y_1, ..., y_n\}$ from label space $\mathcal{Y}$. Our binary classification task $\mathcal{T} = \{\mathcal{Y}, P(Y|X)\}$ is solved by learning from sample pairs (x_i, y_i), where $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$. For the binary classification task, this means our set of labels (True/False or 0/1) is from $\mathcal{Y}$. Given a new instance of x, the binary classification objective function $f(\cdot)$ would classify this sample as one of the available labels.

In the traditional machine learning fashion, we assume the source and the target domains

and tasks to be the same. That is, given a training set with annotated data, our source domain $\mathcal{D}^s = \{\mathcal{X}^s, P(X)^s\}$. Our test set would therefore be defined as the target domain $\mathcal{D}^t = \{\mathcal{X}^t, P(X)^t\}$. Our source and target tasks would be defined as $\mathcal{T}^s = \{\mathcal{Y}^s, P(Y^s|X^s)\}$ and $\mathcal{T}^t = \{\mathcal{Y}^t, P(Y^t|X^t)\}$ respectively. Suppose $\mathcal{D}^s = \mathcal{D}^t$ and $\mathcal{T}^s = \mathcal{T}^t$, then we fall into the traditional machine learning regime, where both the source and target domains match, as well as the source and target tasks. Where we see models result in poorer performance, is when $\mathcal{D}^t \neq \mathcal{D}^s$ or $\mathcal{T}^t \neq \mathcal{T}^s$. Pan and Yang [149] categorizes transfer learning (TL) into three groups, starting with TL:

Definition 1 (Transfer Learning) *We define transfer learning as the scenario where our source/target domain do not match, or our source/target tasks do not match. For the following $\mathcal{D}^s$, $\mathcal{T}^s$, $\mathcal{D}^t$, and $\mathcal{T}^t$, transfer learning improves $f(\cdot)$ in $\mathcal{D}^t$ using information from $\mathcal{D}^s$ and $\mathcal{T}^s$, in scenarios where $\mathcal{D}^t \neq \mathcal{D}^s$ **or** $\mathcal{T}^t \neq \mathcal{T}^s$ Pan and Yang [149].*

Based on the definition given by Pan and Yang [149], transfer learning can account for either differences in the source and target domains, such as data distribution or feature space shifts, or differences in source and target tasks, such as classifying entirely different classes, or an unbalanced number of classes between the source and target tasks.

Definition 2 (Homogeneous Domain Adaptation (DA)) *Given a source domain $\mathcal{D}^s = \{\mathcal{X}^s, P(X)^s\}$ and target domain $\mathcal{D}^t = \{\mathcal{X}^t, P(X)^t\}$, with tasks $\mathcal{T}^s = \{\mathcal{Y}^s, P(Y^s|X^s)\}$ and $\mathcal{T}^t = \{\mathcal{Y}^t, P(Y^t|X^t)\}$, in homogeneous DA the source and target feature spaces are the same $\mathcal{X}^s = \mathcal{Y}^t$, but the marginal probability distributions differ $P(X)^s \neq P(X)^t$.*

Based on definition 2, the examples shown in Figure 3.5 are examples of domain divergence or domain shift where the source and target domains do not match $\mathcal{D}^t \neq \mathcal{D}^s$ when attempting to solve the same task. In this scenario, models trained on the source domain would perform poorly when evaluated on the target domain.

Definition 3 (Heterogeneous Domain Adaptation (DA)) *Given a source domain $\mathcal{D}^s = \{\mathcal{X}^s, P(X)^s\}$ and target domain $\mathcal{D}^t = \{\mathcal{X}^t, P(X)^t\}$, with tasks $\mathcal{T}^s = \{\mathcal{Y}^s, P(Y^s|X^s)\}$ and $\mathcal{T}^t = \{\mathcal{Y}^t, P(Y^t|X^t)\}$, in heterogeneous DA the source and target feature spaces differ $\mathcal{X}^s \neq \mathcal{Y}^t$, with the possibility of different dimensions d. i.e $d^s \neq d^t$.*

In the case of heterogeneous DA, we may have scenarios where the source and target representation are different. For example, our source domain may consist of images, while our target domain would be text. Regardless of the form of DA, there are several techniques that tackle the problem in supervised, unsupervised, and semi-supervised fashions.

The role of synthetic data typically falls under the situation where the source and target data distributions differ when attempting to solve a given task. Despite attempting to solve the same tasks, with images that may appear visually similar, the underlying data distributions would differ. For example, Figure 3.6 shows an autonomous driving scenario with samples from the KITTI [57] and VKITTI [54] datasets.

3.3 Applications of Synthetic Data within Computer Vision and Robotics

This section provides an overview of the state-of-the-art techniques used to bridge the gap between synthetic to real-world scenarios in various computer vision and robotics contexts. The focus will be on visual sim-to-real transfer, with particular attention on using synthetic data in some form in the process. Broadly speaking, major works' usage of synthetic data is categorized into four key areas:

- Using synthetic data only
- A combination of synthetic and real data

Figure 3.6: Images were taken from Gaidon et al. [54]. The figures show five real-world scenes on the right from the KITTI dataset [57], and five matched synthetic scenes from the VKITTI dataset [54] on the left. The matched artificial scenes differ visually from the real-world scenes, such as the lighting (global illumination appears to light up the entire scene compared to the real-world images), textures and material properties used on the cars, or softer shadows in reality.

- Synthetic refinement methods
- Procedural synthetic data generation

Each of the following subsections reviews the most relevant pieces of work that employ such approaches.

3.3.1 Synthetic Data Only

As discussed in section 3.2, relying purely on synthetic data as the source typically underperforms when the target data is real-world due to domain shift ($\mathcal{D}^s \neq \mathcal{D}^t$). However, synthetic data has desirable properties in specific tasks where attaining ground truth annotations in the real world is challenging. For example, optical flow estimation problems are often complicated to acquire high-quality ground truth annotations from real-world scenes, resulting in methods adopting synthetic data as an alternative [7, 21, 129, 133, 223]. Several of these works typically used synthetic data as a means to benchmark their optical flow algorithms, visual odometry, or evaluating image features [54, 128, 131].

More recently, access to high-quality simulators allowed researchers to generate higher quality photorealistic images, shifting the focus from benchmarking existing algorithms to using synthetic data as part of the training process. Richter et al. [171] presented an approach for using high-quality pixel-level annotations for solving semantic segmentation using images obtained from a commercial video game. This work was one of the early adopters of using higher-quality simulators to train Convolutional Neural Networks (CNNs).

Researchers started exploring other possibilities in which synthetic data could fit a training regime for solving various tasks. Further works investigated scene understanding [74, 132, 244], in which large-scale synthetic datasets of indoor scenes were generated. McCormac et al. [132] discovered that pre-training on synthetic indoor RGB images outperformed relying on pre-trained

ImageNet weights before solving a downstream task. This finding indicates that features from synthetic data are still relevant and helpful in solving real-world tasks, despite not matching the target data distribution.

We also see other studies in human pose estimation, person re-identification, and crowd counting, seeing benefits from using synthetic data as part of the process [183, 211, 216]. Varol et al. [211] created the SURREAL dataset, which contains 6 million RGB, depth, and body-part segmentation annotations using photorealistic rendering from a motion-capture (MoCap) system, achieving high accuracy when training a body-part segmentation network.

So far, we have discussed several techniques using only synthetic data that relied on incorporating high-quality renderings for the data generation process. Intuitively, the hope is to align closer the data distributions between the source domain $\mathcal{D}^s$ and the target domain $\mathcal{D}^t$. In contrast, Sadeghi and Levine [180] and Tobin et al. [202] introduced a unique method for using synthetic data as part of the training process called Domain Randomization (DR). In this scenario, simulator parameters are randomized from a pre-determined probability distribution to sample the parameters. The goal here is not to closely align the data distributions between the source and target domain but to "balloon" the source synthetic data domain to encompass the target data distribution. The idea of DR is an integral part of the thesis and is further discussed in section 4.

It is worth noting that solely using synthetic data can achieve high accuracy for a particular task. However, several works have concluded that the addition of some real-world data as part of the training process typically increases performance [45, 66, 73, 211, 232, 248]. The addition of the real-world data can be in the form of splitting the training set as a combination of synth+real [45, 73, 232], or by pre-training on synthetic images and fine-tuning on real images [66, 211, 248]. Although, this does depend on the ability to attain large real-world datasets in the first place, which might be contrary to the problem researchers are trying to solve.

3.3.2 Combining Synthetic and Real Data

As discussed in section 3.2, relying purely on real-world data is often challenging, both in terms of the time required to gather a large-scale, diverse dataset and the time and financial cost of annotating the data. However, we do see complex tasks solved using this method both in vision and robotics [13, 84, 112, 175], with the caveat of additional costs. For example, Levine et al. [112] used a data-driven, deep learning-based approach that utilized 800,000 grasp attempts with 14 robotic manipulators gathered over two months. This large-scale dataset was subsequently used to train a convolutional neural network (CNN) for grasp prediction. Aside from the costs, these approaches may be suitable in constrained environments where a robotic arm can operate but may not be ideal for more open conditions.

Hence, researchers have been adopting methods for combining both real-world and synthetic images for solving various tasks [192, 208, 238]. The approach can function as simply as placing 3D textured models in mid-air on top of real-world background images, such as Su et al. [192] for viewpoint estimation, shown in Figure 3.7.

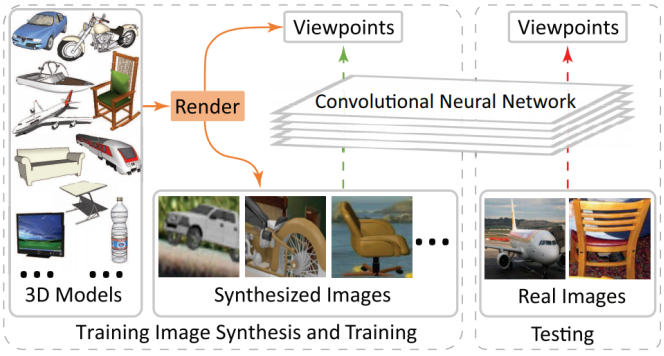


Figure 3.7: Sample training images from Su et al. [192], showing an approach to synthesize synthetic images by placing the 3D models on top of real-world images. A renderer generates synthetic images by overlaying the 3D models on the left from different viewpoints with random real-world backgrounds for solving viewpoint estimation.

Other techniques evolved from this approach, such as placing synthetic objects on ran-

dom real-world backgrounds with additional contextual information such as the work by Dwibedi, Misra, and Hebert [45] for object instance detection in Figure 3.8. Researchers took this further by placing synthetic objects more naturally in a scene, using existing predictions for surfaces or counters to place synthetic items on top of them for solving object detection in indoor scenes.

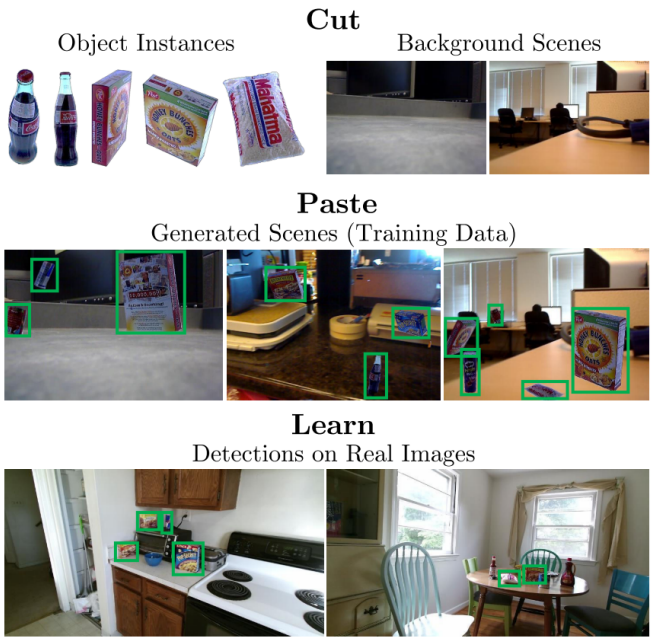


Figure 3.8: A Process for training a CNN for object instance detection taken from Dwibedi, Misra, and Hebert [45]. The idea is to place 3D models on real-world background scenes in more natural positions, such as on the surface of a table, to generate a new synthetic dataset. This synthetic dataset is used to train an object detection network.

The idea of placing synthetic objects of interest on real-world backgrounds continued to see use in other tasks such as hand pose estimation, object pose estimation, and semantic segmentation [66, 73, 248]. The approach is quite simple and does not require much implementation time to increase transfer from synthetic to real. While these approaches tend to improve relative performance, they typically see the highest performance when using a combination of synthetic and real, or an equivalently diverse real data. Varol et al. [211] shows their body part segmentation network performed best when fine-tuning a synthetic network with real images, indicating the importance

of having some notion of the target data distribution.

3.4 Refined Synthetic Data

There are several ways synthetic data is generated outside of using existing simulators or renderers in the generation process, either for synthesizing entirely new images or enhancing the realism of existing synthetic data. The cost of generating photorealistic images can be computationally expensive, paving the way for further research that aims to accomplish the goal of transfer without the costly overhead of traditional rendering. Typically, photorealistic scenes require an expert understanding of 3D modeling and scene creation, which is usually a manual and time-consuming process. This process involves designing and placing objects, accurate materials, and lighting in physically plausible ways, which led to exploring options for enhancing the realism of synthetic images.

Some of the earlier methods for enhancing realism involved transferring color between images. For example, a possible use case would be performing color correction on synthetic images to match colors from a real-world counterpart [156, 166, 229]. While this approach may not modify the positions of the objects, their geometry, or textures, it can enhance realism by using colors that visually appear from the real world. The earlier methods typically relied on statistical analysis to perform image processing on the images, contrary to more modern data-driven approaches that leverage large datasets.

The introduction of Generative Adversarial Networks (GANs) by Goodfellow et al. [64] introduced the world to a novel way of estimating generative models in an adversarial manner, which significantly propelled the field of image synthesis and domain adaptation. The framework is presented as a minimax game, in which two networks are trained simultaneously, with one network tasked to produce data (synthetic). In contrast, the other is tasked to predict the probability a given sample is from the actual dataset (real data) instead of an example from the other network.

The framework is posed as follows: given a generator G, distribution p_g over data x, noise defined as $p_z(z)$, mapping to data space $G(z;\theta_g)$, where θ_g represents the network's parameters of G. Given a discriminator D, a network with parameters θ_d, the network is defined as $D(x;\theta_d)$. The goal of the discriminator is to output a scalar denoting the probability x is from the real data and not a sample from the distribution p_g. The objective function as defined by Goodfellow et al. [64] is as $F(G, D)$:

$$\min_{G}\max_{D} F(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[logD(x)] + \mathbb{E}_{z\sim p_z(z)}[log(1 - D(G(z)))] \tag{3.1}$$

The introduction of GANs leads to several image synthesis techniques based on this approach, mainly concerned with image-to-image translation, where the goal is to learn a mapping from an input image to the desired output image [33, 79, 83, 150, 217, 218, 246]. The majority of these methods use semantic labels to generate photorealistic images using a variation of the GAN called Conditional GANs (CGAN), where the generator and discriminator are conditioned during training time by additional information such as class labels. For example, Wang et al. [215] introduced a framework for image-to-image translation using semantic information, allowing to transfer from semantic descriptions to high-quality images. This approach can involve taking semantic labels from a synthetic dataset and match the real-world's style while preserving the original input's semantics as shown in Figure 3.9.

The general approach is as follows, where I is the guided natural looking image, y is the image to synthesize using semantic label x and image $I : (x, I) \mapsto y$. As seen in the top of Figure 3.9, I serves the purpose of enforcing a stylistic constraint during the image synthesis process, where the synthesized images y in the bottom of the figure are stylistically the same. The authors propose to learn style consistency between pairs of images, either through style consistent or style-inconsistent pairs. While the works mentioned above use supervised or semi-supervised methods during the training process, unsupervised image-to-image translation techniques exist while adding additional cycle consistency constraints [96, 117, 233, 245].

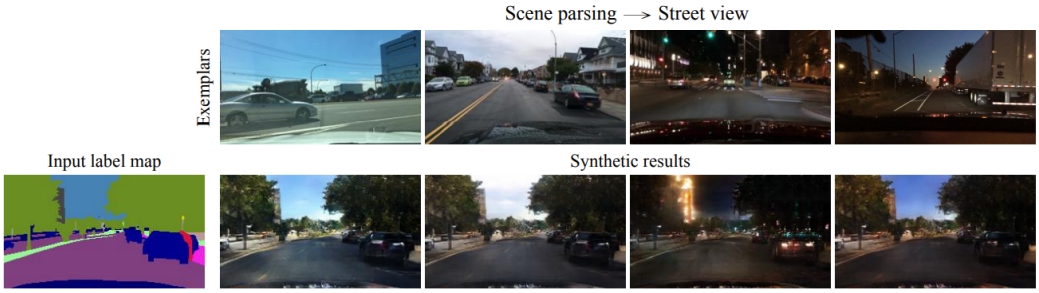


Figure 3.9: Sample images showing style consistency between real-world images at the top and semantic label on the left. The generated synthetic scenes at the bottom use the semantic label on the left and match the style from the real-world images. The generated synthetic images are style consistent by maintaining similar illumination conditions such as the time of day, the visual appearance of the roads, and the sky. Image is taken from Wang et al. [215].

Alternative methods for producing synthetic data, as shown by [15, 170, 186] involve creating datasets that contain synthetic images augmented to resemble the real-world equivalent. The improved realism of the images would help transfer to the target domain. Shrivastava et al. [186] designed a method in which annotation information from the refined images are preserved to train the model for gaze direction prediction. The refined image approach presented by Shrivastava et al. [186] appears to be robust due to the continuous production of new training data with preserved annotations, as it would lessen some of the artifacts that we could find in synthetic data. However, using some of the previous approaches relies heavily on an existing real image training set. For example, Shrivastava et al. [186] used a large collection of 214,000 real annotated images to train the network. Similarly, Bousmalis et al. [15] conclude the best results of successful grasps in a manipulation task on previously unseen objects was a result of an entire dataset comprising slightly over 9.4 million real-world images, which is extremely expensive and time-consuming to attain.

3.5 Procedural Synthetic Data Generation

So far, we have covered techniques for training models purely using synthetic data, combining real and synthetic data, and refining synthetic scenes using variations of GANs to synthesize photorealistic images. However, these methods require some knowledge or understanding regarding the parameters that need to be tuned to generate the desired output. For example, if we were to generate synthetic data using a simulator, what would be the best approach for placing cars in the scene? How would they be positioned in the camera's perspective? There is still some consideration into how we may design the scenes before data collection.

Recently, an exciting research direction involves automatically tuning simulator parameters to a given target data distribution. Take our autonomous vehicle scenario: automatic tuning would involve *learning* where to place roads, lane markings, cars, pedestrians, vegetation, and buildings to create natural-looking scenes resembling the target distribution.

Several methods explore systems of procedurally generating synthetic scenes by directly optimizing simulator parameters, predominantly using some form of reinforcement learning (RL) [41, 85, 177, 231]. The techniques involving the procedural generation of synthetic data usually use a non-differentiable simulator or rendering systems, meaning the gradients must be approximated using alternative methods such as REINFORCE [221]; this is because a given loss with respect to the simulator parameters θ is non-differentiable.

Ruiz, Schulter, and Chandraker [177] proposed the work "learning to simulate", which is an approach that learns to maximize the validation accuracy for a given task based on a set of simulator parameters. In their work, a policy π_ω outputs simulator parameters $\psi \sim \pi_\omega$. Based on parameters ψ, the simulator can be treated as a generative model where $G(x, y|\psi)$, producing data (x,y) based on parameters ψ. The reward R is determined by the accuracy obtained when training a task T with the simulated data and evaluated on the validation set. The authors set to maximize the object below:

$$J(\omega) = \mathbb{E}_{\psi \sim} \pi_{\omega}[R] \tag{3.2}$$

The gradients can be obtained for updating ω using REINFORCE:

$$\nabla_{\omega} J(\omega) = \mathbb{E}_{\psi \sim \pi_{\omega}}[\nabla_{\omega} log(\pi_{\omega}) R(\psi)] \tag{3.3}$$

The approach adopted by [177] is similar to methods used in several other works that involve meta-learning [41, 56, 85, 122, 177, 231]. However, using this method can be unstable due to variance in the learning process, although there are methods to reduce this variance and keep the bias unchanged [177]. The approaches also treat the entire process as a black-box, including the data generation portion, and have costly training times due to multiple objective evaluations at every iteration [11].

Many of these techniques face challenging problems, as the physics simulators and renderers we commonly use are non-differentiable. Behl et al. [11] moves away from REINFORCE type gradient approximations by proposing a novel differentiable approximation on an objective, reducing the computational cost of training, and reducing the number of samples needed while achieving similar accuracies compared to RL-based techniques.

Other alternatives would be to use simulators or renderers that are differentiable such as Figure 3.10, which highlights the differences between traditional rendering and differentiable rendering. In traditional rendering, the discretization step of rasterization prevents the process from being differentiable and learned [92, 118]. Using differentiable rendering [91, 114, 118, 143] allows us to learn 3D representations from 2D images and implement the rendering process in a differentiable way, allowing gradients to be backpropagated through neural networks.

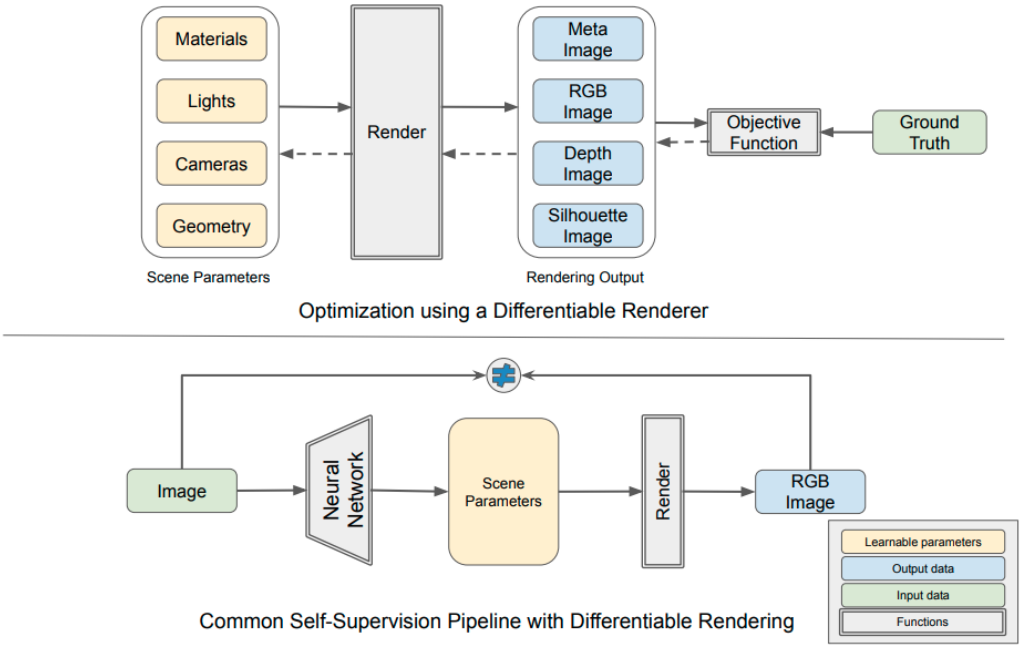


Figure 3.10: Differentiable rendering overview as presented by Kato et al. [92]. With differentiable rendering, we may compute gradients of some objective function with respect to the scene parameters and ground-truth in the image on top. Traditional rendering does not allow for the computation of gradients which is required when using neural networks.

3.6 Conclusion

This chapter presented an overview of the current state of learning from synthetic data. It introduced the attraction and significant benefits of using synthetic data as part of the training regime, from pixel-perfect annotations for generally complex and time-consuming annotations to collect, such as semantic segmentation or depth maps - to the ease, speed, and greater control of parameters to generate the scenes. The added benefit of preserving privacy and increasing fairness while reducing dataset bias is also an important but overlooked part of using synthetic data in a training regime.

The chapter covered key concepts within Transfer Learning and Domain Adaptation, highlighting key issues to overcome to transfer from synthetic to real. Across a broad range of applications within computer vision and robotics, we looked at popular approaches for both offline and online generation of synthetic data, from solely using synthetic data, using a combination of synthetic and real, refining synthetic data, and procedurally generating them.

While the field is still somewhat nascent, it is quickly developing and adopted across a wide variety of tasks. Current data-driven approaches will undoubtedly benefit from leveraging some of these techniques, particularly in scenarios where real-world data collection is daunting.

Chapter 4

Domain Randomization

In chapter 3 we covered many techniques that attempt to tackle the challenge of bridging the gap from simulation to real, ranging from solely using synthetic data rendered offline to procedurally generating the data during the learning process. The thesis is primarily concerned with learning from simulated data through Domain Randomization (DR). This section will further review a theoretical framework for DR, how it affects data distributions and the current state of DR.

4.1 Introduction

DR is an exciting approach that has proven to aid in the transfer from synthetic to real in several tasks, particularly in the field of robotics [15, 127, 147, 180, 195, 201, 202, 205, 219]. Assume we have a labeled training set of k, where x_{dr} is a DR image, and y_{dr} is the label, $\{x_{dr_i}, y_{dr_i}\}_{i=1}^{k}$, and have an objective of solving task T. This task is to be evaluated on a real-world test set of size n, where x_{real} is real-world image, and y_{real} is the label, $\{x_{real_i}, y_{real_i}\}_{i=1}^{n}$. Note from the definition in section 3.2.2, this resembles a domain shift problem and may treat this discrepancy between our source and target domain as a domain adaptation problem. Unlike some common

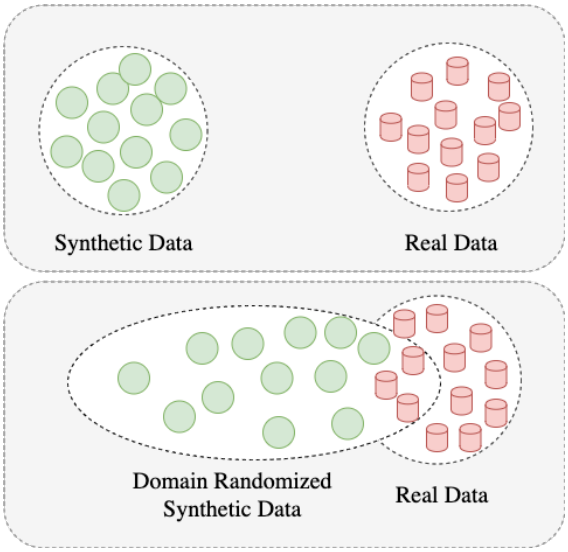


Figure 4.1: A visual representation of a synthetic and real-world data distribution on top, for example, a training dataset of synthetic household tabletop scenes and a test dataset of real-world household tabletop scenes. When using domain randomization (DR), the synthetic data distribution is expanded by including some combination of variations in textures, poses, illumination, or backgrounds. This technique would broaden the data distribution such that a real-world sample may appear as another variation in the training distribution.

domain adaptation methods, which aim to align the mismatched data distributions closer or define a mapping function to map one domain to the other, DR is unique.

The main idea behind DR is to generate highly varied datasets by using a simulator g to randomize a set of parameters θ, to generate labeled DR data $\{x_{dr_i}, y_{dr_i}\}_{i=1}^{k}$. This process is repeated k times, resulting in a wide and varied dataset that would be robust to function on the target data distribution. Figure 4.1 illustrates the differences between several domain adaptation techniques compared to DR. In this figure, we see that the outcome of applying DR expands the D_{dr} domain, such that it may encompass the D_{real} domain. DR, therefore, attempts to produce a large diversity in possible labeled scenes, forcing a given model to be more robust to variation in settings. In contrast to several domain adaptation methods, DR typically functions purely on synthetic data and does not require any information from the target domain D_{real}.

4.2 Algorithm

We first see the term DR introduced in the works of Sadeghi and Levine [180] and Tobin et al. [202] for solving tasks involving object localization and indoor quadrotor collision avoidance using monocular RGB images. For example, Tobin et al. [202] trained an object detector that maps a monocular RGB image to Cartesian coordinates for each object trained only on DR data. Algorithm 1 defines the procedure to generating DR data to use for training, assuming access to a simulator g with simulator parameters θ to generate k samples of labeled data $\{(x_i, y_i)\}_{i=1}^{k}$, where x is a DR image, and y is the label. The simulator parameters are sampled from a probability distribution P_Θ.

Algorithm 1 DR Algorithm

Input: *simulator: g, number of samples: k*

Distribution: P_Θ

Output: *DRdata:* $\{(x_i, y_i)\}_{i=1}^{k}$

1: DRdata = $\{\}$
2: **for** $\{i = 1, ..., k\}$ **do**
3: sample simulator parameters (eg. textures, backgrounds, objects) $\theta \sim P_\Theta$
4: generate labeled DR sample $(x_i, y_i) = g(\theta)$
5: append sample to DRdata
6: **end for**
7: DRdata $= \{(x_i, y_i)\}_{i=1}^{k}$

Typically, the simulator parameters θ that we may randomize are some combination of the following for generating scenes to train task-based networks, resulting in images generated by the seminal DR work by Tobin et al. [202] in Figure 4.2:

- Textures of all objects
- Background of the scene

- Number and position of objects of interest
- Number and position of distractor objects (not of interest)
- Number, positions, and intensity of light sources
- Position, orientation, and camera properties for the camera used
- Random noise applied to generated images

Figure 4.2: Samples of DR data from the original DR work by Tobin et al. [202]. Simple geometric objects are randomized and used as part of the training set for the localization of an object of interest. Object textures, positions, camera positions, and backgrounds are randomized.

The simulator parameters are typically sampled from a predetermined probability distribution. However, the existing works that use DR in their training regime usually select a uniform probability distribution to sample from [202, 237].

4.3 Applications

DR has been widely applied to a variety of tasks as shown in table 4.1, where we see a breakdown of popular features across the existing literature. This includes the type of task solved, the complexity of objects of interest, defined by simple geometric shapes or more complex shapes used, and scene complexity - defined by occlusion, distractor objects, multiple classes, or instances.

Approach	Task	Object Complexity	Scene Complexity
Yan, Tyree, and Kautz [232]	Grasping	Simple geometric	Single object
James, Davison, and Johns [81]	Grasping	Simple geometric	Single object
Sadeghi and Levine [180]	Collision avoidance	N/A	Complex (navigating furniture)
Tobin et al. [201]	Grasping	Complex	Single object
OpenAI et al. [147]	Manipulation	Simple geometric	Single object
Tobin et al. [202]	3D Pose Estimation	Simple geometric	Single object + distractors
Borrego et al. [14]	Object detection	Simple geometric	Complex - Multiple objects
Tremblay et al. [207]	Object detection	Complex	Complex - Multiple objects
Florence, Manuelli, and Tedrake [52]	Grasping	Complex	Complex - Multiple objects
Bousmalis et al. [15]	Grasping	Complex	Complex - Multiple objects
Pinto et al. [155]	Manipulation	Simple geometric	Single object
Ward, Moghadam, and Hudson [219]	Segmentation	Complex	Single object
Matas, James, and Davison [127]	Manipulation	Complex	Single object
Tremblay et al. [205]	6D Pose Estimation	Complex	Complex - Multiple objects
Sundermeyer [195]	Object detection + pose estimation	Complex	Complex - Multiple objects
Tremblay et al. [206]	Object detection + motion planning	Simple geometric	Complex - Multiple objects
Zhang et al. [237]	Object localization	Simple geometric	Complex - Multiple objects
Pouyanfar et al. [159]	Obstacle avoidance	N/A	Complex - (urban driving environment)

Table 4.1: Table showing the types of tasks, object complexity, and scene complexity in the current literature using domain randomization (DR).

From this table, we see various tasks solved, though a large portion seems to focus on pose estimation and scene understanding in object detection and segmentation tasks. Earlier works appear to focus on using more simple geometric shapes, with medium scene complexity, meaning light occlusion, some usage of distractor objects, and single objects of interest. For example, Tobin et al. [202] and Tremblay et al. [205] train a model per object of interest to detect, meaning to solve a complex pose estimation scene with multiple objects, we must instantiate several networks simultaneously.

The usage of DR quickly expanded to several other tasks [81, 147, 219, 232], including in works of Borrego et al. [14], where DR was applied to train object detection models, improving accuracy with limited access to data. Robotic manipulation is commonly being used with DR, as roboticists typically spend a significant amount of time learning in simulation yet normally encounter performance issues when transferring to the real-world. We see works ranging from grasping to in-hand manipulation that uses some form of DR as part of their training set for solving a task [15, 52, 81, 147, 232].

Moving away from the traditional approach of complete randomization, Prakash et al. [160] proposed to enforce some structure and context when applying DR, stating the importance of keeping some semblance of what the target data would be. In their scenario of detecting cars in the scene, they found it important to position certain objects such as buildings, street signs in plausible locations, as opposed to complete randomization. This approach of *structured domain randomization* appeared to increase performance on their object detection task from 56.8 mAP to 69.6 mAP.

Interestingly, the tasks above were still solvable despite the generated data not resembling scenarios we would typically encounter in the real world, which may highlight the importance of sufficient data variation to aid in the transfer from synthetic to real, as we would like the synthetic source domain to encompass the real-world target domain.

4.3.1 Variations in Applying DR

When attempting to apply some of these techniques to a new task, how do we ensure that the data we are creating using DR would help solve our task? Intuitively, we may think that uniformly sampling all possibilities of textures to apply to meshes of objects in our scene may be wasteful, resulting in training data that may hinder performance. In the existing literature for solving the tasks in Table 4.1, there does not appear to be a consistent manner of applying DR techniques, nor is it clear whether the same technique would work across multiple tasks and generalize in the same way.

For example, comparing similar tasks from James, Davison, and Johns [81] and Yan, Tyree, and Kautz [232], the approach to randomization of simulator parameters is quite different. When solving a grasping task using single, simple geometric objects, the method by Yan, Tyree, and Kautz [232] randomized only the background using real-world images and the poses of the objects of interest. Individual textures of the objects and lighting were not randomized. However, the entire synthetic image is color shifted after replacing the background in the HSV space to accommodate for the lack of texture and illumination randomization. In contrast, James, Davison, and Johns [81] used distractor objects, randomized the positions of the camera, lighting, and individual textures of the objects. Furthermore, individual textures were also generated using random noise by applying Perlin noise [152]. Table 4.2 highlights differences in approaches to using DR. While there are common themes such as randomizing textures, camera, and lighting, features such as distractors or random noise are less prevalent.

This inconsistency is quite apparent when breaking down individual randomization techniques as shown in Table 4.3, where we see commonly applied textures in the literature, one of the key features to aid in transfer several works highlight the importance of [207]. In this table, we see a large portion opting to use Flat RGB, simply sampling a value in some color space to apply on the meshes of objects. Despite the range of possibilities of textures to use, it appears to occur more frequently than others. However, there is currently no understanding of the most suitable technique

Approach	Textures	Background	# Objects	Object Pose	Distractors	Camera	Random Visual Noise	Lighting
Yan, Tyree, and Kautz [232]	✗	✓	✗	✓	✗	✗	✗	✗
James, Davison, and Johns [81]	✓ (normal distribution)	✓	✗	✓	✓	✓	✓	✓
Sadeghi and Levine [180]	✓	✓	N/A	N/A	✗	✓	✗	✓
Tobin et al. [201]	✓	✗	✓	✓	✗	✓	✗	✗
OpenAI et al. [147]	✓	✓	✗	✓	✗	✓	✗	✓
Tobin et al. [202]	✓	✓	✓	✓	✓	✓	Gaussian[1]	✓
Borrego et al. [14]	✓	✓	✓	✓	✗	✓	✓	✓
Tremblay et al. [207]	✓	✓	✓	✓	✓	✓	Gaussian[1]	✓
Florence, Manuelli, and Tedrake [52]	✓[2]	✓	✓	✓	✗	✓	✗	✓
Bousmalis et al. [15]	✓	✗	✓	✓	✗	✓	✗	✓
Pinto et al. [155]	✓	✓	✗	✓	✗	✓ (restricted)	✗	✓
Ward, Moghadam, and Hudson [219]	✓	✓	✗	✓	✗	✓ (restricted)	✗	✓ (restricted)
Matas, James, and Davison [127]	✓	✗	✗	✓	✗	✓	✓	✓
Tremblay et al. [205]	✓	✓	✓	✓	✓	✓	✗	✓
Sundermeyer [195]	✓[2]	✓	✗	✓	✗	✗	Gaussian[1]	✓
Tremblay et al. [206]	✓	✓	✗	✗	✓	✓	Gaussian[1]	✓
Zhang et al. [237]	✓	✗	✓	✓	✓	✓	✗	✗
Pouyanfar et al. [159]	✓	✓	✓	✓	✗	✓	✗	✓

Table 4.2: Table showing the selection of parameters when applying domain randomization (DR) in the existing literature. The most commonly used techniques are randomizing textures and object poses.

[1] Gaussian noise added as a post-processing step once the images were generated.

[2] Applied to the background only.

for a given task, as the current literature does not provide an extensive comparative analysis on this issue.

Texture Randomization Techniques	[205]	[219]	[127]	[195]	[15]	[52]	[207]	[155]	[81]	[206]	[180]	[14]	[201]	[202]	[147]	[237]	[159]	[160]	[232]
Flat RGB	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	□	□
Gradient RGB	□	□	□	□	□	□	□	■	□	□	□	□	□	■	□	□	□	□	□
Patterns (Checkerboard)	□	□	□	□	□	□	■	■	□	□	□	■	□	■	□	□	□	□	□
Patterns (Striped)	■	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□
Patterns (Other)	□	□	□	□	□	□	□	□	□	□	■	□	□	□	□	□	■	□	□
Additional Noise (Perlin)	□	□	■	□	□	□	□	□	■	□	□	■	□	□	□	□	□	□	□
Real Images	■	□	□	■	□	□	□	□	□	□	□	□	□	□	□	□	□	■	■

Table 4.3: Table showing texture randomization techniques applied in current literature. The heavily favored approach is to use flat RGB textures, in which each texture is a single RGB color sampled from a predetermined distribution.

4.4 Conclusion

The thesis is motivated by learning purely from synthetic data when applied to real-world scenarios via DR. The appeal of generating vast quantities of annotated data is highly beneficial when training supervised tasks, particularly in vision tasks, where performance on classification, detection, and segmentation tasks increases logarithmically based on the size of the datasets used [151, 193]. However, we must still seek ways to overcome the domain shift, as described in Chapter 3.2.2, where we see purely using synthetic data typically underperforms using real-world data.

We have discussed in Chapter 4 how DR can be a quick method for enabling transfer from synthetic to real, although some open questions remain when using this method. This thesis aims to address several questions surrounding the use of DR. In this thesis, we focus on the randomization process around a central theme of using textures within DR. The reasoning for this focus is twofold: from Table 4.2, we see the top three most commonly applied processes within DR are textures, followed by object poses, and thirdly illumination.

Not only is it the most frequently used within DR, but more importantly, recent research has demonstrated that textures are considered one of the more significant decision criteria when using convolutional neural networks such as VGG [187], and ResNet50 [71] [1, 16, 58]. Several works have shown that convolutional neural networks are still capable of classifying texturized images with a high degree of accuracy, regardless of global object structure [1, 16, 58].

For example, Figure 4.3 shows a figure taken from [16] showing examples of original images on the left and texturized images on the right. An off-the-shelf VGG-16 model is capable of achieving high accuracy (90.1% on the unscrambled image compared to 79.4% on the scrambled image) when solving a classification task, despite a breakdown of global shape in the photos. The example illustrates that unlike humans, which benefit from global shape information, convolutional neural networks such as VGG, in this case, focus more on local image features.

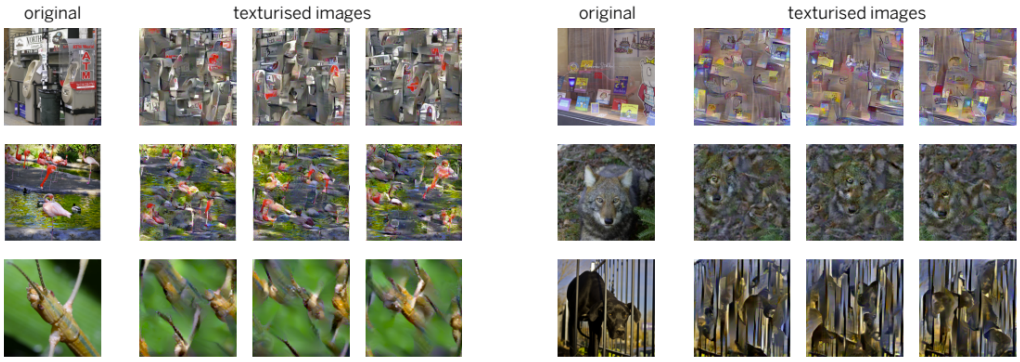


Figure 4.3: Sample images taken from Brendel and Bethge [16]. The figure shows original images to the left and scrambled images to the right. An off-the-shelf VGG-16 [187] model achieves 90% accuracy on an image classification task evaluated on the original unscrambled images and 79.4% accuracy on scrambled images. Despite a breakdown in global shape, the model was capable of yielding good accuracy, indicating that local image features are important.

Another example highlights the poor performance of pre-trained convolutional neural networks on ImageNet when classifying object sketches, which preserves shape but removes texture information [9]. Despite maintaining object shape, the absence of textural information severely hinders task-based performance.

The combination of the above studies indicates the importance of textural information when solving tasks using convolutional neural networks, where local textural information provides sufficient information in the scene and more important than global features such as shape. We envisage researchers incorporating the key findings from our focus on DR with textures in this thesis into the broader DR process.

The thesis tackles three main questions around the use of DR for transfer from synthetic to real:

- What DR techniques would be most suitable for an arbitrary task, and how do we best apply them?
- Does DR behave similarly across various tasks?
- Are there more suitable approaches to applying DR?

We address each of the questions above in the following chapters, starting with the first question presented in Chapter 5, which proposes a novel method for selecting more appropriate DR methods for a given task. Chapter 7 is related to the second question, where we investigate how DR generalizes across multiple tasks. Finally, an alternative method of producing DR textures is presented in Chapter 8.

Chapter 5

QDRNet - Quantifying the use of Domain Randomization

In this chapter, we present our work [1] [5] titled Quantifying the use of Domain Randomization (QDRNet). As discussed in Chapter 4, DR provides a method for expanding a synthetic data distribution to contain a realistic data distribution, as shown in Figure 4.1. The expanded synthetic data distribution is useful in scenarios where we cannot determine what the realistic data distribution is, therefore, achieving better performance for a given task when utilizing the synthetic dataset to train a given task-based model.

However, there is no agreed-upon method for applying various DR techniques for solving a task. The diversity in ways of applying the different techniques may lead to different performance depending on how the techniques are applied. We propose a novel approach for ranking DR

[1] The work presented in this chapter is part of research presented in Ani, Basevi, and Leonardis [5]. Results, figures, and text from the publication have been reused and adapted for the chapter in this thesis. I designed, programmed, conducted the experiments, and analyzed the results in the work, along with supervisory support from Dr. Hector Basevi and Professor Ales Leonardis. I wrote the manuscript presented in [5], with feedback from Dr. Hector Basevi and Professor Ales Leonardis.

methods by quantifying the differences in realistic and synthetic data distributions without the cost of measuring task-based performance.

The chapter opens with the proposed method in Section 5.2, which introduces the problem formulation and core approach for QDRNet. Section 5.2 also introduces the model architecture and flow of data through the network. Following this, Section 5.3 presents the algorithm for the data generation routine used in subsequent experiments. The experimental section comprises of two sections, the first being experiments conducted in the image space, followed by experiments conducted in the feature space. The two sections mentioned above are Section 5.4.1 and Section 5.4.2 respectively. The chapter concludes with a discussion of experimental outcomes and key conclusions from our work.

5.1 Introduction

In Chapter 3, we discussed the immense benefits of using synthetic data to train deep neural networks, particularly in supervised deep learning. Scenarios typically time-consuming or challenging to acquire vast amounts of labeled data can be drastically reduced when using synthetic data. Take, for example, an urban driving scenario, where we would like a vision system to detect various classes such as lanes, pedestrians, buildings, cars, or street signs. While not an exhaustive list, annotating several millions of frames containing the above information would be extremely costly, both financially and the amount of time required to acquire dense annotations. Additionally, dataset diversity is an integral part of the learning process, particularly in urban driving environments [31, 54, 57, 62], requiring us to collect diverse scenes under varying illumination conditions, weather conditions, or cities. Using synthetic data allows us to gather a diverse labeled dataset faster and cheaper than collecting images using RGB or RGB-D cameras in the real world.

However, we have previously discussed in Chapter 3 that solely using synthetic data typically underperforms real-world data due to the domain shift. We also discussed several proposed

methods for achieving transfer via domain adaptation [75, 115, 154, 186], particularly in the field of robotics using DR [14, 15, 81, 160, 180, 201, 202, 207, 232]. The key idea behind DR is to randomize simulator parameters such as the objects, their positions, camera positions, illumination, and textures. Tobin et al. [202] who were one of the first to coin the term DR, associate this approach with generating lots of variation during the training process for solving some arbitrary task, such that during evaluation on real-world images, the samples from the real-world would encompass some subset of the DR data distribution.

The goal of DR is not to generate data indistinguishable from real-world samples, as would be the case when applying generative models for synthesizing new data [18, 64, 87, 88, 90, 140]. While the process for randomization of simulator parameters using DR has shown promise [180, 202], it is unclear what particular set of randomizations would be most appropriate for an arbitrary task. Currently, there is no universal approach for applying the DR routine for solving a task, as previously shown in Table 4.2 from Chapter 4. For example, some DR methods choose a flat color such as shades of red, green, or blue as the texture to apply to objects when creating synthetic scenes, while others use more complex patterns such as checkerboards [14, 52, 155, 195, 219].

The selection of textures plays a crucial role in solving vision tasks using convolutional neural networks [1, 16, 58], and an unfavorable choice of the texture data distribution to sample our textures from may lead to worse task-based performance. We address the challenge of selecting more appropriate DR methods for generating DR synthetic images by introducing an approach for quantifying the differences between a source and target data distribution. In our scenario, our source domain would be the DR synthetic data, and the target domain would be realistic datasets.

We achieve this using two statistical distance measures - the Wasserstein and Fréchet Inception Distance (FID) - to measure the differences between our source and target data domain in the image and feature space. Here, we take the image space as a set of RGB images, while the feature space is defined as the set of extracted features after passing through a feature extractor. In this work, we propose to measure the difference between the most commonly used texture randomiza-

tion methods from Table 4.3, and realistic images using real-world equivalent textures. The most widely applied textures used in this work are non-patterned (Flat RGB, Gradient RGB), patterned (Checkerboard, Striped, Zig Zag), and additional noise in the form of Perlin noise [152]. Samples from these textures in the investigation are in Figure 5.1. We investigate currently implemented textures within the DR literature as shown in Table 4.3. Note from the table that the works that used additional noise are in the form of Perlin noise [14, 81, 127]. Perlin noise is a noise generation algorithm commonly used in computer graphics for procedural texture generation [106]. It is a computationally inexpensive method for generating smooth, intricate patterns such as clouds, smoke, marble, or water. The bottom row of Figure 5.1 shows the Perlin patterns used in the investigation, which is based on visually replicating the noise patterns from current works utilizing this method [14, 81, 127].

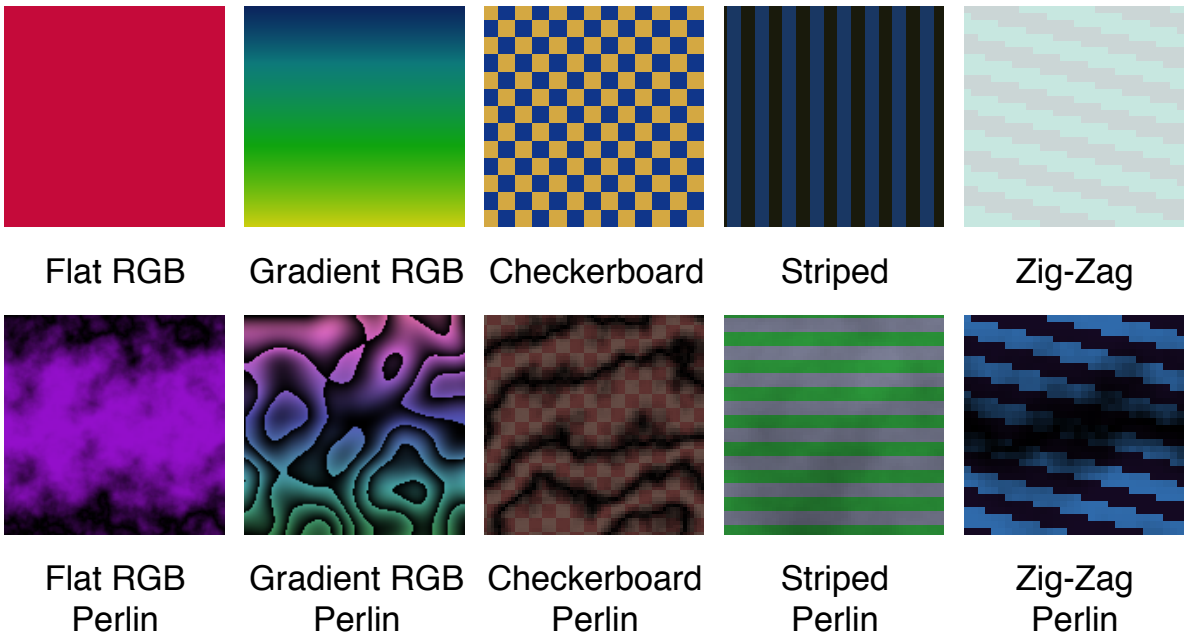


Figure 5.1: Sample textures used to generate domain randomized (DR) data. A combination of non-complex textures (Flat RGB, Gradient) and complex textures (Checkerboard, Striped, Zig-Zag, and using Perlin noise [152]) are used to create the data. The textures were selected to cover the types used in existing DR literature from Table 4.3.

We evaluate each of the commonly used texture DR methods on the same localization task

and similar architecture to one of the seminal works of DR by Tobin et al. [202]. We show that patterned textures achieve the highest task-based performance, unlike the most frequently deployed Flat RGB shown in Table 4.3. We illustrate that the localization task performance, and distance estimate using Wasserstein and FID in the feature space pre-trained on ImageNet weights, produce similar rankings in distance estimates and performance.

The main contributions of the chapter are as follows:

- We proposed a novel method to quantify the differences between source DR data and target realistic data using a small amount of data with neural networks.
- We show that the approach can rank the different DR methods, and the rankings produced are reflected in final task-based performance. Using the generated rankings for the different DR methods, we may predict task-based performance without the overhead of task-based training and find more complex patterned textures are most beneficial when generating DR synthetic data.

5.2 Method

In this section, we open with the problem formulation for measuring the distance between DR and non-DR data distributions, and the approach for solving this.

In chapter 4, we looked at how real-world, and synthetic data distributions differ, despite sharing similar features such as the positions of an object of interest in a scene, object type, or shared textures. This domain-shift increases the difficulty in ensuring our synthetic dataset performs similarly to an equivalent real-world dataset. DR is an attempt at broadening the synthetic data distribution such that the real-world may encompass some sub-space within the synthetic data distribution.

From Table 4.3, we see that while the approach is quickly being adopted to bridge the simulation-to-real gap, notably within the robotics community, we see that there does not seem to be a principled approach for applying this technique. The lack of a clearly established approach for applying DR motivated an investigation for further understanding and applying DR more efficiently.

Our proposed method measures the statistical distance between commonly used texture DR and realistic data distributions and ranks the various distances and their performance in a 3D object localization task. This eliminates the cost of using task-based networks to evaluate the impact of the texture DR that was applied.

We apply the above method in the image and feature space on various datasets that increase in complexity. As there are numerous ways to measure the distance between data distributions, we explore several distance measures in the existing literature. The distance measures investigated are Jensen-Shannon Divergence (JSD), Wasserstein Distance (WD), and Fréchet Inception Distance (FID).

5.2.1 Texture Domain Randomization

In this work, we focus on the use of texture randomization, as textures are considered one of the most important in neural networks, and the most heavily used in robotics DR applications. [14, 15, 52, 58, 81, 127, 147, 155, 159, 180, 195, 201, 202, 206, 207, 219, 237]. We implement all the methods used in current DR literature in Table 4.3.

For our experiments, we use a custom simulator to perform physics simulation and rendering, which allows us to generate physically plausible scenes, and control rendering parameters during experimentation. We implemented the texture generation methods shown in Table 4.3 and can apply the resulting textures, such as those seen in Fig. 5.1, to the objects in scenes. Illumination is fixed, and object poses are shared across datasets, such that the differentiating factor between

samples are the textures applied to the objects. Further details about the dataset generation routine is described in Section 5.3.

5.2.2 Quantifying Distances Between Distributions

Quantifying distances in data distributions is an integral part in several machine learning processes, particularly in generative models. For example, Variational Auto-Encoders (VAE) [98] and GANs [6, 64, 68] primary objective is to replicate a given data distribution.

VAEs make use of KL-Divergence in Equation (5.1) to measure the distance between two continuous probability distributions P and Q.

$$D_{KL}(P||Q) = \int_x P(x) log\left(\frac{P(x)}{Q(x)}\right) dx \tag{5.1}$$

We see that $D_{KL} = 0$ when $P(x) = Q(x)$. As the KL-Divergence is asymmetric, an issue arises in the instance where $Q(x) \approx 0$ and $P(x) > 0$, as the distance measure may tend to infinity.

The standard GAN [64] makes use of the JSD between P_{data} (the original data distribution) and P_g (the model's generated distribution), both defined on a compact data space χ.

$$D_{JS}(P_{data}||P_g) = \frac{1}{2}D_{KL}\left(P_{data}||\frac{P_{data} + P_g}{2}\right) + \frac{1}{2}D_{KL}\left(P_g||\frac{P_{data} + P_g}{2}\right) \tag{5.2}$$

JSD in Equation (5.2) being symmetric and bounded by [0,1] allows for smoother training for the generative models. However, if the distributions are far apart, the estimate is less meaningful as an indicator for sample quality of a generator G [6]. This is highlighted in the first implementation of the Wasserstein-GAN (WGAN), in which a correlation between lower error produced by the Wasserstein metric as a loss function, and better sample quality from a given

Generator G is seen. [6]. While JSD saturates at $ln(2)$, and continues to improve in the generated sample quality, it proves to be a less insightful metric on how different such distributions are when they are sufficiently far apart [6].

The use of WD in Equation (5.3) addresses the issue of JSD being a less meaningful measure for determining generated sample quality in GANs. Where $\Pi(P_{data}, P_g)$ denote the set of all joint distributions $\gamma(x, y)$ with marginals P_{data} and P_g

$$W(P_{data}, P_g) = \inf_{\gamma \in \Pi(P_{data}, P_g)} \mathbb{E}_{(x,y)\sim\gamma} \left[||x - y||\right] \tag{5.3}$$

There are several different ways works have enforced a Lipschitz constraint on the discriminator, each with varying results. Weight clipping was initially used to enforce the constraint [6], while others implemented a gradient penalty [68]. How to implement the Lipschitz constraint on the discriminator still remains an open problem, and in turn, affect how accurate the estimate is. This prevents direct comparison between the methods that do this differently. Here, we are interested in ranking the different texture randomization methods.

As we are dealing with unknown distributions when using images and in the feature space, we estimate JSD, WD, and FID using neural networks from distribution samples. In our experiments, JSD estimation is implemented identically as the standard GAN [64]. We use the same loss function and discriminator portion, while replacing the generated data from G with the randomized simulated data. In turn, we train the discriminator until optimal to compute the JSD between two data distributions. The estimate is computed in equation 5.2.

For WD, we use WGAN-GP instead of the original WGAN due to a more robust method of enforcing Lipschitz-Continuity [6, 68]. Due to the gradient penalty term in WGAN-GP restricting the norm of the gradient of the discriminator, we expect the implementation to affect the estimate of the Wasserstein distance, but not affect the ranking (ordering of the different texture randomization methods). We use a modified version to replace the generated samples with the randomized

simulated data [26].

Algorithm 2 Modified WGAN with gradient penalty. We use default values of $\lambda = 10$, $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$

Require: Gradient penalty coefficient λ, batch size *m*, Adam optimizer hyperparameters α, $\beta_1 = 0$, $\beta_2 = 0.9$, number of iterations *n*

Require: WGAN-GP critic *D* and initial parameters ω_0, two data distributions P_r as the real-texture data distribution and P_{aug} as the DR data distribution. $D_\omega(x)$ is the WGAN-GP critic with parameters ω and realistic samples x, and $D_\omega(\tilde{x})$ is the WGAN-GP critic with parameters ω and DR samples $\tilde{x}$

1: **for** *i* = 1, ..., *n* **do**
2: Sample a batch from realistic data $\{x\}_{i=1}^{m} \sim P_r$
3: Sample a batch from DR data $\{\tilde{x}\}_{i=1}^{m} \sim P_{aug}$
4: Sample a random number $\epsilon \sim U\ [0, 1]$
5: $\hat{x} \leftarrow \epsilon x + (1 - \epsilon)\tilde{x}$
6: $\{L\}_{i=1}^{m} \leftarrow D_\omega(\tilde{x}) - D_\omega(x) + \lambda(||\nabla_{\tilde{x}} D_\omega(\hat{x})||_2 - 1)^2$
7: $\omega \leftarrow \text{Adam}(\nabla_\omega \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, \omega, \alpha, \beta_1, \beta_2)$
8: **end for**

5.2.3 Image Space

With WD, we use a modified implementation of WGAN-GP [68] and replace the generated samples with DR samples to the WGAN-GP critic. We use the computed WD to rank the various texture randomization methods from the lowest to highest distances.

We evaluate this ranking on an object localization task, where the goal is to predict 3D position of an object of interest using VGG-16 [187]. In the image space, we use the same VGG-16 architecture implementation found in one of the seminal DR works by Tobin et al. [202].

5.2.4 Feature Space

Pre-trained backbones are widely available to bootstrap learning for new tasks. The availability of pre-trained backbones makes it possible to measure the distance between distributions in feature space from existing networks already trained on a large amount of data. Additionally, FID is regularly used in generative models to measure the quality of the generated samples compared to the original distribution. Equation (5.4) shows the computation, where P_r is the real-textured equivalent dataset, with mean and covariance (m_r, C_r) and P_{aug} is the augmented domain randomized synthetic dataset with mean and covariance (m_{aug}, C_{aug}). The real-texture or real-textured equivalents are defined as the original textures for the object of interest, for example, using the original texture of the Cheez-it box as shown in the object of interest at the top of Figure 5.3.

$$d^2((m_{aug}, C_{aug}), (m_r, C_r)) = ||m_{aug} - m_r||_2^2 + \mathrm{Tr}(C_{aug} + C_r - 2(C_{aug}C_r)^{\frac{1}{2}}) \tag{5.4}$$

We subsequently modify the WGAN-GP's discriminator to take the input shape of each of the feature vectors tested. Using FID means we no longer need to train a discriminator to estimate the distance between distributions. However, FID assumes that the distributions are Gaussian and must process the entire dataset to estimate the covariance matrices, which can be computationally expensive for large feature vectors.

In the feature space, we estimate the WD and FID based on features extracted from the Conv5 block in a ResNet-50 [71] model. Fig. 5.2 shows the ResNet-50 backbone with the Conv5 block, and Fig. 5.3 shows the proposed approach for quantifying the distances between distributions. When exploring the feature space, we use ResNet-50 as the feature extractor as it is a widely accepted and robust model for this use case [71].

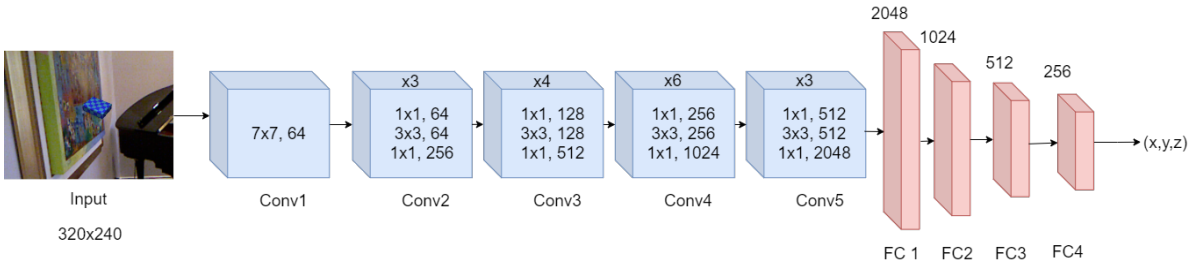


Figure 5.2: The localization network using a ResNet-50 backbone to predict the position x, y, z of the object of interest [71].

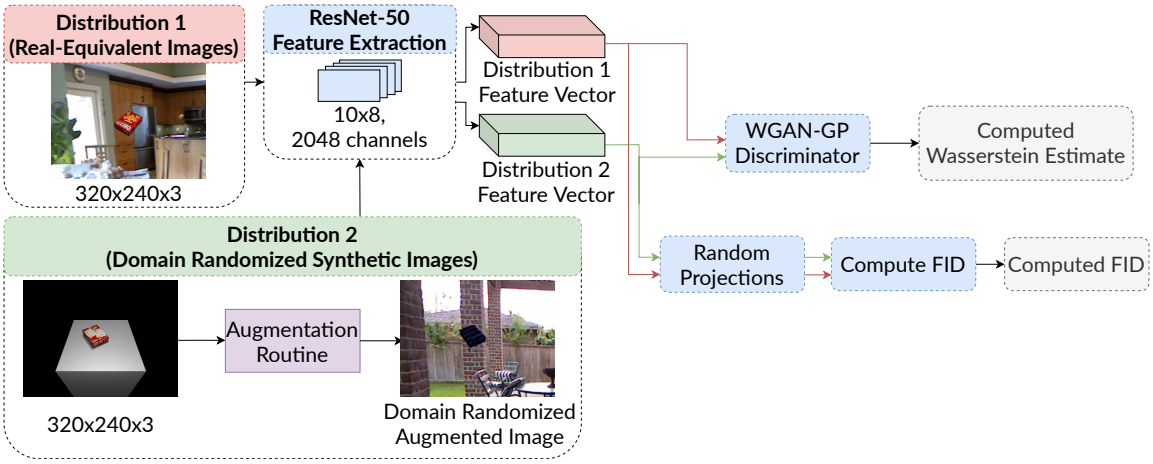


Figure 5.3: The flow of data for QDRNet. Using two data distributions, real-equivalent images and domain randomized synthetic images, we extract the features using a ResNet-50 network [71]. The extracted feature vectors are passed through a WGAN-GP critic to compute the Wasserstein distance using the standard loss function. An additional step of dimensionality reduction is used to reduce computational cost before computing FID.

5.2.5 Localization Task

To validate the ranking generated by the Wasserstein distance and FID, we train an object detector to localize an object in a scene in its 3D spatial position, (x, y, z). In the image space, we replicate the VGG-16 architecture from Tobin et. al. [187, 202] and in the feature space, we use a modified version of the ResNet-50 [71] architecture, as shown in Fig. 5.2. The standard convolutional layers are used, with the addition of three fully connected layers. We use the mean squared error (MSE) loss between the predicted object positions and ground truth using the Adam optimizer [97] with a learning rate of $1e-4$.

5.3 Data Generation

The data used for the experiments were generated using a custom physics-based rendering engine, allowing greater control of parameters during experimentation. The following section describes the data generation routine for the multiple datasets that were used during this work.

First, we generate toy datasets of images of shades of blue, representing our two data distributions. Subsequently, we develop more complex scenes using objects from the YCB dataset [23] – a robotics benchmarking dataset that contains real-world household objects and their equivalent mesh models and high-resolution scans. Real-world objects from the YCB dataset is shown in Figure 5.4 The inclusion of meshes and their real-world texture allows for greater flexibility of experimental design, as we can directly compare the effects of modifying the original object's texture. Below, details the methodology for producing these datasets.

Figure 5.4: Real-world objects from the YCB dataset taken from [23]. Each of the real-world objects has a corresponding mesh and high resolution texture associated with it.

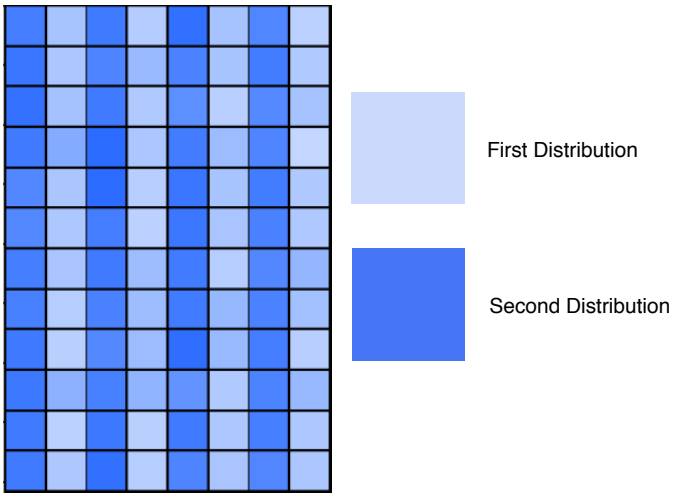


Figure 5.5: Toy dataset sample distributions. The toy dataset is generated by uniformly sampling Flat RGB colors from two known Gaussian distributions in the HSV color space. The shades of blue have a fixed hue of 220 and a value of 1, while uniformly sampling for saturation. In this dataset sample, the saturation for our first distribution P has a mean of 0.3 and std of 0.05, while our second distribution Q has a mean of 0.65 and std of 0.05. The difference between means in the two distributions is 0.35.

5.3.1 Toy Dataset

The toy dataset was created as one of the most simplistic scenarios for applying our method. The dataset is generated by sampling Flat RGB colors from two known Gaussian distributions, P and Q, in the HSV color space. The hue is fixed at 220, and a value of 1 while sampling for the saturation. Our first two datasets set both Gaussian distributions to a mean 0.3 and a standard deviation of 0.05. Following this, we generate additional datasets where we increase the difference in means by an additional 0.05. For example, if P has a mean of 0.3, Q would have a mean of 0.35 for the next dataset, then a mean of 0.4 following that, and so on. A sample of the toy dataset is shown in Figure 5.5. A total of 9,600 images per dataset of size 32x32x3 were generated.

5.3.2 DR Datasets

For the DR datasets, we use a custom Matlab based simulator to render our synthetic scenes.

Camera

Each scene configuration comprises a fixed, stationary camera position at roughly 0.9m from the center of the table of size 0.6m x 0.6m. The camera intrinsics are modeled after a RealSense d435, as this would enable us to conduct further experiments in the real-world. The resolution was set to 640x480 and further downsampled to 320x240.

Object and Poses

The experiments use the Cheez-It box as the object of interest, primarily due to the size, shape, and texture complexity. The shape of the object allows us to create scenes flexibly, ensuring our dataset consists of a wide range of poses present in a practical robotics-based application. The

Cheez-it box poses are generated by first Uniformly sampling 6 primary orientations for the box. These 6 primary orientations are shown in Figure 5.6.

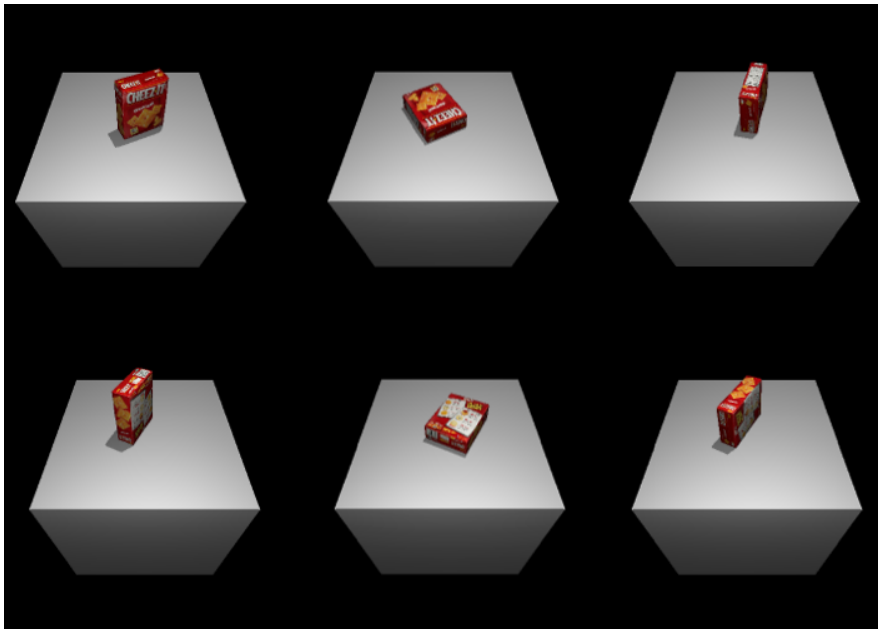


Figure 5.6: Sample Images from the dataset generation routine. Synthetic images of the object of interest are positioned around the center of the table.

We sample both rotational and translational components from a Gaussian distribution with $\mu = 0$ and $\sigma = 0.05$ m from the center of the table. A histogram of the subset of 5000 images is in Figure 5.7. These poses are re-used across all experiments to ensure a fair comparison between datasets and their relevant textures applied.

Illumination

The lighting remains fixed across all scenes and is based on the Phong [153] shading model and is a combination of ambient, diffuse, and specular illumination. Illumination is from two point lights sources, one directly above the table and another from the camera's perspective.

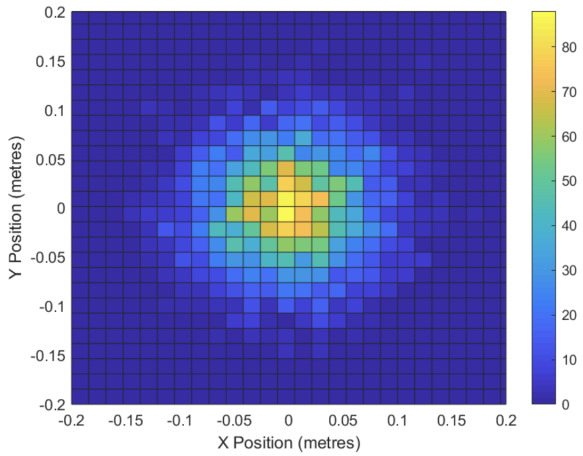


Figure 5.7: 2D-Histogram showing the x, y positions of the object of interest around the center of the table. The histogram corresponding to the dataset consists of 5000 samples.

5.3.3 Texture Randomization Routine

The textures applied are derived from commonly used textures in literature. Table 4.3 highlights the wide range of potential texture randomizations being utilized in this domain. We replicate all of the texture randomization methods from Table 4.3, and generate a dataset using the above scene configuration while modifying the textures using their texture coordinates. Figure 5.1 shows samples from the textures that are applied, and the complete data generation pipeline is outlined in algorithm 3.

Algorithm 3 Texture generation routine

Require: Texture to generate $t \in$ {Flat, Gradient, Checkerboard, Striped Zig-Zag, Perlin}, size of the texture to generate (s_1, s_2), uniform distribution to sample colors $U(0, 1)$, number of textures to generate *n*

for *i* = 1, ..., *n* **do**

if $t =$ Perlin **then**

$P_{noise} =$ {sin, cos, arcsin, clouds}

$p_{appliedNoise} \sim U(P_{noise})$

Apply $p_{appliedNoise}$ to a previously generated texture t

else

Generate blank texture $colour_1$ of size $(s_1, s_2, 3)$

Sample $(hue, saturation, value) \sim U(0, 1)$

if $t =$ Flat RGB **then**

Fill blank texture with sampled $(hue, saturation, value)$

else if $t =$ Gradient RGB, Checkerboard, Zig-Zag, Striped **then**

Sample $(hue_2, saturation_2, value_2) \sim U(0, 1)$

Fill blank texture $colour_1$ with $(hue, saturation, value)$

Generate blank texture $colour_2$ of size $(s_1, s_2, 3)$

Fill blank texture $colour_2$ with $(hue_2, saturation_2, value_2)$

else if $t =$ Gradient RGB **then**

Generated texture is interpolation between $colour_1$ and $colour_2$

else if $t =$ Checkerboard RGB **then**

Sample size of squares *sq* $\sim U(4, 8)$

Generated texture with *sq* squares with colours $colour_1$ and $colour_2$

else if $t =$ Striped RGB **then**

Sample number of lines *sq* $\sim U(3, 8)$

Generated texture with *ln* lines with colours $colour_1$ and $colour_2$

else if $t =$ Zig-Zag RGB **then**

Sample size of zig-zags *zg* $\sim U(3, 8)$

Generated texture with *zg* lines with colours $colour_1$ and $colour_2$

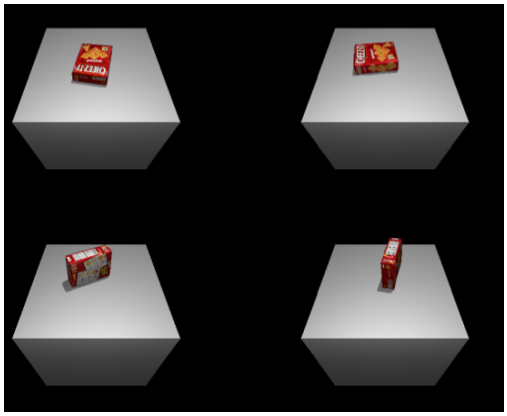
end if

end if

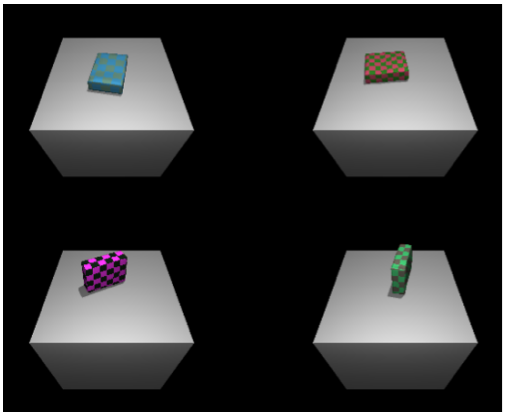
end for

5.3.4 Static Background

The static background dataset consists of a Cheez-It box in various positions around the centre of the table, with a fixed black background. This is the most simplistic case as the only variation across scenes are the positions of the box on the table. We generated a total of 20,000 images split across training/test sets for the two distributions to be measured. Samples of this dataset can be seen in Figure 5.8



(a) Real texture applied to the meshes on a static black background



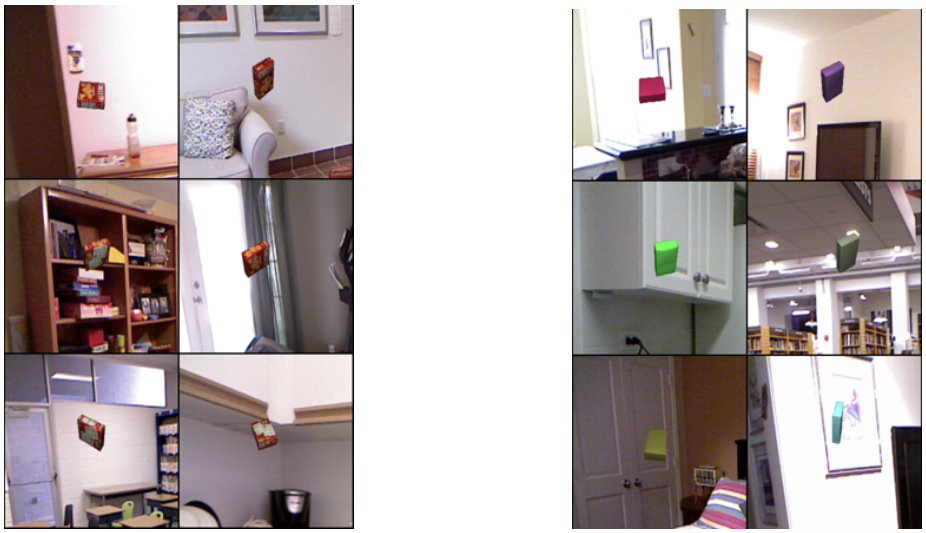
(b) Checkerboard texture applied to the meshes on a static black background

Figure 5.8: Sample data from the black background dataset. The dataset consists of the real texture applied to the object of interest on the left and the domain randomized (DR) version on the right. The black background, table, and illumination are fixed. Object poses and textures are randomized.

5.3.5 Real-world Backgrounds

The realistic backgrounds dataset consists of the same 20,000 images previously generated in the black background dataset, with the exception of replacing the static backgrounds with each scene containing a unique real-world image from the NYU Depth V2 dataset [142]. The NYU Depth V2 dataset is a large-scale dataset of indoor real-world RGB images at a resolution of 640x480. These images were used as a background for this dataset to investigate the influence of

unique backgrounds per scene. We expect the change in background to force the network to learn information about the foreground rather than the background. Sample images from this dataset can be seen in Figure 5.9



(a) Real texture applied to the meshes on unique real-world backgrounds

(b) Flat RGB texture applied to the meshes on unique real-world backgrounds

Figure 5.9: Sample data from the real-world backgrounds dataset. The previous black backgrounds were replaced with real-world images from the NYU Depth V2 dataset [142]. The real texture versions appear on the left, and the domain randomized (DR) versions are on the right.

5.4 Experiments

This section describes the experiments conducted in the image and feature space. The section opens with experiments conducted on the toy dataset described in section 5.3.1, where an investigation on the potential for using statistical distance estimates in the most simplistic case is conducted. Following this, further experiments in the image space using statistical distance estimates and task on the static background dataset and real-world backgrounds described in sections 5.3.4 and 5.3.5. The section concludes with experiments in the feature space, investigating the influence

on the features used and quantifying the differences in distributions.

5.4.1 Image Space

Toy Dataset

To evaluate the potential for using statistical measures to rank different textures currently being used within DR literature, both JSD and WD estimate are explored in a simple scenario. The dataset used is described in further detail in section 5.3.1. Starting with two known Gaussian distributions P and Q, where $\mu = 0.3$ and $\sigma = 0.05$ for both P and Q such that $P = Q$. The next trial fixes distribution P to remain at $\mu = 0.3$ and $\sigma = 0.05$, while Q now is defined with $\mu = 0.35$ and $\sigma = 0.05$. This separation between P and Q continues 10 additional times, with each trial increasing the μ of Q by 0.05.

The two statistical distance measures evaluated are JSD and WD. In the case of JSD, the vanilla GAN is used with the only modification being replacing the generated samples from a Generator G with image samples from distribution Q from the toy dataset [64]. The same intuition follows for WD, where WGAN-GP is used to estimate the distance between P and Q, and a similar modification is made to replace generated samples with image samples from distribution Q [68].

Static Backgrounds

Building from the previous experiment, a more representative dataset of a robotics scenario is explored, where objects of interest are placed roughly around the center of the table. In this case, the static backgrounds dataset is used as outlined in section 5.3.4. A similar experiment is conducted on the images from the static background dataset in the image space. Here, the same modification for WGAN-GP is used. The critic's inputs are the real-texture (unmodified) synthetic images and images from a dataset consisting of one of the aforementioned texture randomization

methods. The WD estimate is used to measure the difference between the two distributions. The experiment is repeated for each of the texture randomization methods explored to analyze the impact of varying the object of interest's textures.

To evaluate how the above WD estimate for each texture randomization method affects task performance, an object localization task to predict the object's 3D position on the table is applied. After replicating the VGG-16 network architecture from one of the original implementations of DR [202], the model is trained using the static backgrounds datasets.

Real-world Backgrounds

In some instances, deep learning models are biased towards background appearance rather than the object of interest [199], particularly when the image is treated holistically, such as in the current scenario outlined in previous experiments in the image space. To address this concern, the previous experimental setup using static backgrounds is repeated using real-world backgrounds from the NYU Depth V2 dataset described in section 5.3.5. The introduction of a unique background per image in the dataset, would reduce the influence of the background-bias problem.

5.4.2 Feature Space

Deep learning models are commonly bootstrapped using pre-trained weights that are available for a variety of networks. These pre-trained networks make it possible to quantify the distance between distributions in the feature space using existing networks that have already been trained on large amounts of data. We conduct the subsequent experiments in the feature space described in the following subsection.

Real-world Backgrounds

Using the feature extractor network's appropriate features, the previous experiment conducted on real-world backgrounds in the image space is repeated. This gives a comparison between previous image space approaches and those in the feature space.

Additionally, FID is computed between the selected features using the method detailed in section 5.2.4. Given the size of the representation and dataset, directly computing FID on the features is computationally expensive. For that reason, an additional step of dimensionality reduction is performed to reduce computational expense. The dimensionality for the $3,000$ samples is reduced using 10 random projections to get the FIDs' variance, note that this may cause some loss of information. Each of the datasets used for this method uses the same 10 projection matrices, ensuring a fair comparison between the different texture randomization methods.

5.5 Results and Discussion

5.5.1 Image Space

Toy Dataset

Starting with the JSD estimate in Figure 5.10a, the plot shows the JSD estimate against the hand-calculated ground truth estimate computed using Equation 5.2. The JSD estimate using the neural network appears to follow the ground truth generally. While this is promising and useful for estimating distances between different distributions in certain scenarios, such as their widespread usage in GANs, using JSD appears to saturate at $ln2$ when the two distributions are sufficiently far apart. In Figure 5.10a, this happens when the difference in the means of the Gaussian distributions is 0.3 and above. I.e, if the first distribution $P = \mathcal{N}(\mu = 0.3, \sigma^2 = 0.0025)$ and the second

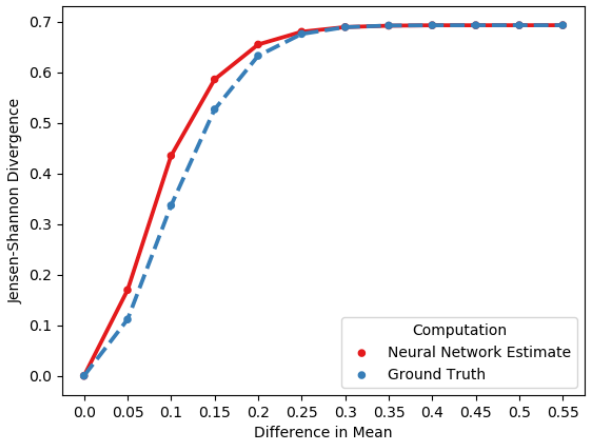
distribution is $Q = \mathcal{N}(\mu = 0.6,\, \sigma^2 = 0.0025)$, using JSD would produce an estimate of $ln2$ when the difference in μ for P and Q is $\mu \geq 0.3$. In scenarios where the two distributions are known and not fat apart, JSD would be a useful measure for quantifying the differences in distributions. However, the cases most applicable in this work involves unknown distributions, where it would not be possible to guarantee a small difference in distances between distributions beforehand.

A possible alternative solution would be using WD to estimate instead. As seen in Figure 5.10b, the plot shows the WD estimate monotonically increasing as the difference in μ between P and Q continues to increase. This outcome is desirable as the scenarios encountered will predominantly be unknown distributions, where there are no guarantees the distance between the two distributions are small. Based on this experiment's results, it would be clear to rank the distance between P and Q using the WD estimate. An additional observation is the estimate's smoothness across varying the dataset's size for P and Q. There appears to be less variance in the estimate when using a toy dataset with at least $1,000$ images for P and Q, respectively. It is worth noting that despite the higher variance on smaller dataset sizes, the WD estimate is still monotonically increasing as the distance between P and Q is increased.

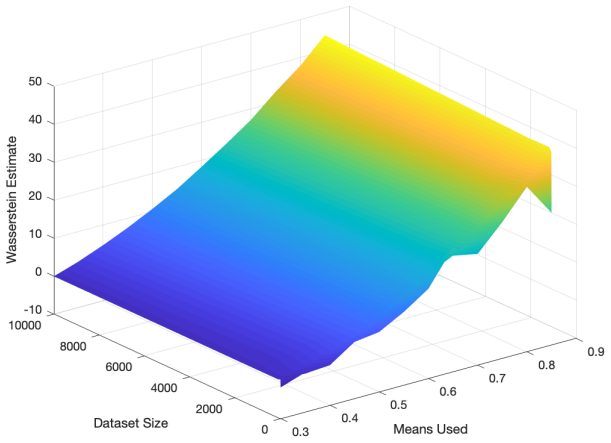
Given the nature of the current work, JSD would not be a suitable choice for estimating and ranking the difference between different distributions. Future utilization would be more complex than the current toy dataset in use. There may be situations where the JSD estimate would be equal despite the two distributions being farther apart. A more appropriate approach would be to use the WD estimate, which monotonically increases as the distance between two distributions increases.

Estimating Distance - Static Backgrounds

The results from increasing the dataset complexity by using the static black backgrounds dataset to evaluate the use of the WD estimate is seen in Figure 5.11. Here, it is evident that there are three distinct groupings between the various texture randomizations currently used within the existing literature. These groupings can be classified as patterned, non-patterned, and significant noise.



(a) JSD estimates when evaluated on a toy dataset saturates when the distance between distributions P and Q are sufficiently far apart using the full $9,600$ images sampled from distribution P and $9,600$ images sampled from distribution Q.



(b) Contrary to JSD, WD estimate monotonically increases as the means used for distribution Q increases after each trial. The size of the dataset is also examined, and this method of computation appears to stabilize when using a toy dataset with at least $1,000$ images from each distribution.

Figure 5.10: Comparison of JSD and WD estimates using a toy dataset shows that WD estimate provides a more practical way of quantifying separations between distributions when they are far apart.

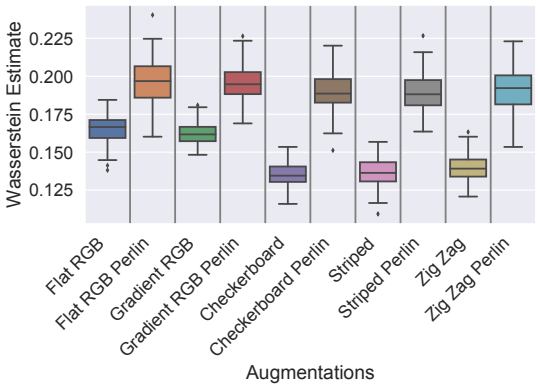


Figure 5.11: Figure showing the WD estimate using commonly used texture randomization techniques. We compute the estimate between real-equivalent synthetic and DR synthetic RGB images with black backgrounds. There are three distinct groupings between patterned (Checkerboard, Striped, Zig-Zag), non-patterned (Flat RGB and Gradient RGB) and dominant noise (Perlin).

In the case of patterned: checkerboard, striped, and zig-zag appear to result in the lowest WD estimate when measuring the distance between a randomized texture and the real-equivalent texture. This alludes to patterned textures being closer to the real-equivalents underlying distribution. The second grouping involving non-patterned: flat RGB, and gradient RGB, resulted in the second-highest estimates, while significant noise (all previous patterns with Perlin noise added) yielded the highest estimates. This means that the non-patterned and significant noise textures are farther away from the real-equivalent texture.

While the results indicate clear separations in distances between patterned, non-patterned, and significant noise relative to the real-equivalent distribution, at this stage, it is unclear whether the various estimates would correlate to higher task-based performance.

Localization Task - Static Backgrounds

Figure 5.12 presents the results to verify the rankings of patterned, non-patterned, and significant noise produced by the WD estimate is preserved when solving a localization task using the static black background dataset.

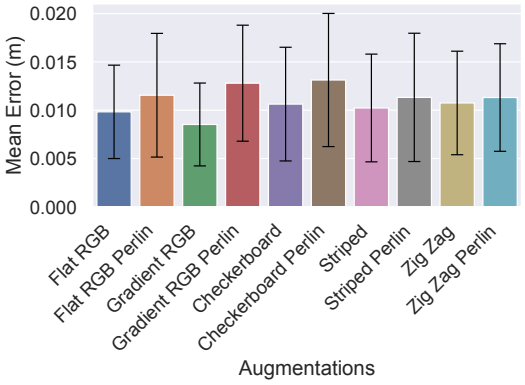


Figure 5.12: Figure showing results from a localization task where the model was trained on DR synthetic images, and evaluated on real-equivalent synthetic images with black backgrounds. The MSE is between the predicted and ground truth positions of the object on the table.

Despite there being differences in the mean error across the different texture randomization methods, the variance of each result is too high to conclude that ranking WD estimates reflect actual task performance. Based on this, actual task performance in the static black background behaves similarly regardless of the texture randomization technique applied. One possible explanation for this is there may not be challenging samples existing in the static black background dataset. Apart from the textures applied to the object of interest, all other variables remain the same. One possible solution to increase the complexity would be introducing real-world backgrounds per sample in the dataset. This would ensure that limited background information is used to identify objects across samples.

Wasserstein Estimate - Real-world Backgrounds

The same WD estimate experiment is repeated with the distinction of using real-world backgrounds in the dataset. Figure 5.13 highlights the results from the experiment. Note, the only difference between the static black background dataset and the real-world dataset is that a unique real-world image is used for the background.

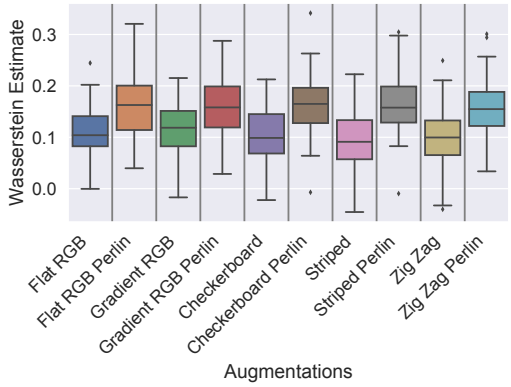


Figure 5.13: Figure showing WD estimate for various texture randomization techniques when operating in the image space. There are no clear separations in randomization techniques in the image space using real-world backgrounds. We are only able to differentiate between methods involving non-Perlin and Perlin noise.

Here, there seems to be a less obvious ranking for the texture randomization methods. There is a distinction between textures with and without noise, though there is a significantly higher variance for each of the results. The introduction of varying real-world backgrounds, which would be more representative of challenging scenes in the wild, increased the difficulty in attaining a clear ranking between the different texture randomization techniques.

The holistic nature of using raw RGB images in this approach may be too complex for solving this particular problem. Operating on raw RGB images appears to be too high level for this particular approach, and fine-grained or low-level features extracted via a feature extractor would be an option.

5.5.2 Feature Space

Wasserstein Estimate

Using the same real-world background dataset and operating in the feature space using features extracted from a ResNet-50 using pre-trained ImageNet weights, clearer separations are seen in Figure 5.14. Unlike the WD estimates seen in Figure 5.13 using the same dataset but operating in the image space, there is now a clear ranking between the different texture randomization techniques applied to the object of interest. Note that the values between the image space and feature space experiments are not directly comparable, as the measurement is on different underlying distributions. It is more important to compare how the two computations in the image and feature space differ based on the presented rankings.

The clearer rankings indicate the highest estimates are produced by non-patterned textures (flat RGB, gradient RGB), while the lower estimates result from patterned textures (striped, zig-zag, checkerboard). In the case of the addition of Perlin noise, it generally appears to lower the WD estimate of the corresponding texture. For example, both flat RGB and gradient RGB appear to exceedingly reduce the WD estimate. This appears to suggest that more complex patterns reduces the distance between the textured and real-equivalent distributions.

However, there are some instances where the addition of Perlin noise increases the estimate. For example, checkerboard Perlin results in a slightly higher estimate than it's checkerboard counterpart. Similarly, zig-zag Perlin also follows, resulting in a higher WD estimate than zig-zag. One possibility is that the addition of significant Perlin noise to existing patterned textures increases the difficulty in discerning the original texture applied.

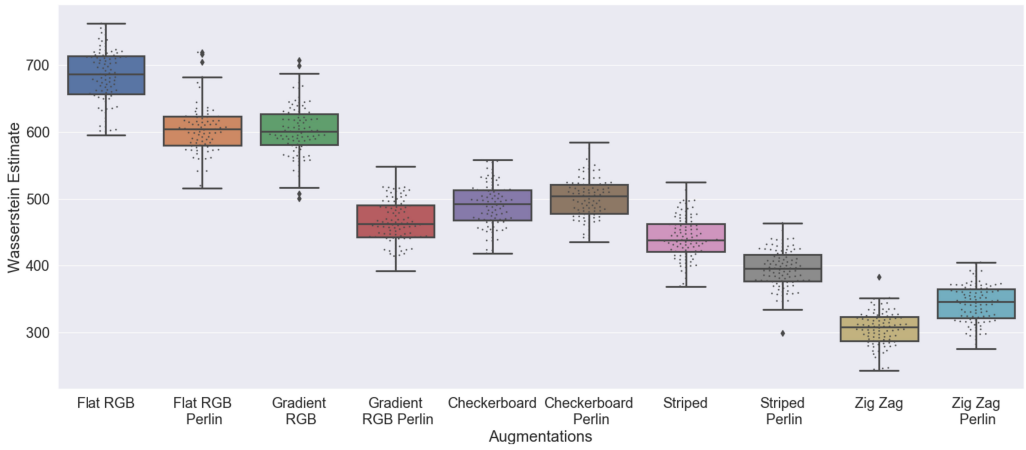


Figure 5.14: WD between real-equivalent and DR synthetic data with backgrounds from NYU V2 dataset [142] when operating in the feature space. The distance is measured using feature vectors extracted from a ResNet-50 backbone [71]. When working in the feature space, we can more clearly distinguish between the various texture randomization techniques.

FID Estimate

In the case of FID, there appears to be a significantly lower variance in the results than the WD estimate, as seen in Figure 5.15. A similar ranking holds, where patterned textures resulted in much lower estimates compared to non-patterned.

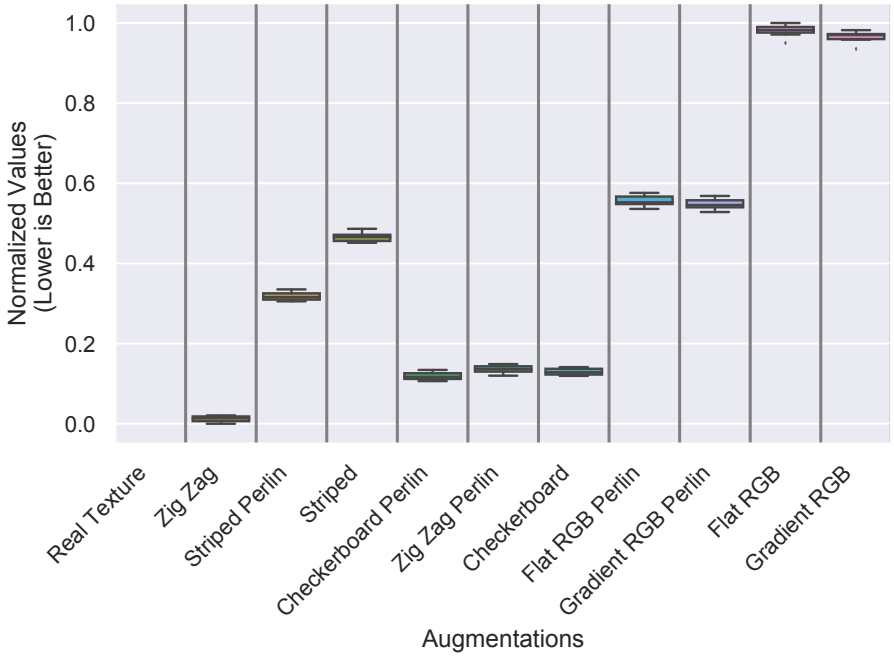


Figure 5.15: FID estimates using the real-world background dataset.

The addition of Perlin noise generally lowers the FID estimates in all cases apart from zig-zag Perlin. The increase in the distance between zig-zag Perlin and the real-equivalent texture compared to zig-zag is similar to the ranking produced by the WD estimate.

It is worth noting that using FID on the extracted features can be computationally expensive depending on the feature vector distributions used. In this scenario, an additional step of dimensionality reduction was required to allow the current hardware to compute the FID. This can be mitigated with a larger RAM capacity to perform the computation more accurately.

Localization Task

Supporting the previously produced WD and FID estimates ranking via a localization task, Figure 5.16 shows a combined plot of the localization task error, WD, and FID estimates, sorted by the mean localization task error.

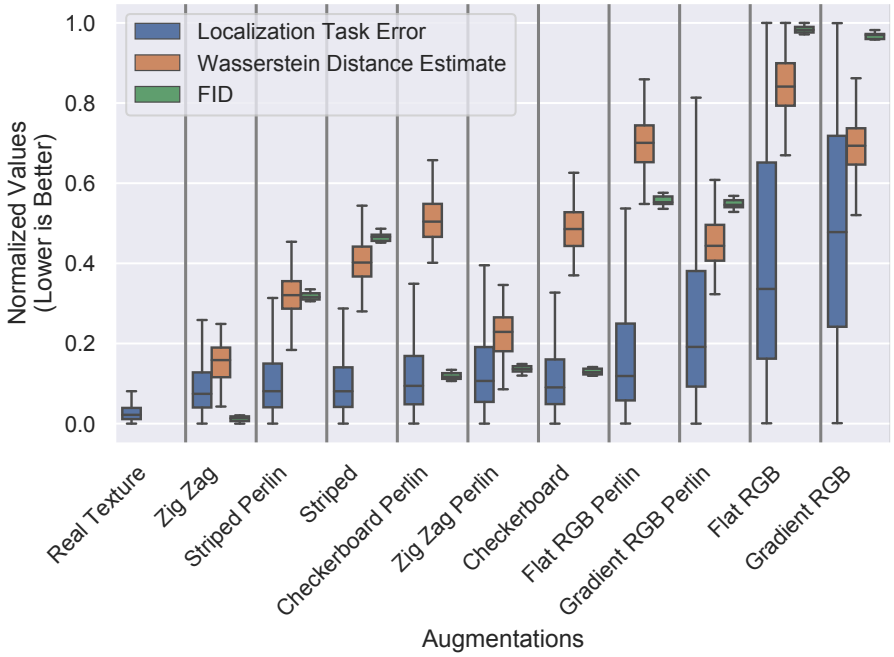


Figure 5.16: Figure illustrating the comparison of localization task, WD, and FID estimate in feature space. The values are normalized and sorted by the lowest mean error in the localization task. The effects of additional noise (Perlin) increase the difficulty in obtaining a clear ranking. In general, the addition of dominant Perlin noise appears to aid performance, in addition to using patterned textures.

Based on WD and FID's produced ranking, there is more apparent evidence of a correlation between estimating the distance between the texture randomization methods and task-based performance. Generally, as the distance between the real-equivalent texture and applied texture randomization method increases, the localization task error increases. This indicates the importance of selecting a more favorable texture randomization method closer to the original distribution.

For example, patterned textures such as zig-zag, striped, and checkerboard yield higher performance than non-patterned counterparts (flat RGB and gradient RGB). Interestingly, flat RGB results in one of the worst performers, despite being the most commonly used texture randomization, as shown in Table 4.3. The selection of flat RGB in the literature may be due to ease of generation, as these textures can be created relatively quickly programmatically.

There are some instances where the distance measures do not clearly predict task-based performance, such as Striped or Checkerboard. While we can see a trend that more complex patterned textures outperform non-complex textures such as Flat RGB or Gradient RGB, we would have to investigate further the approach on different datasets with other objects of interest. The additional data to evaluate the method may lead to further insight into how the distance estimates are related to task-based performance.

Significant Perlin noise is also an interesting scenario, where it appears to aid task-based performance in all textures, apart from zig-zag Perlin. The addition of Perlin noise is particularly useful for non-patterned textures when solving the localization task.

For both WD and FID estimates, there is a clear distinction between patterned and non-patterned textures reflected in task-based performance. A statistical z-test was performed to evaluate the statistical significance of the results. A null hypothesis that the mean is equal between two texture randomization methods at $\alpha = 0.05$. The z-test results in a p-value < 0.05 for striped Perlin and zig-zag, and a p-value < 0.001 for the remaining texture randomization pairs.

5.6 Conclusion

This chapter presents a novel method for measuring the differences in data distributions between a small amount of synthetic and real-equivalent data. Across all commonly selected DR texture methods, we quantified the differences in the data distributions in the feature space. We found

clear rankings between patterned and non-patterned textures when ranking methods using both the Wasserstein and FID estimates against localization-task error. Our findings suggest that a low estimate using Wasserstein or FID is associated with a lower task-based error.

We demonstrate that by using a more complex patterned texture with a lower Wasserstein of FID estimate such as Zig Zag, we achieve lower task-based error on a localization task. Generally, more complex patterns such as textures are more desirable and aid task-based performance. However, the use of Perlin noise makes it more challenging to obtain a clear ranking, especially when using FID. The rankings for FID estimates in patterned and non-patterned are comparable, which could be due to Perlin noise covering the original texture pattern we applied to it.

Our approach is an effective means of predicting final task-based performance with good agreement using Wasserstein and FID estimates, without the need for the expensive cost of task-based training and evaluation using a small amount of data. Future work could investigate mixing different textures with different ratios, for example, using a mix of complex textures or a mix of complex and non-complex textures. Texture diversity may increase when using such combinations. In scenarios where comparable complex textures are used, we suspect a similar behavior would arise to the complex textures that have been explored. However, it would be interesting to see if a mix of complex and non-complex textures would follow a similar trend when quantifying the differences between the distributions and the ranking compared to task-based performance. Further analysis can also be explored in correlating distances between augmentations, which would require additional datasets of varying scenes and objects of interest.

Our work has so far focused on scenes similar in complexity to several DR works, which use simple geometric shapes or single objects [81, 127, 147, 155, 201, 202, 219, 232] for solving a given task. However, in Chapter 4, we have seen that DR is used for solving other vision and robotics tasks and is unclear as to how the concept translates to different scenarios. For example, would DR behave similarly when solving pixel-level tasks such as semantic segmentation? Or how would DR behave in more complex scenes involving multiple object classes, varying levels

of occlusion, and clutter?

In the following Chapter 6, we present a new DR dataset focused on complex scenes for solving vision tasks such as pose estimation, object detection, object segmentation, and depth estimation. The dataset replicates real-world scenes to enable investigations into cross-domain settings, which allows us to address our second question in the thesis - how DR behaves across various tasks? Making use of the dataset in Chapter 6, we present our work in the generalizability of DR across object detection and semantic segmentation tasks in complex scenes in Chapter 7.

Chapter 6

SRDR Dataset: Sim-to-Real Domain Randomized dataset for Benchmarking Tasks in Visual Sim-To-Real Transfer

This chapter presents a new matched synthetic, real, and DR dataset for visual scene understanding such as object detection, semantic segmentation, or 6D pose estimation called the SRDR dataset. The SRDR dataset replicates real-world scenes from the YCB-M dataset [67], which contains 20 household objects from the YCB dataset [23]. We replicate all real-world scenes from an Intel RealSense R200 camera from the YCB-M dataset using the most commonly applied texture DR methods and combine the replicated scenes with backgrounds with varying visual and compositional complexity levels. Due to the replicated scenes across synthetic, real-world, and DR data, researchers can use the unique dataset to facilitate cross-domain, training, evaluation, and comparison studies. 6D poses, per-pixel segmentation, 2D and 3D bounding boxes, and depth maps are provided for each RGB image in the dataset. We use the SRDR dataset for evaluations on detection and segmentation tasks in Chapters 7 and 8.

6.1 Introduction

Visual scene understanding in computer vision and robotics typically relies on approaches requiring vast annotated data across multiple object classes. This data can sometimes be laborious to gather and annotate under different conditions. Synthetic data and DR have been rising in popularity due to the ease and control of generation and freely available annotations. The use is beneficial in scenarios where acquiring the annotations is more tedious, such as semantic segmentation [22, 54], optical flow [128], or stereo vision [241]. In semantic segmentation, pixel-level labels are required per object class, which can be time-consuming to annotate, particularly in cluttered and occluded scenes where it is more challenging to label manually.

Several researchers have been using synthetic data in various ways, where some choose to use synthetic data solely [171, 211], others using a combination of synthetic and real-world data [192, 208, 238] or using DR data [81, 147, 180, 202, 232] to train models for solving the above tasks, and function on real-world data. However, few datasets accommodate the broad use of synthetic data for a particular problem, such as combining synthetic, real-world, or DR data. For example, of the most commonly used object-centric synthetic datasets from Table 6.1, none provide matched real-world scenes and DR versions. In contrast, we have seen urban driving datasets such as VKITTI by Gaidon et al. [54] which contains matched real-world data from the KITTI dataset [57], enabling researchers to perform comparative evaluation and training of models in the autonomous driving settings under different domains.

This chapter introduces the Sim-to-Real Domain Randomized dataset (SRDR) dataset, a large matched synthetic, real-world, and DR dataset containing 291K frames using realistic household objects for training and evaluating models for visual scene understanding in vision and robotics. As shown in Figure 6.1, for each real-world image from the Intel RealSense R200 camera from YCB-M dataset [67], we generate synthetic and DR versions for each texture type in the current literature and five unique environments. Annotations contain segmentation masks, depth maps, 6D object pose, and 2D/3D bounding boxes for each object in the scene. Furthermore,

we expand on the tools provided by To et al. [200] to allow researchers to generate their own DR datasets using each texture type in the current literature from their existing annotated real-world datasets. We envisage the dataset as a natural fit for researchers involved in using synthetic data who are interested in combining data from different domains, using all commonly applied texture DR methods, and in unique environments. Further examples from the dataset are shown in Figure 6.2.

Alongside the SRDR dataset, we also introduce the SRDR plugin. The SRDR plugin is a program that extends the software by To et al. [200], which enables the generation of synthetic scenes using Unreal Engine [49]. The SRDR plugin allows practitioners to finely control the creation of DR scenes by allowing users to re-create scenes as standard by providing scene description files to reproduce DR datasets. This plugin would allow users to create scenes such as those presented in Figure 6.2, which is not a standard feature available in the existing tools.

We outline the main contributions of our dataset below:

- We create a large DR dataset containing 291K frames using realistic household objects that are widely used in robotics and vision benchmarking [23]. We expand upon the dataset by Grenzdörffer, Günther, and Hertzberg [67] by taking images from the dataset and generating DR versions for each texture type in current literature and five unique environments with varying scene complexity
- To our knowledge, this is the first dataset to contain DR data using the most commonly applied texture randomization techniques, and matched real-world and real-textured synthetic data, allowing researchers to perform exhaustive comparisons, evaluation, and training using DR techniques in current literature, particularly in cross-domain synthetic-to-real settings.
- We built upon the tools from To et al. [200] to enable DR for existing labeled real-world datasets and provide new tools for researchers to create DR versions of their own real-world datasets.

Figure 6.1: We created the Sim-to-Real Domain Randomized (SRDR) dataset by taking real-world images from the YCB-M dataset [67] (top left) and matching 3D household models [23] (e.g., gelatin box, cracker box, meat can, and tuna can) to their positions in the real-world. We generated matched synthetic (top middle) and DR synthetic (top right) versions of the real-world, against five unique environments. Each scene (real-world, synthetic, or DR synthetic) contains pixel-wise segmentation of objects of interest (bottom left), 2D/3D bounding box coordinates (bottom middle), and depth images (bottom right). The camera positions and 3D positions of each object of interest are also provided.

(a) Synthetic images (b) DR Synthetic images (c) Real-world images

Figure 6.2: We show sample images from the SRDR dataset displaying synthetic (left column), DR synthetic (middle column), and real-world images (right column). The DR images use all the most commonly applied texture randomization techniques in the existing literature and were using real-world backgrounds from the Active Vision dataset [4].

6.2 Related Work

This section comprises two parts: the first investigates current synthetic datasets in computer vision and robotics, the second explores the existing tools used to generate the data.

6.2.1 Datasets

The use of synthetic datasets has grown tremendously since 2016. Table 6.1 summarizes the key synthetic datasets in use for a wide variety of applications. Yet, despite the increase in popularity of DR, very few publicly available datasets incorporate this or are limited in scope. For example, the SIDOD dataset [80] only selects textures from a collection of 7 textures, while the SRDR dataset contains 10,000 textures per randomization technique. Furthermore, the texture randomizations are performed only on the distractor objects in the SIDOD dataset. In contrast, the SRDR dataset provides various combinations of textures ranging from commonly applied DR methods to the real-textured equivalents depending on the use case.

The FlyingThings3D dataset [128] does provide full texture randomizations for the various objects from a more extensive collection of procedurally generated images, landscapes, texture-style photographs from ImageAfter, and photographs from Flickr. However, it is unsuitable for pose estimation or scene understanding, as the dataset does not have any notion of semantics, as they are using random labels across objects and scenes. Additionally, the dataset re-uses 200 static backgrounds, decreasing the background diversity available for solving desired tasks.

During the creation of scenes using SUNCG using the Planner 5D platform [53, 191], users could select positions of objects and textures to create realistic indoor scenes. However, it does not employ typical DR textures, nor does it replicate real-world settings, which the SRDR dataset provides. While there are synthetic datasets that do reproduce real-world scenes that are useful for solving scene understanding, such as the VKITTI or VKITTI 2 dataset [54, 57], the dataset does

Dataset	Purpose	Objects	Description	# Objects	# Frames	# Environments	Real Scenes	Domain Randomized	Distractors	2D bbox	3D bbox	Segmentation	6D Pose	Occlusion
SIDOD [80]	Pose Estimation Scene Understanding	YCB	Household	21	122k	3	✗	✓ [1]	✓	✓	✓	✓	✓	✓
Falling Things (FAT) [204]	Pose Estimation Scene Understanding	YCB	Household	21	60k	3	✗	✗	✓	✓	✓	✓	✓	✓
ObjectSynth [74]	Pose Estimation Scene Understanding	RU-APC LineMod	Household	15 & 14 [2]	400k	6	✗	✗	✗	✓	✓	✓	✓	✓
FlyingThings3D [128]	Optical Flow	ShapeNet	Various	20	39K	200	✗	✓	✗	✗	✗	✗	✗	✗
SceneNet RGB-D [132]	Scene Understanding	ShapeNet	Indoor Scenes	Varies	5M	57	✗	✗	✗	✗	✗	✓	✓	✓
SUNCG [191]	Scene Understanding Indoor Navigation	SUNCG	Indoor Scenes	Varies (10 classes)	40k	Varies	✗	✓	✗	✓	✓	✓	✓	✓
SRDR Dataset	**Pose Estimation** **Scene Understanding**	**YCB**	**Household**	**20**	**291k**	**5**	✓ [4]	✓	✓ [3]	✓	✓	✓	✓	✓

Table 6.1: Table showing related object-centric synthetic datasets. The SRDR dataset is highlighted in bold.

[1] Only distractor objects are texture randomized from a selection of 7 textures.

[2] 15 objects are from LineMod dataset, and 14 from Rutgers APC (RU-APC) [169].

[3] Distractors are in the form of background objects from different categories.

[4] SRDR dataset replicates scenes from the YCB-M [67] dataset using RealSense camera.

not use DR on object instances and instead replicates data from the KITTI dataset under various conditions such as varying the weather. A primary motive for the SRDR dataset and plugin is to consolidate a collection of commonly applied DR methods, unique backgrounds, and incorporate real-world scenes to answer research questions.

6.2.2 Simulators

Many different software packages enable researchers to generate high-quality data for various purposes, as shown in Table 6.2. More recently, game engines such as Unreal Engine [49] or Unity Game Engine [197] are used as a starting point for further developing tools to generate desired data [22, 44, 204]. Modern game engines reduce the complexity towards the creation of photorealistic scenes, more recently with the use of ray-traced illumination and physically based rendering. Comparable to robotics and vision simulators such as PyBullet [36] or Gazebo [100], game engines also incorporate physics engines to enable physically plausible scenes, with a bonus of increased visual realism compared to traditional robotics simulators. This added bonus means the quality of the synthetic data has dramatically improved in recent years, as evident by the photorealism improvement between VKITTI [54] and VKITTI 2 [22]. While realism plays a part in transfer from synthetic to real, DR has demonstrated that highly photorealistic scenes are not the only solution to bridge the domain gap.

There is an increase in available tools for generating DR data [82, 125, 144, 204], allowing researchers to develop highly randomized scenes for use as training data. Typically, these tools provide a multitude of features such as high-quality annotations, randomization of parameters such as camera, textures, illumination, or object poses. However, some are limited in the scope of their usage. For example, Maciek Chociej [125] provides a flexible randomization routine to generate high-quality data that is based on the Unity Game Engine, although it is strictly used for in-hand manipulation with a single cube. Other simulators provide the ability to create more tailored scenes, such as Isaac Sim [144], which also includes real-time ray tracing and path tracing

Name	Engine	Description	Texture Randomization[1]	Camera Randomization[1]	Object Pose Randomization[1]	Illumination Randomization[1]	Reproducable DR Scenes	Annotations[1]
Object-centric Vision and Robotics								
NDDS [204]	Unreal Engine	Object-centric DR scenes	✓	✓	✓	✓	×	✓
ORRB [125]	Unity	In-hand robotic manipulation	✓	✓	✓	✓	×	✓
RLBench [82]	Coppelia Sim	RL-based robotics	✓	✓	✓	✓	×	✓
Isaac Sim [144]	Unreal Engine	General purpose vision and robotics	✓	✓	✓	✓	×	✓
SRDR Plugin	Unreal Engine	Generate reproducable DR scenes	✓	✓	✓	✓	✓	✓
Indoor Environments								
iGibson [225]	Custom	Physical robotic interactions in indoor household and office scenes	✓	×	×	×	✓	✓
AI2-THOR [102]	Unity	Procedurally generated indoor household scenes	×	×	✓	×	✓	✓
Sapien [227]	Custom	Indoor household scenes	×	×	×	×	×	✓
VRGym [230]	Unreal Engine	Virtual Reality human-robot interaction	×	×	×	×	×	✓
Urban Environments								
Deepdrive [39]	Unreal Engine	Urban driving scenes	×	×	×	×	×	✓
CARLA [44]	Unreal Engine	Urban driving scenes	×	×	×	×	×	✓
TORCS [224]	Custom	3D racing car simulator	×	×	×	×	×	✓
VIVID [107]	Unreal Engine	Urban environments Human and aerial drone centred	×	×	×	×	×	✓
ProcSy [95]	Unreal Engine	Urban driving scenes	×	×	×	×	×	✓
Robotics and Vision Simulators								
Gazebo [99]	Custom	General purpose robotics	×	×	×	×	×	×
MuJoCo [203]	Custom	General purpose robotics	×	×	×	×	×	×
PyBullet [36]	Bullet	General purpose robotics	×	×	×	×	×	×
UnrealCV [161]	Unreal Engine	General purpose computer vision	×	×	×	×	×	✓
Coppelia Sim [173]	Custom	General purpose robotics	×	×	×	×	×	×

Table 6.2: Overview of existing simulators allowing generating synthetic data.

[1] Denoting built-in tools for generating the data.

to generate highly realistic scenes, as well as a domain randomization routine. A typical sample scene using DR is in Figure 6.3.

Figure 6.3: Sample Image taken from NVIDIA Isaac Sim [144]. Parameters to render the scene are randomized and non-reproducible.

While the existing tools do provide a way for creating highly randomized DR scenes [82, 125, 144, 204], one drawback is the ability to create reproducible DR scenes out-of-box. All existing tools require additional modifications to enable to re-create DR scenes, making it more difficult to disentangle various features when analyzing the effects of DR. The SRDR plugin helps in the ability to re-create scenes as standard by providing scene description files to reproduce DR datasets. This feature allows the generation of scenes such as Figure 6.2, which would not be readily available without extending the existing tools.

6.3 The SRDR Dataset

As previously introduced, the SRDR dataset replicates 31 real-world scenes from the YCB-M dataset, replicating images captured by the Intel RealSense R-200 camera. For each of the 31 scenes, ten different DR texture randomizations are applied from commonly used textures in literature. Examples of these textures are in Figure 5.1 from Chapter 5. Furthermore, five different

background environments are applied for each of the ten different texture randomizations used, ranging from synthetic, real-world scenes. The SRDR dataset includes 291k synthetic frames, making it one of the larger DR object-centric dataset available, with annotations such as 2D, 3D bounding boxes, segmentation masks, 6D pose, and object visibility. Since the dataset specifically replicates scenes from the RealSense R-200 camera from the YCB-M dataset [67], researchers can also use the real-world images from the YCB-M dataset in conjunction with the SRDR dataset for cross-domain investigations.

The plugin intends to eliminate the additional programming required to replicate scenes using existing simulators. The plugin easily allows reproducing real-world settings by providing a scene description file to generate the data. Unlike the existing software to create synthetic data, no additional programming is required beyond the initial scene description files; meaning scene replication is functional out of the box. While the current SRDR dataset uses mesh models from the YCB dataset, researchers can use different meshes to replicate their scenes.

As the SRDR plugin extends software by To et al. [200], the annotation files generated by the original program allows cross-compatibility between other datasets such as the FAT, SIDOD, and all real-world scenes from the YCB-M dataset, meaning the same data processing steps can be used when combining various synthetic and real-world sources.

Unlike the original software by To et al. [200], the SRDR plugin can place actors into a scene based on a scene description file. These scene actors can include the object models, camera, or illumination positions, all with the desired randomization techniques. Of the commonly applied texture DR methods, only RGB and Gradient RGB is supported by the original software by To et al. [200], limiting researchers on the types of textures researchers may use out of the box.

6.3.1 Data Generation

This section discusses the data generation routine for replicating scenes from the YCB-M dataset, how we generated the textures, the selection of backgrounds for the synthetic scenes, and how we approximated illumination.

Texture Generation

The textures applied to the object meshes from the YCB dataset are based on existing works from Table 4.3. In Chapter 5, we saw measurable differences in task-based performance when using various DR techniques. For this reason, all ten different DR texture randomization techniques are incorporated in the SRDR dataset and include the original real-world equivalent synthetic textures. Samples from the different textures used for the various objects are seen in Figure 5.1 from Chapter 5.

Backgrounds used

There are several reasons for using various environments in the SRDR dataset. First and foremost, using distinct backgrounds increases the variety in the dataset, which is generally helpful when training DNNs. A second reason is that diverse environments can help answer different research questions. For example, how useful is using real-world backgrounds compared to synthetic backgrounds? Or do highly cluttered environments significantly reduce task-based performance? Or would varying degrees of synthetic realism influence task-based performance?

For this reason, five different environments are used in the SRDR dataset. The datasets used are real-world backgrounds from indoor scenes from the Active-Vision dataset [4], real-world backgrounds of a table surface from the IRLab, a photorealistic indoor scenes dataset [244], a highly cluttered photorealistic indoor scenes dataset [74], and a non-photorealistic synthetic dataset

[132]. Samples from these datasets using the same scene configurations and real-world equivalent textures are in Figures 6.4 through 6.8.

Due to the inclusion of several different backgrounds, including real-world scenes, it is unsuitable for rendering the desired variations in-engine. The properties that are important such as real-world images, would be lost if we created and designed environments in the game engine. For this reason, the backgrounds are replaced by using the object masks to replace the background category with pixels from the selected background dataset. It is worth noting that this introduces some limitations, such as shadows not being cast on the backgrounds themselves. However, replacing image backgrounds does still achieve transfer from synthetic to real as demonstrated by existing literature [45, 192].

Figure 6.4: Sample training images using backgrounds from the Active-Vision dataset [4].

Figure 6.5: Sample training images using backgrounds from the Structured3D dataset [244].

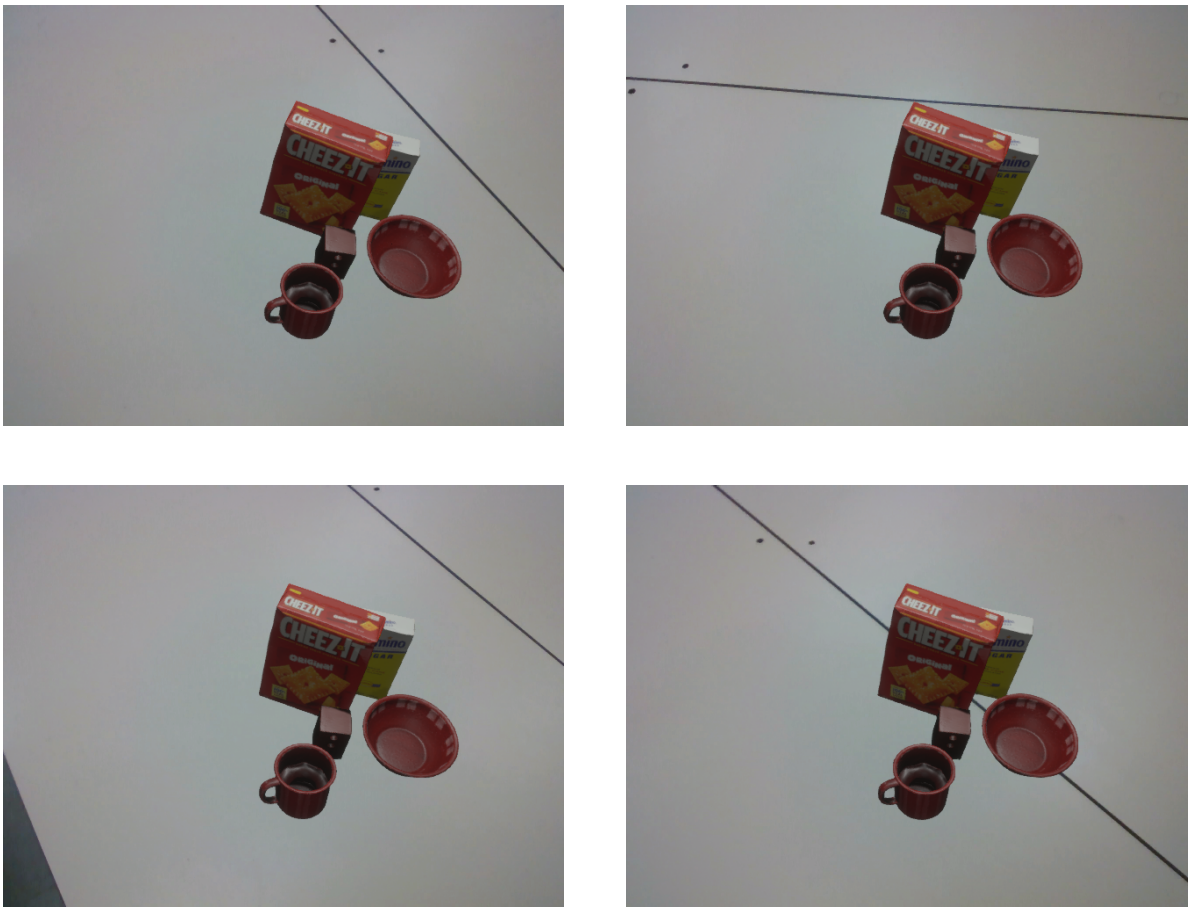


Figure 6.6: Sample training images using backgrounds from the IRLab.

Figure 6.7: Sample training images using backgrounds from the Photorealistic dataset [74].

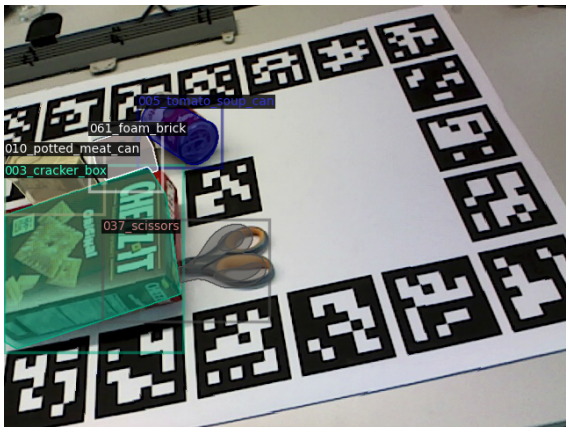
Figure 6.8: Sample training images using backgrounds from the Scenenet dataset [132].

Illumination

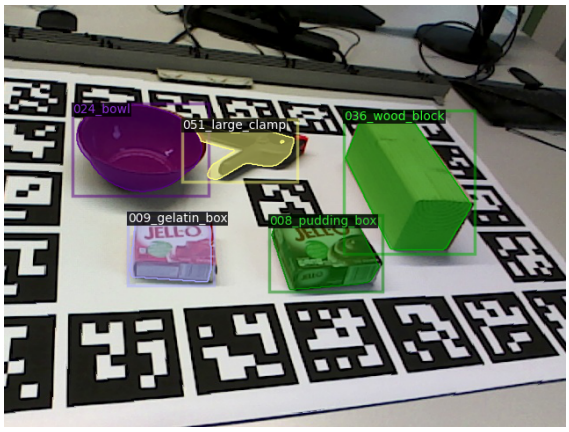
The illumination used in the SRDR dataset is manually placed by visually inspecting and tweaking one to five point light sources for each of the 31 scenes, as the YCB-M dataset does not provide illumination information. Intensity, light color, and radius of the point sources are manually adjusted for each scene configuration, attempting to match the real-world settings. Approximating the location of the light sources from real-world scenes is challenging. Alternatives can further improve this manual approach by trying to optimize the lighting positions for each scene using differentiable rendering. However, there would be trade-offs, particularly in creating more complex scenes with multiple objects, which would require additional computational power and introduces an auxiliary optimization problem when attempting to generate scenes.

Ground Truth Annotations

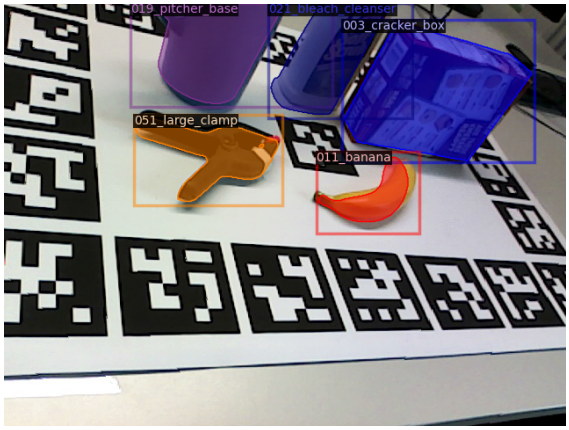
As the SRDR dataset replicates the YCB-M scenes, the original annotations from that dataset were used as the basis for placing the objects and cameras in the desired positions. Grenzdörffer, Günther, and Hertzberg [67] used a semi-automatic approach to gather the real-world image annotations, which involved labeling 6DoF poses of all objects in relation to a fixed reference frame defined by a border of fiducial markers, specifically ArUco markers. Grenzdörffer, Günther, and Hertzberg [67] generated initial guesses of object poses by using PoseCNN [228], then manually refining the guesses to remove false positives and missing objects. Due to this approach, there may still be some flaws in the original annotations produced by Grenzdörffer, Günther, and Hertzberg [67], which would subsequently be reproduced in the SRDR dataset. Some examples of imperfect annotations are in Figure 6.9.



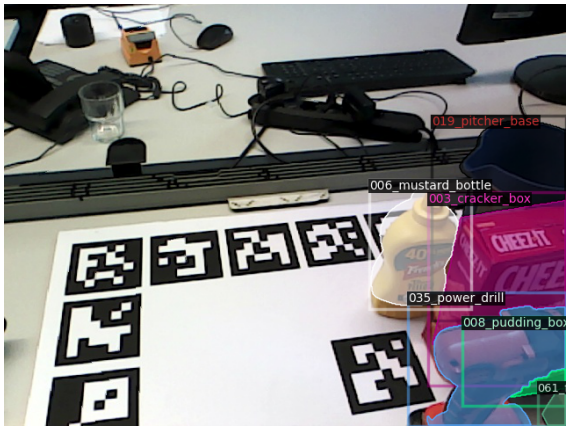
(a) Cracker Box and Scissors



(b) Large Clamp



(c) Banana



(d) Mustard Bottle

Figure 6.9: Sample annotations from the YCB-M dataset highlighting misaligned segmentation masks and bounding boxes. Grenzdörffer, Günther, and Hertzberg [67] use ArUco markers and generated initial guesses of object poses by using PoseCNN [228], then manually refining the guesses to remove false positives and missing objects. Despite the manual cleanup, there are still some imperfections such as Figure 6.9c, which has the segmentation mask slightly rotated relative to the original position of the object. Similarly, Figure 6.9b shows the large clamp bounding box and segmentation mask shifted to the left.

6.3.2 Testing

There are several ways researchers may use the SRDR dataset as part of their training regime to answer different research questions. Because of how the dataset is generated, one possible way of splitting the training and test set would be to hold out specific scenes from the available 31 as part of the test set, still allowing for each object instance to roughly be uniformly represented in the training and test sets. Another possible way would be to hold out environments from the training and test sets, for example, holding a single environment for testing and training on the remaining datasets. Due to the various DR techniques applied, researchers may choose to train on specific DR methods while testing real-textured equivalents. Finally, a combination of synthetic and real training images may comprise the training set while evaluating on a held-out real-world test set. The SRDR dataset is flexible in answering questions regarding domain adaptation. It is beneficial in assessing cross-domain investigations due to scene replication in a real-world object-centric setting.

6.3.3 Dataset Statistics

This section shows the statistics of the SRDR dataset, including distributions of object instances across all frames in the dataset, as well as samples from individual object statistics showing visibility and its centroid within the RGB images.

The number of visible objects for any given frame, where 0% is defined as fully occluded, and 100% as full visible, is shown in Figure 6.10. We generally see between four to six objects for any given frame in the SRDR dataset, with a few highly cluttered scenes containing seven or eight objects, and similarly with three objects. The dataset would mostly be useful in scenarios investigating low to medium cluttered scenes, as these are the most common number of objects in the frames in the dataset.

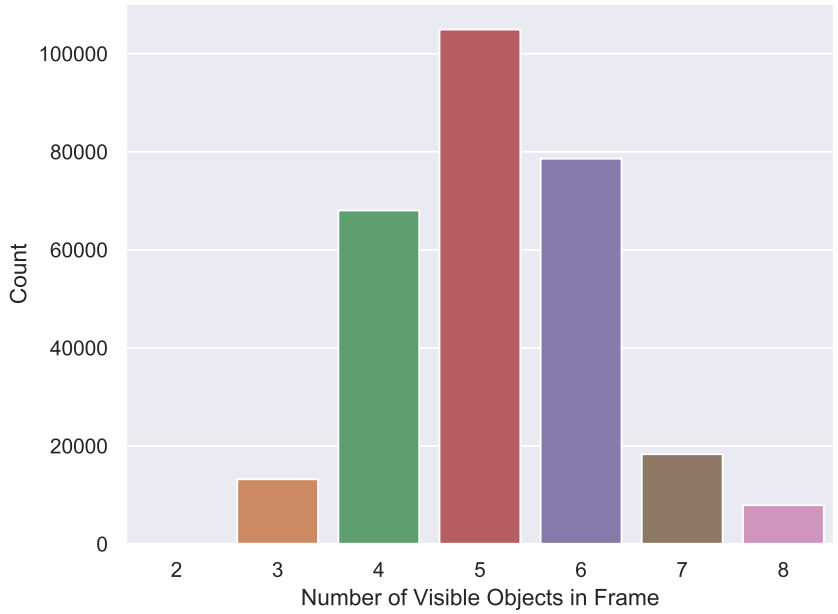


Figure 6.10: Figure showing the distribution of the number of objects per frame across the SRDR dataset. SRDR most commonly contains scenes with four to six objects.

In Figure 6.11, we see the total number of object instances across the entire SRDR dataset for all 20 classes. The most transparent bars highlights the number of objects that are greater than 25% visible across the entire dataset, while the second most transparent bar shows objects that are greater than 75% visible. The number of highly occluded instances are shown as solid bars, with less than 25% of those instances visible in a given frame. As shown, the objects are generally uniformly represented in the dataset, with a number of objects such as the power drill, mustard bottle, and foam brick occurring more frequently. Some smaller objects such as the scissors,

In Figure 6.12, we show the distributions for visibility for a subset of four objects of varying sizes, the smallest of the subset being the scissors, and increasing in size, to the banana, cracker box, and the pitcher. Here, we see that a majority of the frames for some of the smaller to medium sized objects such as the scissors, banana, and cracker box, have a majority of their frames being moderately visible (greater than 75% visibility). Although some of the smaller objects such as the scissors and banana, are more likely to be occluded due to their size. Similarly, the pitcher being

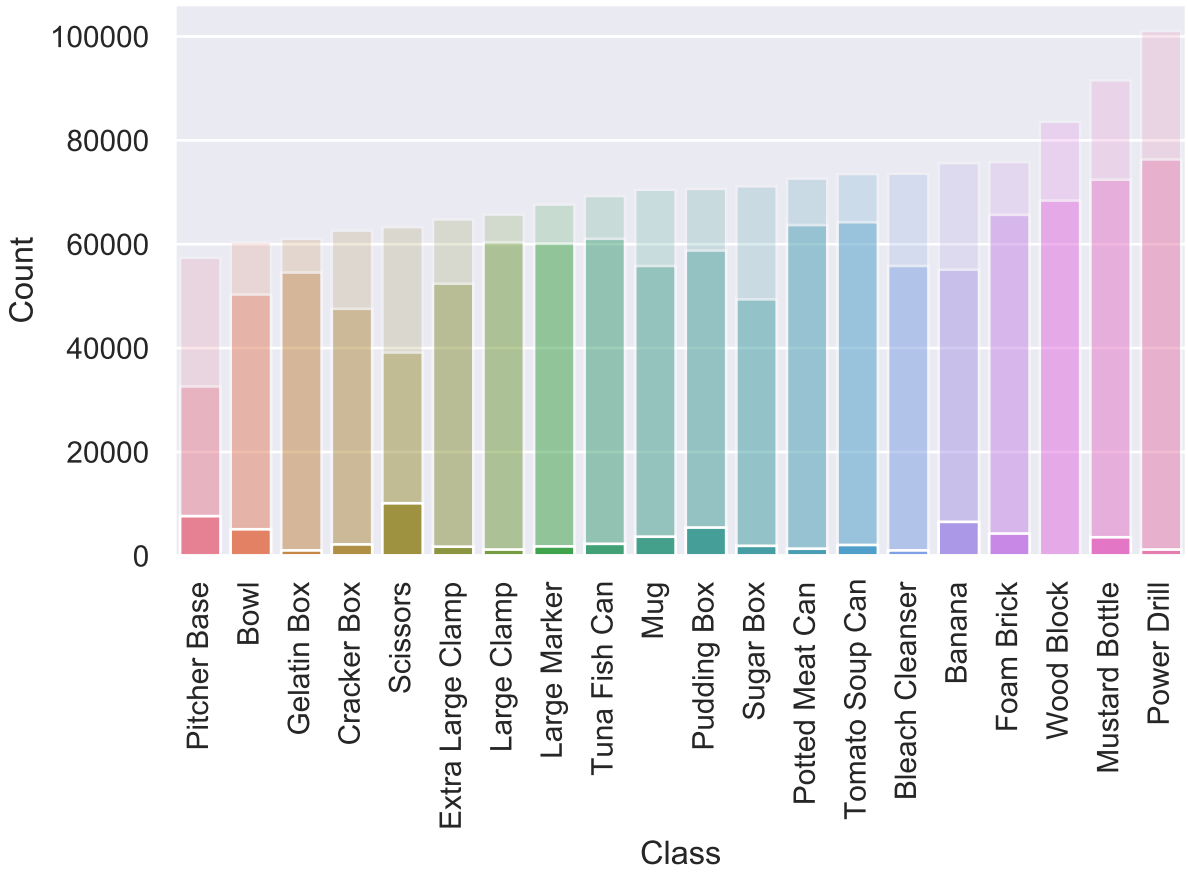


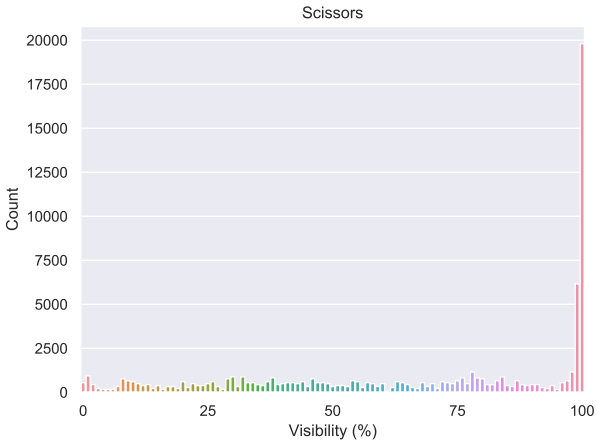
Figure 6.11: Figure showing the total number of object instances across the SRDR dataset. The most transparent bars highlight the number of objects greater than 25% visible, while the second most transparent bar shows objects that are more than 75% visible. Objects that are highly occluded (less than 25% visible) are solid bars.

the largest of the objects is more often to not be fully visible in the entire frame, also due to it's size.

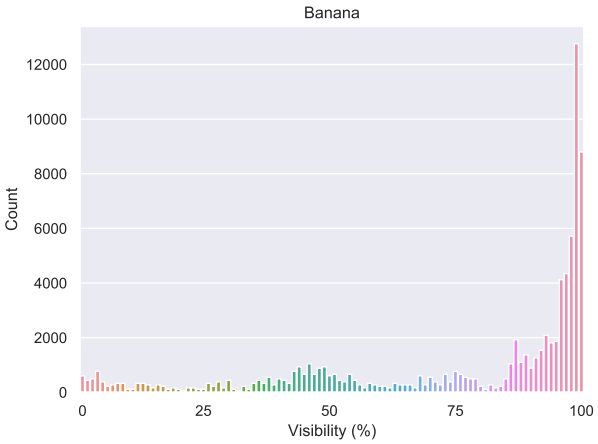
In Figure 6.13, the same subset as in Figure 6.12 is shown, with each object's position in the exported RGB frame is presented. Generally, the objects are placed around the centre of the frame, forming a rough Gaussian centred around the middle of the image. Although some objects such as the scissors or cracker box appear more often on either sides of the frame. This is similar to the other classes in the dataset.

6.4 Conclusion

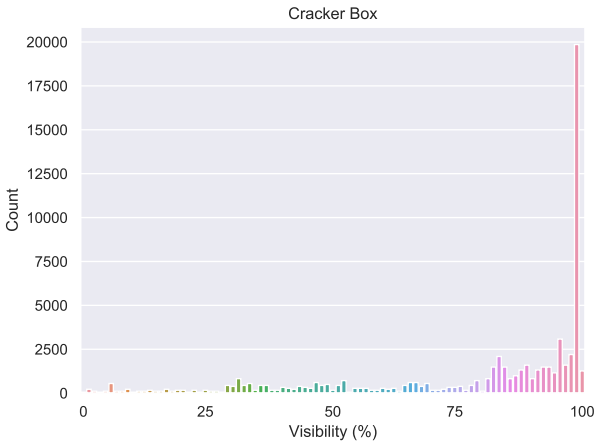
In this chapter, we introduced a new DR focused dataset called the SRDR dataset, one of the larger DR datasets currently available. The dataset is in a unique position to probe questions surrounding using DR in both robotics and computer vision, as it provides rich annotations for solving tasks in pose estimation depth estimation, and scene understanding using a wide variety of commonly applied DR techniques. The dataset replicates real-world scenarios from 31 scenes of various complexities, allowing researchers to incorporate synthetic, DR synthetic, and real-world images for cross-domain applications. The scenes range from various complexities including a range of visible objects, occlussion, and cluttered scenes. Additionally, the SRDR plugin can be used as an out-of-the-box software for replicating real-world scenes using scene description files, allowing greater flexibility outside of the YCB objects currently in use.



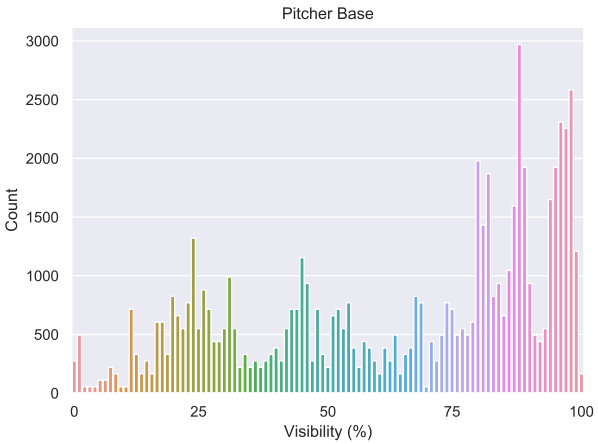
(a) Scissors



(b) Banana

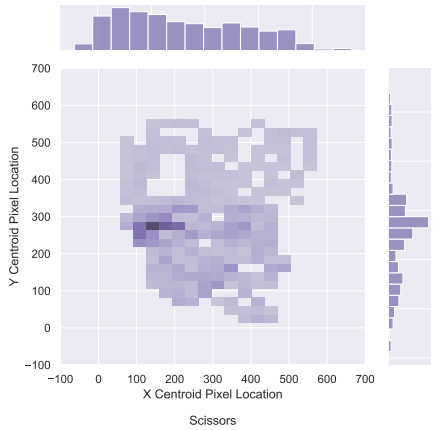


(c) Cracker Box

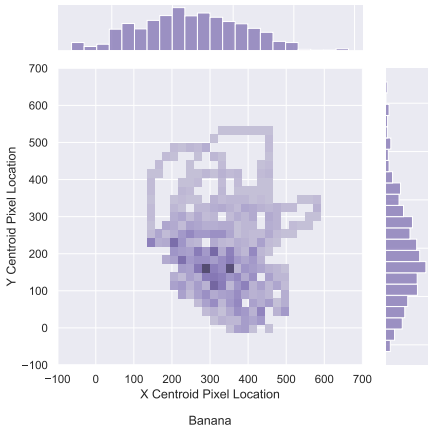


(d) Pitcher Base

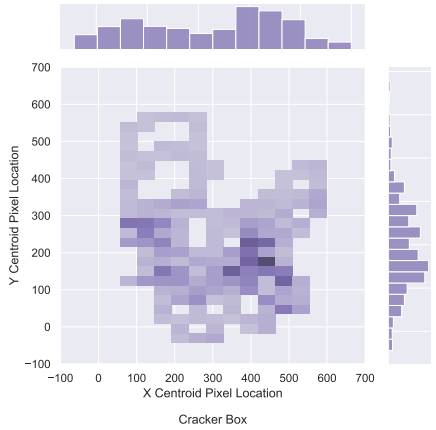
Figure 6.12: Visibility across all frames for a subset of four objects varying in shape and size.



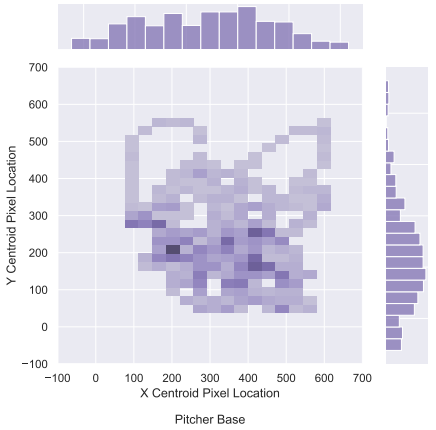
(a) Scissors



(b) Banana



(c) Cracker Box



(d) Pitcher Base

Figure 6.13: Position of object centroids across all frames for a subset of four objects varying in shape and size.

Chapter 7

Generalizability of DR for Multi-Tasks

This chapter explores DR's generalizability across multiple tasks for transfer from synthetic to complex real-world scenes. Currently, DR is used in several computer vision and robotics tasks [14, 52, 81, 147, 159, 180, 201, 202, 206, 207, 232]. However, current literature does not appear to agree on a preferred method for performing DR when solving a particular task. For example, would using a particular type of texture result in higher task-based accuracy when solving object localization than object detection? Additionally, a large portion of DR's use cases is with geometric primitive object shapes, which are simple geometric shapes such as a sphere, cylinder, cube, cone, or pyramid [14, 81, 147, 155, 201, 206, 232, 237]. The use of simple geometric shapes simplifies the problem. These shapes are typically rotationally invariant, meaning their visual appearance would not differ drastically from various viewpoints and may require fewer data to solve.

Furthermore, some researchers solve tasks involving a single object of interest [81, 147, 155, 201, 232], which removes another critical challenge in computer vision - addressing intra-class variability. For example, boxes can come in various shapes and sizes, such as a cereal box being larger than a pudding box. It is unclear how DR would perform in scenes of increased complexity arising from multiple objects of interest, increased object geometry complexity, and increased clutter. Recent work has attempted to benchmark DR for sim-to-real transfer for pose

estimation and found that increased photorealism in the synthetic training data, distractor objects, and randomization of textures played a crucial role in aiding transfer from sim-to-real [3]. It is important to note that the investigation used simple geometric shapes (cube, prism, hexagon, and triangular prism), a single object of interest, and no occlusion (object of interest is always visible).

As examined in Chapter 5, and echoed by the results by Alghonaim and Johns [3], the selection of texture randomization influences final task-based performance in an object localization task [5]. Here, we find that the use of more complex patterned textures generally increases task-based performance. For example, using a checkerboard pattern instead of the most widely applied texture randomization technique using flat shades of colors resulted in higher task-based performance.

Two open questions remain regarding the use of DR for transfer from sim-to-real in complex scenes involving multiple objects of interest, increased object geometric complexity, and occluded/ cluttered scenes. What are the design choices when attempting to solve some arbitrary task in complex scenes, and are the design choices influenced by the task at hand? For example, would the same DR approach for objection detection perform similarly to semantic segmentation? Would we have to alter the DR design choices when solving tasks involving multiple objects and increased occlusion/ clutter?

This work investigates the influence of commonly selected DR choices across multiple tasks. We study poses, backgrounds, and object textures in complex sim-to-real settings. The tasks investigated are object detection and semantic segmentation, as these remain complex tasks in real-world robotics and computer vision applications. The work involves analyzing the selection of pre-trained weights, the influence of object poses, the selection and type of backgrounds, and the texture randomization technique for objects of interest in complex scenes.

The chapter opens with the problem formulation and the proposed approach for investigating this using the appropriate networks. Next, the real-world dataset and synthetic dataset generation routines are defined, wherein a similar fashion to chapter 4, several synthetic datasets are

created to evaluate the impact of a current choice of a parameter such as the poses, backgrounds, or textures used. The experimental section details the setup for the choice of the parameter being investigated, outcomes for each. The chapter concludes with a discussion summarizing the key findings and suggestions for using DR for multiple tasks.

We outline our contributions below:

- In this work, we perform a comprehensive study to evaluate DR's generalizability and robustness in sim-to-real settings by randomizing poses, textures, and backgrounds in cluttered and occluded scenes using realistic household objects [23] for object detection and semantic segmentation.
- We find that the performance ranking is largely similar across the two tasks when evaluating models trained on DR synthetic data and evaluating on real-world data, indicating DR performs similarly across multiple tasks.
- Our findings indicate that training a single Mask-RCNN network to solve detection and segmentation tasks in complex scene configurations using DR data with 20 object classes does not work well when evaluating the model on real-world images.
- Based on our findings, we advise the following design choices when utilizing DR for complex scenes:
  - We propose researchers developing new DR data to focus on the diversity of poses for an object of interest that would more likely appear in the target setting (e.g., using a normal distribution centered around the middle of a table for countertop scenes).
  - Using textures containing complex patterns such as Checkerboard or Zig Zag.
  - Using a robust set of backgrounds from the real-world, or photorealistic synthetic backgrounds.

7.1 Method

This work is primarily concerned with investigating the generalizability of DR across multiple tasks. In this section, the problem definition that is shared across numerous experiments and evaluation metrics are defined.

7.1.1 Problem Definition

The goal of the experiments in this chapter is to learn how DR generalizes when we train a model for solving multiple tasks: object detection and semantic segmentation. This goal would guide researchers towards fundamental design choices for future DR work in complex scenes. Object detection aims to classify individual objects in an RGB image and localize them using tight 2D bounding boxes. With semantic segmentation, the task is to classify individual pixels of an RGB image to a fixed set of available classes without considering separate instances of a particular object.

Note in Table 4.1 that DR researchers use DR in a wide variety of tasks within computer vision and robotics. From this table, we select two challenging tasks to investigate the generalizability of DR across multiple tasks, specifically object detection and semantic segmentation. From Table 4.1 we see a large portion of the tasks solved involve grasping and object manipulation. However, detection and segmentation tasks typically serve as a backbone for solving a downstream task, such as in work by Sundermeyer [195], where they first detected an object with a 2D object detector, cropped the region of interest, then performed 6D pose estimation on the resulting image. Similarly, this occurs in 6D pose estimation tasks without using depth information, where we may want first to use detection or segmentation to locate an object of interest in 2D before solving the task at hand [94, 162, 198, 228]. The tasks are inherently challenging as they must solve several subtasks before reaching the goal. For example, we must first classify and localize an object of interest in an image with object detection.

7.1.2 Network Architecture

Several different networks and algorithms are commonly used for solving object detection [60, 72, 165, 168] and segmentation tasks [32, 71, 72, 137, 243]. For this Chapter, we use Mask-RCNN due to its robustness, high accuracy, and it has proven to be effective in the current literature [72]. Mask-RCNN combines losses for the predicted class, bounding box coordinates, and the segmentation mask, meaning a single network trains all of them jointly, reducing the possibility of performance discrepancies that may arise from hyperparameter selection or architecture when training across multiple networks. The model architecture used and training scripts were adapted from Wu et al. [222] using Mask-RCNN with ResNet-50-C4 backbone as the feature extractor, where the features are extracted at the conv4 block using a conv5 head.

To ensure a fair comparison, we fix the same hyperparameters across all experiments, using images of size 640x480, and a batch size of 4. The models were trained using Stochastic Gradient Descent (SGD), with a base learning rate of 0.00025, using a linear warmup factor of 0.001 for 10000 iterations, weight decay of 0.0001, and momentum 0.9. Unless stated otherwise, each of the models was initialized using pre-trained COCO weights [116, 222], and we trained the trained models until convergence.

7.1.3 Evaluation Metrics

Evaluating the performance of the tasks in object detection and segmentation is based on standard COCO metrics, reporting average precision (AP) over Intersection-over-Union. Specifically, using the primary challenge COCO metric where AP are averaged over 10 IoU thresholds between $[0.5 : 0.95]$ in increments of 0.05 [116]. We also report the conventional requirement for AP at IoU 0.5 (AP_{50}), based on the PASCAL Visual Object Classes (VOC) challenge [51], which defines a correct prediction when the IoU between a prediction and ground truth exceeds 0.5 (50%). We also report IoU at 0.75 (AP_{75}), which is considered a strict threshold [116].

Using equation 7.1, for a given predicted bounding box or segmentation detection D_{pred} and ground truth D_{gt}, the overlap between the two must meet the above thresholds. $D_{pred} \cap D_{gt}$ is the intersection of the predicted and ground truth detection or segmentation, and $D_{pred} \cup D_{gt}$ is their union. Figure 7.1 shows an illustration to compute IoU for bounding box detection for the sugar box.

$$IoU = \frac{|D_{pred} \cap D_{gt}|}{|D_{pred} \cup D_{gt}|} \tag{7.1}$$

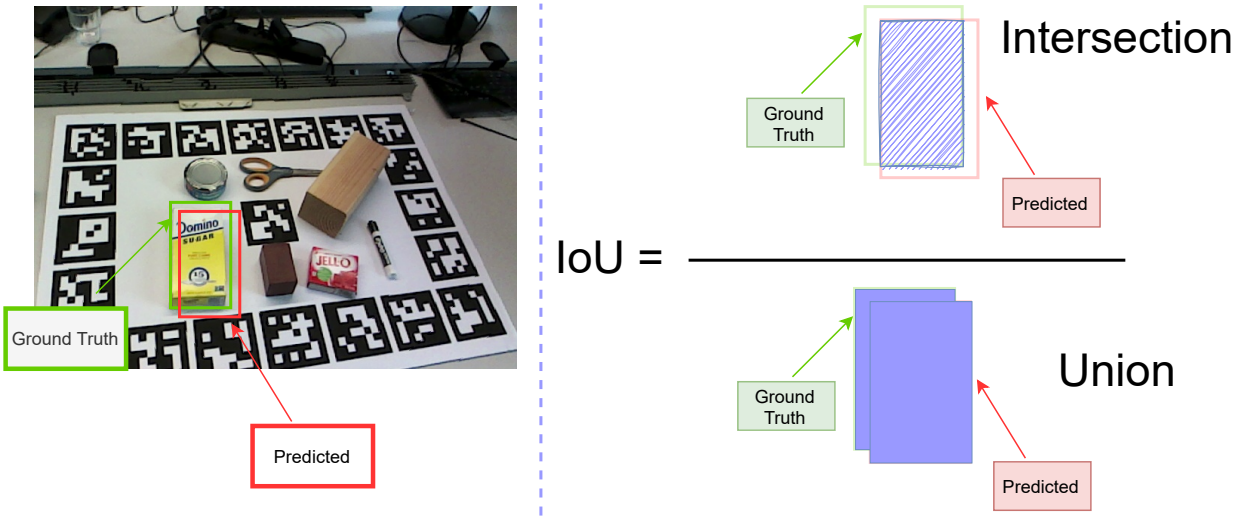


Figure 7.1: Figure showing how IoU is computed using bounding boxes. The RGB image [67] on the left with a green bounding box represents the ground truth, and the red boxes represent a model's prediction. To compute the model's accuracy, IoU is used by using the ratio between the overlapping area of the ground truth bounding box and the predicted bounding box and the total area from the ground truth and model predictions as shown on the right.

In addition to the above metrics, we report AP at different scales AP_S, AP_M, AP_L. These scales are for small objects (area $< 32^2$), medium objects ($32^2 <$ area $< 96^2$), and large objects (area $> 96^2$). We compute the area as the number of pixels in the segmentation mask to define the size of each object.

Similar to the COCO metrics, the evaluation for detection using bounding boxes and seg-

mentation using masks are the same, with the distinction being the use of boxes or masks for the IoU computation in Equation 7.1 [116]. In the case of reporting AP at different scales, AP_S, AP_M, AP_L, the areas are computed using the bounding box area for the detection task and the mask area for the segmentation task.

7.1.4 Dataset Generation

As we are interested in testing several factors that may influence using DR across multiple tasks, several different synthetic datasets were generated using NDDS [200] and the SRDR framework presented in Chapter 6, where further details surrounding the generation routine is presented. To address the research questions regarding the transfer from synthetic to real across multiple tasks, we used a complex real-world dataset that uses the realistic household objects from a commonly used computer vision and robotics dataset [23, 67]. The real-world scenes are from the YCB-M dataset, which features complex scenes involving varying degrees of clutter, occlusion, and non-primitive shapes that would more accurately represent functionality in real-world settings. Further details for each of the datasets used in the following experiments are presented in the proceeding subsections.

Synthetic Dataset

Two primary datasets are used in the subsequent experiments: the first being non-replicated scenes, wherein the poses of the objects used to generate the data do not match a real-world target dataset. The second is synthetic data of replicated scenes, where the objects and associated poses replicate the real-world target dataset.

Non-Replicated Scenes

The non-replicated synthetic dataset comprises four non-primitive objects (simple geometric shapes such as a cube, cylinder, sphere, pyramid, or cone) from the YCB dataset [23]. The four objects used are a power drill, bleach cleaner bottle, banana, and mustard bottle. To generate the scenes, a set of parameters, including the camera information, poses of objects, light source, textures, and choice of backgrounds, are provided to the data generation routine [200]. Camera parameters remain fixed, as is the illumination, which is a static single point-source centered roughly above the camera and positioned towards the objects of interest. The choice of the point source and camera position is such that the scene is sufficiently lit and all objects are visible in the frame. The poses of the objects are randomized for each frame, for the position and orientation. This dataset includes real-world equivalent textures applied to the object meshes and ten different texture DR techniques that are commonly used in the existing literature [5]. Depending on the experiment conducted, the background used is one of the following: IRLab, Photorealistic, SceneNet, Active Vision [4, 74, 132] as shown in samples from the non-replicated dataset using the real-textured equivalents are in Figure 7.2.

Replicated Scenes

The replicated synthetic dataset matches the 31 scenes containing a total of 20 objects from the YCB-M dataset [67]. The 20 objects used in the replicated scenes datasets are in Figure 7.3. The YCB-M dataset comprises scenes from multiple cameras. For all replicated scenes, we chose to replicate the images from the viewpoint of the Intel RealSense R-200 camera. We match object poses, camera poses, camera intrinsics, and manually match illumination from the YCB-M dataset [67]. Further information regarding the generation of this dataset is detailed in Chapter 6.

Figure 7.2: Sample images from each of the non-replicated scene datasets using realistic household objects from the YCB dataset [23]. The positions and orientation for each of the object is sampled from a uniform distribution for each frame in each dataset. Illumination and camera position remain fixed, and a different background is applied to each image in the datasets. The backgrounds show varying degrees of realism and background clutter, which acts as distractor objects.

Figure 7.3: Figure showing object models and real-equivalent textures from the YCB dataset [23]. The 20 objects shown are used in all experiments involving replicated scenes from the YCB-M dataset [67].

Real-world Dataset

The real-world dataset used to answer the research questions is from the YCB-M dataset [67], in particular the RGB images from the Intel RealSense R-200. The dataset consists of 20 YCB objects [23] across 31 unique scenarios, with varying degrees of scene complexity. The challenging dataset includes varying degrees of occlusion, object complexity, and clutter, which is more indicative of complex environments that may be challenging when using DR. We investigate poses, textures, and types of backgrounds used. Samples of the real-world dataset and the photorealistic synthetic replicated scenes are shown in Chapter 6.3.1.

7.2 Sensitivity to Weight Initialization

This experiment serves as the basis for all proceeding experiments to use the pre-trained weights, as performance can significantly vary depending on the pre-trained weights used before fine-tuning [185]. Two commonly used pre-trained weights are ImageNet and COCO weights for solving image-based tasks. This experiment investigates the usability of pre-trained ImageNet and COCO weights in Mask-RCNN for solving object detection and semantic segmentation. Here, the experiment investigates the sensitivity for weight initialization for solving complex object-centered

tasks, addressing the effects of pre-training with ImageNet over COCO, and ensuring that proceeding investigations are not hindered by poor initialization.

7.2.1 Experimental Setup

Mask-RCNN with ResNet-50 backbone networks are trained with two initialization settings, the first using the original ImageNet weights provided by MSRA in the deep residual networks paper by He et al. [71], the other is using MSCOCO weights provided by Wu et al. [222], which is trained on all images in the "train2017" dataset from COCO [116].

The dataset used for this investigation is from a subset of the real-world YCB-M dataset described in Chapter 6. The subset is a selection of four objects of varying complexity: a power drill, bleach cleanser bottle, banana, and mustard bottle. The four objects selected have unique shapes and sizes and are more visible in the available frames from the YCB-M dataset, as shown in Table 6.11, from Chapter 6, which uses the poses and visibility annotations from the YCB-M dataset. The breakdown of instances for the training and test set is in Table 7.1, with samples of images from the dataset in Figure 7.4, which shows RGB images overlaid with the detection and segmentation masks. In the case of the test set, entire scenes are held out from the training set, such that no frames in the test set are visible in the training set.

Class	Train	Test
Power Drill	1456	647
Bleach Cleanser Bottle	1180	275
Banana	1429	587
Mustard Bottle	1664	647

Table 7.1: Instances per class for a training set of size 2320 and a test set of 647 real-world images at a resolution of 640x480 from the YCB-M dataset [67]. This dataset is used for determining a set of pre-trained weights for Mask-RCNN for object detection and segmentation tasks.

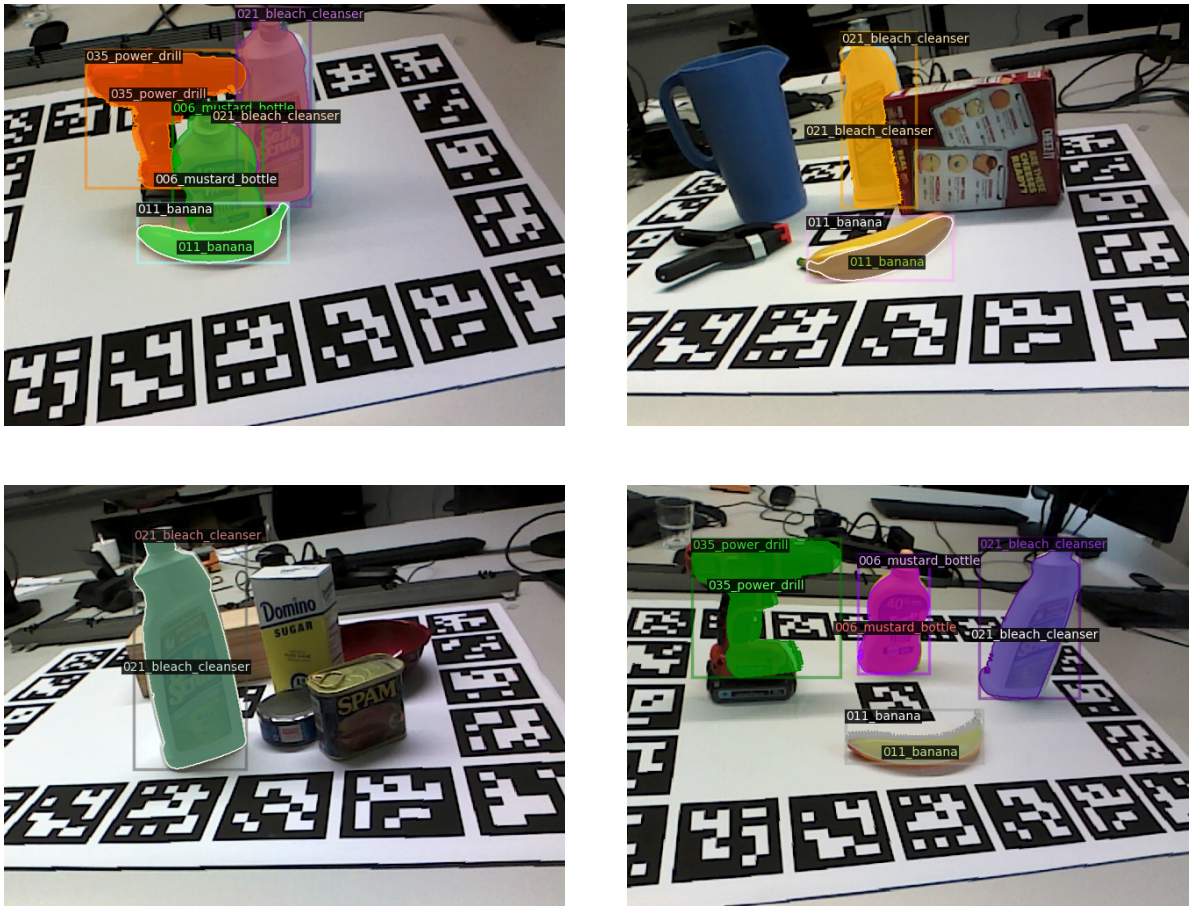


Figure 7.4: Sample images from several object-centric real-world scenes from the YCB-M dataset [67] containing the power drill, banana, mustard bottle, bleach cleanser. Ground truth annotations are overlaid with the RGB images.

7.2.2 Results

The results for the object detection tasks are in Table 7.2 for all evaluation metrics, and Table 7.3 for the per-category results for the power drill, bleach cleaner bottle, banana, and mustard bottle. The segmentation task results are presented in table 7.4 and 7.5 for the per-category results.

Object Detection

From Table 7.2 and 7.3, we see the selection of COCO weights greatly improves the performance in all cases, where the AP score using ImageNet weights is 29.572 AP and COCO weights is 47.191 AP. This conclusion is also reflected in the per-category results, where relative performance between each category is consistently higher when using COCO weights. This result may stem from the data available in ImageNet compared to the COCO datasets. ImageNet contains single objects with tight crops around the object of interest for classification tasks. In contrast, images in the COCO dataset contain complex scenes with multiple objects for detection and segmentation tasks.

In some cases, initializing model backbones for detection and segmentation tasks using ImageNet classification task are practiced [32, 43, 61]. However, studies have shown that pre-training on ImageNet does not improve accuracy in solving a detection task on the COCO dataset [59, 70, 184]. For our experiments, we select the COCO weights due to the higher performance when solving our task.

Semantic Segmentation

Tables 7.4 and 7.5 show a similar result, where the AP score for using COCO weights improves from 28.446 AP when using ImageNet weights to 38.148 AP when using COCO weights. Generally, this trend follows in the per-category results, where performance is higher when using COCO

Weights	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
COCO	**47.191**	**81.733**	**42.184**	**10.674**	**45.995**	**66.648**
ImageNet (MSRA)	29.572	74.447	11.538	0.000	32.303	44.460

Table 7.2: Bounding box AP using both pre-trained COCO [116] and pre-trained ImageNet (MSRA) weights [71] with a Mask-RCNN network [72] and a ResNet-50 backbone. Pre-trained COCO outperforms pre-trained ImageNet across the board.

Weights	AP	Power Drill	Bleach Cleanser	Banana	Mustard Bottle
COCO	**47.191**	**38.376**	**84.438**	**14.973**	**50.978**
ImageNet (MSRA)	29.572	28.998	41.027	4.266	43.999

Table 7.3: Per-Category bounding box AP using both pre-trained COCO [116] and pre-trained ImageNet (MSRA) weights [71] with a Mask-RCNN network [72] and a ResNet-50 backbone. Pre-trained COCO outperforms pre-trained ImageNet for all classes.

weights over ImageNet, apart from the power drill, where performance is 15.218 AP when using COCO weights and 16.812 AP using ImageNet weights.

Weights	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
COCO	**38.148**	**61.737**	**35.673**	**1.089**	**36.064**	**62.131**
ImageNet (MSRA)	28.446	61.049	17.966	0.000	26.963	51.542

Table 7.4: Semantic segmentation mask AP using both pre-trained COCO [116] and pre-trained ImageNet (MSRA) weights [71] with a Mask-RCNN network [72] and a ResNet-50 backbone. Pre-trained COCO outperforms pre-trained ImageNet across the board.

Weights	AP	Power Drill	Bleach Cleanser	Banana	Mustard Bottle
COCO	**38.148**	15.218	**78.871**	**9.209**	**49.291**
ImageNet (MSRA)	28.446	**16.812**	48.474	3.191	45.306

Table 7.5: Per-Category semantic segmentation mask AP using both pre-trained COCO [116] and pre-trained ImageNet (MSRA) weights [71] with a Mask-RCNN network [72] and a ResNet-50 backbone. Pre-trained COCO outperforms pre-trained ImageNet for all classes apart from Power Drill.

Based on the results above, it seems evident that using COCO weights would generally yield higher performance for this particular object-centric dataset to solve for the following experiments. The selection of the weights is susceptible to the specific tasks and dataset at hand, highlighting the importance of pre-trained weights to use when attempting to solve detection and segmentation. Mask-RCNN with ResNet-50 backbone pre-trained on COCO weights is used for all subsequent experiments, reducing the possible performance degradation with poor weight initialization. While it is possible to achieve comparable performance to pre-training on ImageNet weights when using random weight initialization [70], finding an optimal set of hyperparameters is required, and the use of pre-trained weights yields adequate performance.

Furthermore, in our experiments, we investigated weight initialization using one architec-

ture utilizing Mask R-CNN. While Mask R-CNN is commonly used for solving detection and segmentation tasks yielding state-of-the-art results, other architectures such as AlexNet [103], VGG [187], or GoogLeNet [196] have varying capacities. It would be interesting to compare the previous findings to alternative architectures.

7.3 Object Poses

Attempting to bridge the gap from synthetic to real using domain randomization also involves some form of randomization of poses as seen in Table 4.2. Pose randomization introduces a greater variety of orientations and positions for objects of interest, which increases the number of viewpoints for a particular object and aid in generalizing to unseen object poses. This variation in poses can be in the form of modifying the camera positions and angles to get different viewpoints of objects of interest, modifying object locations and orientations, or a combination of both. In the following set of experiments, the positions and rotations of objects of interest are randomized. The experiment aims to analyze how poses can influence the accuracy of object detection and segmentation tasks. In cases where the training and test sets sample poses from a similar distribution, for example, sampling positions and orientations from a uniform distribution, we expect high performance due to similar sampling strategies and visual appearance.

7.3.1 Experimental Setup

Synthetic datasets are generated to answer this question containing four objects of interest: power drill, bleach cleanser bottle, banana, and mustard bottle with 5400 images for each synthetic training set. The positions and orientations of each of the objects are sampled from a uniform distribution. The initial starting locations for each of the four objects are centered approximately around positions of the four objects in the real-world test set shown in Figure 7.5 from a single scene in the

YCB-M dataset [67]. The initial starting positions for the four objects are to ensure the synthetic dataset does not contain the four objects in positions that are drastically dissimilar from the real-world test set, such as extreme corners of the image frame. While it is possible to freely sample positions from a uniform distribution across the entire frame, the size of the training set will also be significantly larger to ensure similar poses in the training set would appear in the test set.

The objects can be generated ± 5 cm from the starting location and can freely rotate. A total of 11 training sets are evaluated, one for each of the texture randomization techniques commonly applied as previously shown in Figure 5.1, and one using the real-world equivalent textures. The real-world equivalent textures are the original textures for each of the objects of interest, as shown in Figure 7.2.

Two test sets are used to evaluate the performance of the detection and segmentation tasks using the above training sets. The first test dataset is a synthetic dataset containing 2700 images, which uses the equivalent real-world textures and the same strategy to generate random poses as the synthetic training set. Evaluation of performance on the tasks on the synthetic test set gives us an understanding of performance in a similar domain.

The real-world test set is a real-world scene from the YCB-M dataset containing the same four objects with no occlusion or clutter. The removal of occlusion and clutter in the test set allows us to focus on the effects of randomizing the pose and on the textures used in the training sets. Samples from the real-world test set are in Figure 7.5, with the visible instances in each frame of the dataset shown in Table 7.6.

7.3.2 Evaluation on Synthetic Images

The results for the object detection and segmentation tasks are in Tables 7.7 and 7.8, showing the AP scores and the per-category scores for each of the objects of interest. This set of results are evaluating trained synthetic models on a synthetic test set.

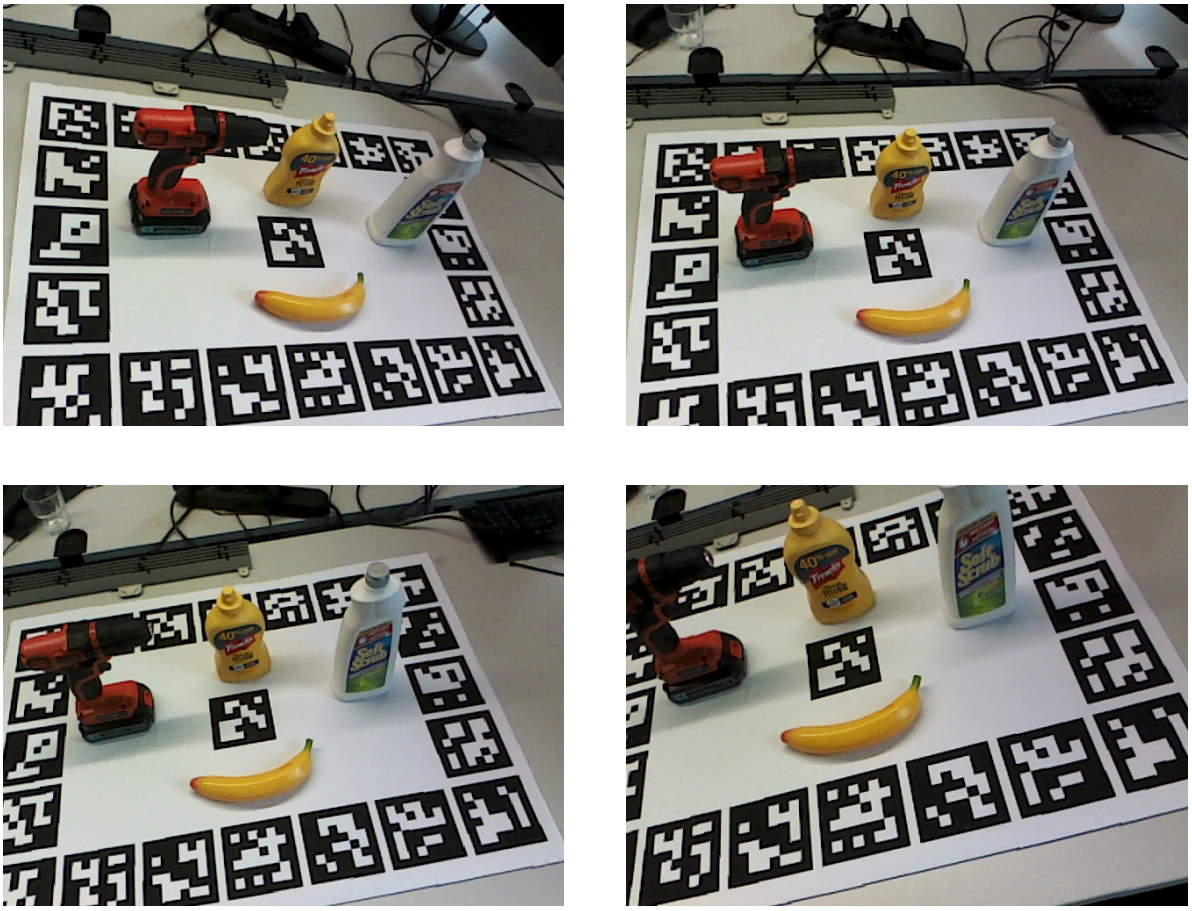


Figure 7.5: Sample images from the real-world test dataset consisting of 169 images from the YCB-M dataset [67]. The scene contains four objects found in the synthetic training set, with no clutter or occlusion.

Class	Train (real-world)	Test (real-world)
Power Drill	1456	169
Bleach Cleanser Bottle	1180	169
Banana	1429	154
Mustard Bottle	1664	169

Table 7.6: Instances per class for the real-world training set of size 2320 and an unoccluded test set of size 169 real-world images at a resolution of 640x480 from the YCB-M dataset [67]. The test set is a single scene shown in Figure 7.5 containing the four objects and is not visible in the real-world training set.

Object Detection

From Table 7.7, when we evaluate the model using images from the synthetic test set, we find that texture type strongly affected task performance. Similar to the findings in Chapter 5, more complex patterned textures appear to perform better than non-patterned textures. For example, in the object detection task, the highest performing DR model used striped textures, achieving an AP of 67.483, compared to the lowest-performing DR model using Gradient RGB, achieving an AP score of 63.483. A possible explanation for better performance when using complex patterned textures is that the majority of the objects in the test set do not contain a flat shade of color. The banana is the only object that visually appears to be a single color, as shown in previous samples from Figure 7.2. In the detection task in Table 7.7 we see that selecting Flat RGB as the texture type for the randomizations results in the highest performance when detecting the banana with an AP score of 84.416 compared to the overall highest performing texture type Striped, which resulted in an AP score of 82.642. One possible cause for the banana achieving the highest performance with Flat RGB may be the object's visual appearance, which contains fewer patterns than the other objects such as the power drill, bleach cleanser, or mustard bottle. The banana's closer smooth texture resemblance could be more beneficial when solving the detection task for this particular

object.

Texture	AP	Power Drill	Bleach Cleanser	Banana	Mustard Bottle
Real-Texture	62.709	78.263	59.146	83.293	30.132
Gradient RGB	63.483	78.069	58.705	83.578	33.581
Flat RGB	64.487	76.912	60.451	**84.416**	36.168
Gradient RGB Perlin	64.739	79.216	61.351	82.900	35.489
Striped Perlin	65.363	77.865	58.626	80.860	44.103
Flat RGB Perlin	65.653	79.355	59.983	82.426	40.849
Checkerboard	66.232	77.825	59.398	83.923	43.784
Zig Zag Perlin	66.624	78.944	61.122	82.707	43.725
Zig Zag	66.737	77.599	61.657	81.955	**45.735**
Checkerboard Perlin	67.028	80.613	61.886	83.807	41.807
Striped	**67.483**	**80.654**	**62.227**	82.642	44.408

Table 7.7: Per-Category object detection (bounding box) AP using COCO weights [116]. The network was fine-tuned using synthetic images with the original object textures defined as Real-Texture. The remaining datasets are the ten texture DR techniques used in the current literature. Each model was evaluated on a synthetic Real-Texture test set of size 2700 for the four objects of interest.

Semantic Segmentation

The same follows for the segmentation task in Table 7.8, where Striped patterns are also the highest performing texture, while Gradient RGB Perlin being the lowest with AP scores of 87.970 AP and 83.197 AP, respectively. The selection of texture types affects task performance when operating in the same domain, in this case, using synthetic data as the source domain and synthetic data as our target domain. Performance generally increases when selecting more complex patterned textures compared to non-patterned ones. Sample detections and segmentations from this set of

experiments are shown in Figure 7.6 using Striped patterned textures.

An interesting observation in both the detection and segmentation tasks is the models trained on the real-textured weights resulting in the lowest AP scores. The model may be overfitting on the poses in the training set when detailed real-textured versions are used. An increase in dataset size when using the real-textures may reduce the likelihood of this event from occurring.

Texture	AP	Power Drill	Bleach Cleanser	Banana	Mustard Bottle
Real-Texture	80.190	83.134	90.064	83.312	64.249
Gradient RGB Perlin	83.197	82.761	88.426	85.002	76.598
Gradient RGB	83.292	83.429	87.350	84.772	77.615
Flat RGB	83.639	83.078	88.034	84.831	78.614
Checkerboard	85.180	83.735	88.012	85.050	83.925
Flat RGB Perlin	85.427	83.224	**89.020**	84.889	84.575
Checkerboard Perlin	85.603	83.511	88.331	85.139	85.431
Zig Zag Perlin	86.347	84.616	88.757	86.687	85.326
Striped Perlin	86.575	84.706	87.370	87.016	87.210
Zig Zag	86.775	84.952	88.155	86.895	87.097
Striped	**87.970**	**84.992**	88.485	**87.076**	**91.328**

Table 7.8: Per-Category object segmentation AP using COCO weights [116]. The network was fine-tuned using synthetic images with the original object textures defined as Real-Texture. The remaining datasets are the ten texture DR techniques used in the current literature. Each model was evaluated on a synthetic Real-Texture test set of size 2700 for the four objects of interest.

7.3.3 Evaluation on Real Images

So far, we have established baseline performance when testing in similar domains, where our source domain contains synthetic images, and our test domain is also synthetic images. We have

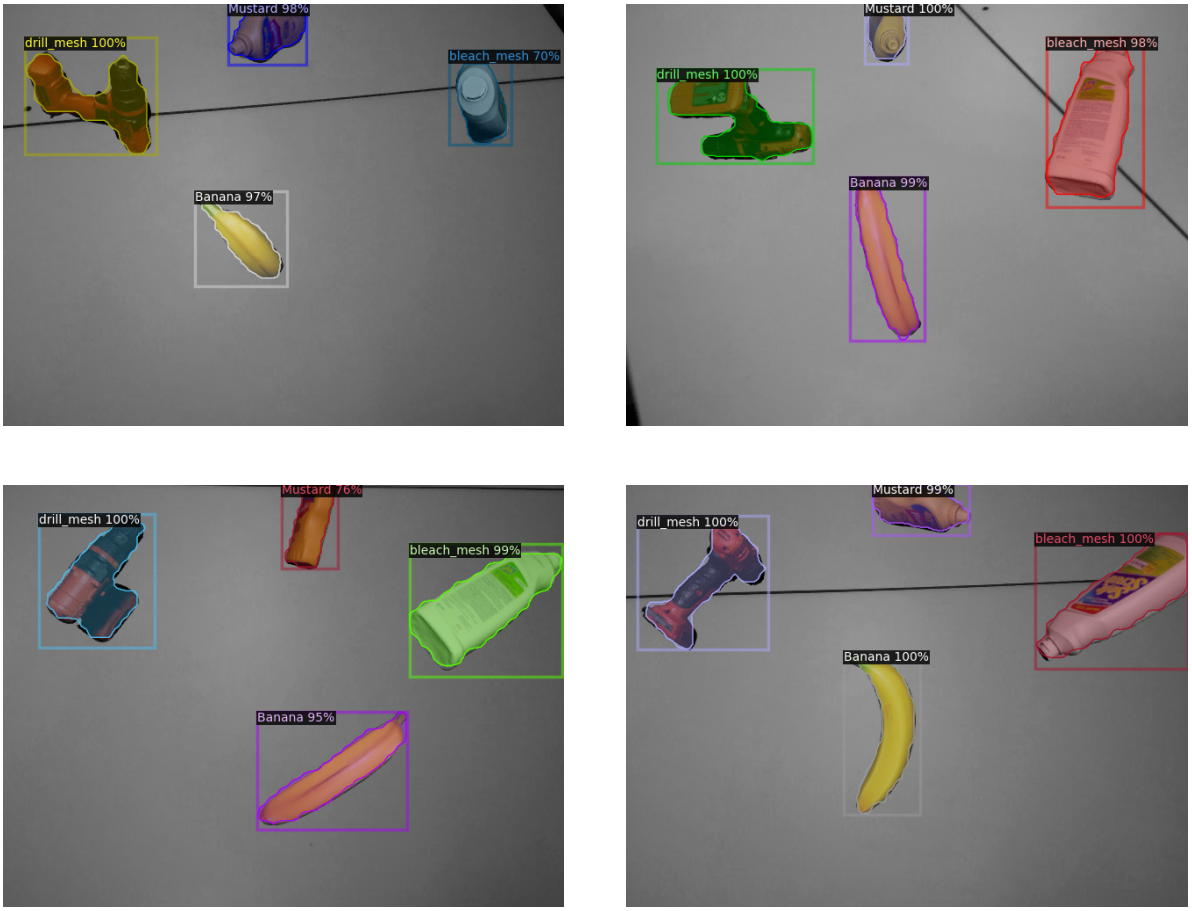


Figure 7.6: Using weights from the highest performing network (striped), we visualize some of the predictions from the network. Color has been removed from the images, apart from where an object has been detected and segmented. These are some examples where the network trained on Striped synthetic images, is able to do well.

found that in both detection and segmentation tasks, using more complex patterned textures outperforms non-complex patterned textures and resembles our findings from Chapter 5, where we find a similar ranking of performance for an object localization task.

We are interested in how this translates to performance in the real world. In this set of experiments, the previous models trained using synthetic images are evaluated on a test set consisting of the same four objects in the real world. The test set from the real world does not contain any occlusion or clutter, as shown by the samples in Figure 7.5. In tables 7.9 and 7.10, we see the performance of the highest performing texture, Striped, the real-world equivalent textures, and when trained on real images using the four objects from the YCB-M dataset [67]. The real-world training set uses 2320 images containing the four objects, and none of the images from the real-world test set appear in the training set. The number of instances per class for the training and test set for the real-world datasets are in Table 7.6.

Despite using a smaller training set for the real images, performance is significantly higher than synthetic images, which is expected due to the domain shift. Since our real-world training images are from the same real-world domain as our test set, the two datasets share very similar features such as the visual appearance of the objects of interest, similar poses, and illumination. Despite being one of the worst performers in the last set of experiments when evaluating synthetic test images, using the real-texture synthetic data when evaluating real-world images results in 23.320 AP compared to the score of the DR synthetic data of 15.971 AP for the detection task. In the segmentation task, the real-texture synthetic data scored 22.243 AP, and the DR synthetic data scored 14.272 AP. Since the target domain is quite different from the previous synthetic test domain as shown in Figures 7.2 and 7.5, the real-world equivalent textures may contain more important texture features for transfer from synthetic to real, particularly visual appearance. Such features could be the labels on the mustard and bleach bottles, the distinct colors on the tip and bottom of the banana, or the lettering on the power drill. However, with a larger, more diverse synthetic dataset using various textures, it is possible performance would be higher, as the increased dataset variety may encompass the real-world target data distribution.

Weights	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Real Images	**69.479**	**93.148**	**84.442**	**10.733**	**65.112**	**72.702**
Real-texture Synthetic Images	23.320	40.958	22.420	0.000	20.794	24.543
Striped Synthetic Images	15.971	34.889	5.546	2.376	12.652	9.318

Table 7.9: Object detection AP scores evaluating several models on a real-world test set from the YCB-M dataset [67] shown in Figure 7.5. The weights used are the highest performing texture DR method (striped images), the real-texture synthetic images, and real-world images.

Weights	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Real Images	**61.619**	**92.393**	**73.441**	**1.815**	**54.971**	**66.154**
Real-texture Synthetic Images	22.243	40.914	19.660	0.000	20.502	26.488
Striped Synthetic Images	14.272	33.531	5.382	0.562	11.533	10.619

Table 7.10: Object semantic segmentation AP scores evaluating several models on a real-world test set from the YCB-M dataset [67] shown in Figure 7.5. The weights used are the highest performing texture DR method (striped images), the real-texture synthetic images, and real-world images.

To better analyze the predictions when using the highest performing DR texture, using the synthetic dataset with striped textures, we overlaid the detection bounding boxes, classifications, and segmentation masks on the real-world RGB images in Figure 7.7. We see significant false positives in the ArUco markers and misclassifications of several objects, both in the middle of the image and background objects on the table beyond the markers. One possible consideration for such poor performance is the lack of background diversity compared to the textures and pose randomizations. Currently, the same 100 background scenes are repeated of a table from the IRLab, making it easier to distinguish the foreground from the background rather than the objects of interests themselves. The next set of experiments will investigate how influential a diverse background is to aid transfer from synthetic to real.

7.4 Image Backgrounds

While the previous models could transfer from synthetic to real using random poses and a fixed set of image backgrounds from the IRLab, using a training set of real-world images nearly doubles performance over the equivalent synthetic real-textures or highest performing Striped pattern. Variation in the image backgrounds and poses may play a role in more affecting performance when it comes to transferring to the real world. As Table 4.2 highlights several methods incorporating background randomization, this set of experiments investigates the use of unique backgrounds per frame in the training set. This set of unique backgrounds means that no two images in a training set would share the same background. These experiments would further develop an understanding of the selection of backgrounds for aiding transfer from synthetic to real.

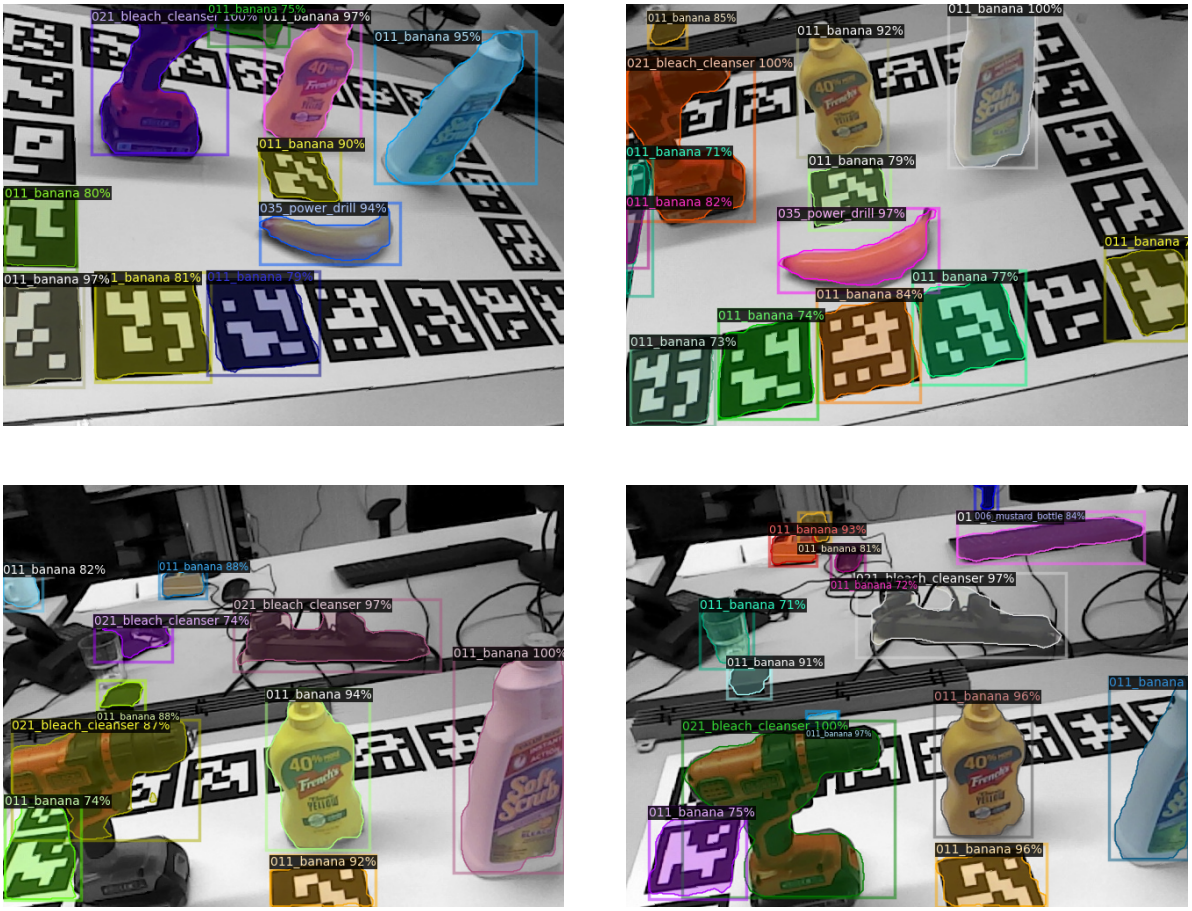


Figure 7.7: Visualization of network predictions on a real-world dataset when using Striped weights and the IRLab table background dataset from the SRDR dataset described in Chapter 6. The model has many false positives and commonly mistakes the ArUco markers as an object of interest.

7.4.1 Experimental Setup

The same synthetic dataset of random poses and four objects from Section 7.3.1 is used as the training sets, while the test set remains the 169 image real-world images with no clutter or occlusion. The differentiating factor between the training sets from the previous experiments is every sample from the training set uses a unique background from two distinct datasets. The first dataset of image backgrounds used is from the Active-Vision dataset [4], which is a set of real-world RGB images of indoor household scenes. The second dataset is a photorealistic synthetic dataset containing indoor household scenes from Hodaň et al. [74]. Figures 7.8 and 7.9 shows the difference between backgrounds used in the following experiment. The reasoning for selecting the two datasets for the backgrounds is due to having a range of realism (real-world and photorealistic synthetic images) and using indoor scenes where background clutter may be visible.

In these experiments, the object masks are used to segment the background from the foreground, and pixels from the synthetic training set are replaced with pixels from the background sets. Each sample contains a unique background, meaning no two samples can share the same background. Using different background datasets means we can further analyze how the selection of random backgrounds can influence the final task-based performance or whether any unique background is helpful. Samples from the two replaced backgrounds datasets are in Figures 7.8 for the Active-Vision real-world dataset, and 7.9 for the photorealistic synthetic dataset.

7.4.2 Photorealistic Background

The results for the detection and segmentation tasks using the synthetic photorealistic dataset are in Tables 7.11 and 7.12, broken down by AP scores per category.

Figure 7.8: Visualization of training images from the SRDR dataset described in Chapter 6 using replaced backgrounds from the Active-Vision dataset [4]. The set of backgrounds are from a real-world dataset and contains background clutter in the form of additional household objects. Note that there are some instances where real bananas appear in the background. These are not labeled as a sample in a dataset.

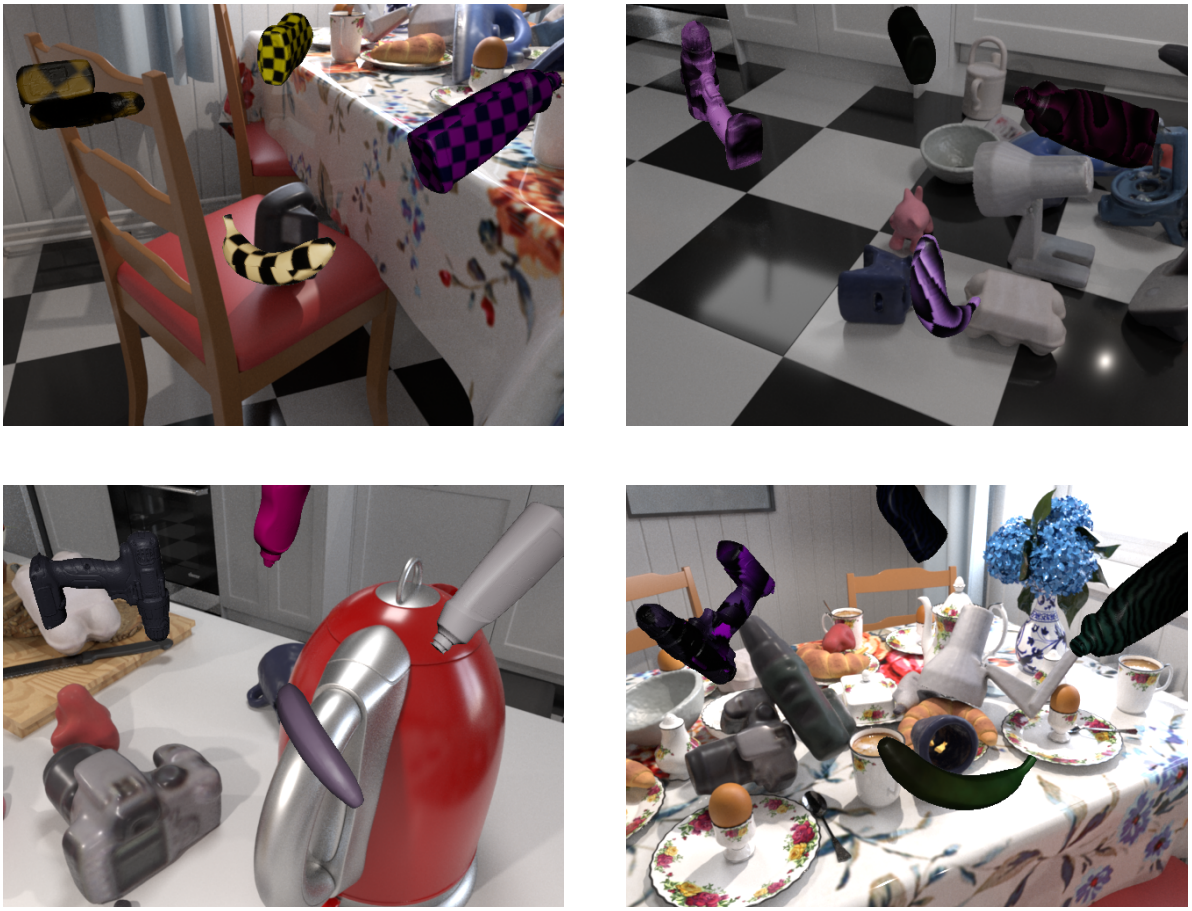


Figure 7.9: Visualization of training images from the SRDR dataset described in Chapter 6 using replaced backgrounds from the Photorealistic dataset [74]. The backgrounds used are from a photorealistic synthetic dataset containing a high degree of clutter from household objects. None of the objects in the backgrounds are included in the training dataset.

Object Detection

In Table 7.11, despite seeing variation in performance depending on the textures used, per-category detection for all classes apart from banana does not work. The current model using random poses cannot transfer to the real world for categories other than bananas. Compared to the IRLab real-world background performance using the same highest-performing Striped texture, in Table 7.9, the photorealistic synthetic background dataset results an AP score of 8.283 compared to the IRLab background resulting in a score of 15.971 AP. The significant degradation in performance may be twofold: one, the backgrounds used are still synthetic and appear quite different from real-world images, and two, there is significantly more clutter in the backgrounds as seen in Figure 7.9. This increase in clutter may have a non-trivial impact on how the network is challenged during the learning process.

Semantic Segmentation

In Table 7.12, similar performance is observed, where the trained models for each of the texture randomization methods used cannot transfer to the real world when segmenting objects that are not the banana, with a similar disagreement between backgrounds used as in the detection task.

7.4.3 Active-Vision Background

The results for detection and segmentation using the Active-Vision real-world dataset are in Tables 7.13 for detection, and 7.13 for segmentation.

Texture	AP	Power Drill	Bleach Cleanser	Banana	Mustard Bottle
Gradient RGB	4.880	0.000	0.000	19.521	0.000
Flat RGB	3.979	0.000	0.000	15.915	0.000
Gradient RGB Perlin	2.953	0.000	0.000	11.812	0.000
Flat RGB Perlin	4.216	0.000	0.000	16.863	0.000
Zig Zag Perlin	3.760	0.000	0.000	15.041	0.000
Striped Perlin	5.150	0.000	0.000	20.598	0.000
Zig Zag	7.457	0.000	0.000	29.829	0.000
Striped	**8.283**	0.000	0.000	**33.133**	0.000
Checkerboard Perlin	3.514	0.000	0.000	14.055	0.000
Checkerboard	7.583	0.000	0.000	30.331	0.000
Real-texture	4.365	0.000	**0.246**	17.215	0.000
Real Images (YCB-M)	69.479	59.902	85.466	64.321	68.228

Table 7.11: Per-Category object detection (bounding box) AP using COCO weights [116]. The network was fine-tuned using synthetic images with the original object textures defined as Real-Texture. The remaining datasets are the ten texture DR techniques used in the current literature. Each synthetic dataset uses a unique background per frame from a synthetic photorealistic dataset [74]. The dataset is described in detail in Chapter 6. Each model was evaluated on a real-world test set from a single scene shown in Figure 7.5 containing the four objects from the YCB-M dataset [67].

Texture	AP	Power Drill	Bleach Cleanser	Banana	Mustard Bottle
Gradient RGB	4.096	0.000	0.000	16.383	0.000
Flat RGB	3.477	0.000	0.000	13.909	0.000
Gradient RGB Perlin	2.503	0.000	0.000	10.012	0.000
Flat RGB Perlin	3.442	0.000	0.000	13.768	0.000
Zig Zag Perlin	3.430	0.000	0.000	13.718	0.000
Striped Perlin	4.800	0.000	0.000	19.200	0.000
Zig Zag	6.593	0.000	0.000	26.370	0.000
Striped	**7.864**	0.132	0.000	**31.322**	0.000
Checkerboard Perlin	3.135	0.165	0.000	12.377	0.000
Checkerboard	6.439	0.222	0.000	25.534	**0.696**
Real-texture	4.447	**0.938**	**0.054**	16.797	0.000
Real Images (YCB-M)	61.619	46.028	79.855	55.876	64.716

Table 7.12: Per-Category object semantic segmentation AP using COCO weights [116]. The network was fine-tuned using synthetic images with the original object textures defined as Real-Texture. The remaining datasets are the ten texture DR techniques used in the current literature. Each synthetic dataset uses a unique background per frame from a synthetic photorealistic dataset [74]. The dataset is described in detail in Chapter 6. Each model was evaluated on a real-world test set from a single scene shown in Figure 7.5 containing the four objects from the YCB-M dataset [67].

Object Detection

When using the Active-Vision dataset, Table 7.13 shows the detection results where we see Checkerboard Perlin outperforming the Striped texture using the IRLab backgrounds. However, it still does not outperform the real-world equivalent textures using the IRLab backgrounds from Table 7.9. Similarly to the results from the synthetic photorealistic backgrounds, we are still not capable of transferring using the current random poses dataset for classes that are not the banana.

Texture	AP	Power Drill	Bleach Cleanser	Banana	Mustard Bottle
Gradient RGB	12.533	1.499	0.000	48.633	0.000
Flat RGB	12.285	0.174	0.000	48.968	0.000
Gradient RGB Perlin	13.146	0.059	0.000	52.452	0.000
Flat RGB Perlin	13.189	0.348	0.000	52.409	0.000
Zig Zag Perlin	14.140	1.686	0.000	54.875	0.000
Striped Perlin	14.239	**2.720**	0.020	54.216	0.000
Zig Zag	14.483	0.945	0.078	56.909	0.000
Striped	14.532	1.213	0.344	56.573	0.000
Checkerboard Perlin	**15.385**	1.742	0.020	**59.779**	0.000
Checkerboard	14.301	1.257	0.185	55.082	**0.678**
Real-texture	12.037	0.057	**0.794**	47.295	0.000
Real Images (YCB-M)	69.479	59.902	85.466	64.321	68.228

Table 7.13: Per-Category object detection (bounding box) AP using COCO weights [116]. The network was fine-tuned using synthetic images with the original object textures defined as Real-Texture. The remaining datasets are the ten texture DR techniques used in the current literature. Each synthetic dataset uses a unique background per frame from the real-world Active-Vision dataset [4]. The dataset is described in detail in Chapter 6. Each model was evaluated on a real-world test set from a single scene shown in Figure 7.5 containing the four objects from the YCB-M dataset [67].

Semantic Segmentation

Similarly, Checkerboard Perlin is also the highest performing texture for solving segmentation using the Active-Vision background. However, we still see issues with transfer outside of the banana class as shown in Table 7.14.

Texture	AP	Power Drill	Bleach Cleanser	Banana	Mustard Bottle
Gradient RGB	12.915	1.485	0.000	50.177	0.000
Flat RGB	12.761	0.223	0.000	50.821	0.000
Gradient RGB Perlin	12.721	0.059	0.000	50.825	0.000
Flat RGB Perlin	13.160	0.401	0.000	52.238	0.000
Zig Zag Perlin	12.825	0.981	0.000	50.319	0.000
Striped Perlin	13.315	**1.724**	0.000	51.536	0.000
Zig Zag	13.215	0.485	0.000	52.376	0.000
Striped	13.443	0.609	0.000	**53.161**	0.000
Checkerboard Perlin	**13.444**	0.842	0.000	52.936	0.000
Checkerboard	13.365	0.440	0.000	52.325	**0.696**
Real-texture	12.380	0.694	**0.343**	48.485	0.000
Real Images (YCB-M)	61.619	46.028	79.855	55.876	64.716

Table 7.14: Per-Category object semantic segmentation AP using COCO weights [116]. The network was fine-tuned using synthetic images with the original object textures defined as Real-Texture. The remaining datasets are the ten texture DR techniques used in the current literature. Each synthetic dataset uses a unique background per frame from the real-world Active-Vision dataset [4]. The dataset is described in detail in Chapter 6. Each model was evaluated on a real-world test set from a single scene shown in Figure 7.5 containing the four objects from the YCB-M dataset [67].

Discussion

It is clear from the previous results that the current dataset using random poses is quite limited in terms of achieving transfer from synthetic to real. Despite modifying the backgrounds to aid in the randomization process, the training size of 5400 images for the four objects does not appear to be sufficient in helping transfer to the real world. The models are consistently failing to detect and segment objects that are more complex than the banana. This may stem from the similarities in the shapes of the mustard bottle and bleach cleanser, or the tip of the drill and tip of the bottles appearing visually similar. The extreme orientation randomizations may be causing the networks to fail to generalize to the real-world test dataset, where the objects are upright and stationary, meaning few samples may contain poses similar to the real-world test set. Using separate unique backgrounds does however, reduce the number of false positives that arises from using a limited set of backgrounds as shown in Figure 7.10 compared to Figure 7.7 when using the IRLab backgrounds.

To further probe this idea of similar poses, the next set of experiments evaluates the performance of the models trained on synthetic scenes that replicate the YCB-M dataset, eliminating the effects of object poses for the following investigations. Replicating scenes means we can analyze how texture randomizations and selection of image backgrounds affect final task-based performance.

7.5 Scene Replication

The models could not transfer from synthetic to real for three out of the four possible classes from the previous results using random poses and random backgrounds. The hypothesis is that the random poses provided in the training set of 5400 images were not adequate to aid in transfer, as the orientation randomizations were significantly different from the real-world test set. For example, the training set would have included several samples of the bottles from the underside,

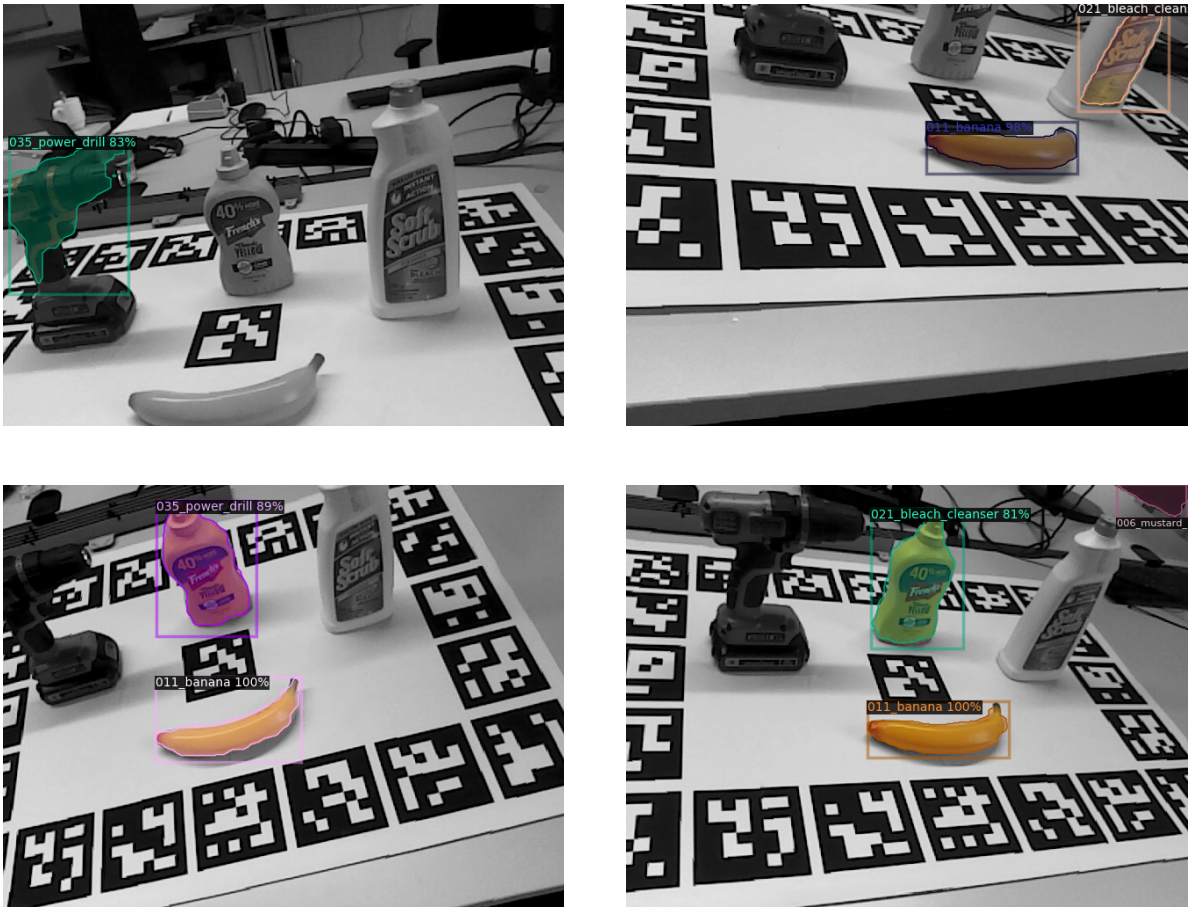


Figure 7.10: Visualization of the network predictions on a real-world dataset when using Checkerboard weights and using replaced backgrounds from the Active-Vision dataset [4]. Using a unique background per scene in the training set eliminates the false positives previously seen in Figure 7.7. While there are correct, true positive and true negative predictions, there are still some false negatives with missed detections of objects and misclassifications of them.

which does not appear in the test set. Significantly more training samples would be required to overcome the domain gap, such that the positions and orientations of the test set would seem like another sample from the training set. To reasonably analyze how various texture randomizations affect task-based performance, several other factors of influence must be removed or diminish the effects of. From Table 4.2 of common randomization techniques, this includes objects and poses, camera positions, backgrounds, and illumination. The next set of experiments fixes the above, such that the only influencing factor would be the textures that are applied, as the use of textures is one of the more significant factors for achieving transfer from synthetic to real [1, 16, 58].

7.5.1 Experimental Setup

To investigate the influence of DR textures when generalizing to multiple tasks, the SRDR dataset detailed in Chapter 6 is used to aid in this investigation. The dataset replicates all real-world scenes from the YCB-M dataset consisting of 31 different scenarios with various scene and object complexities. The training set consists of 3935 images, and the test set is 837 images, with the detailed breakdown of the instances shown in Table 7.15. The test set contains 5 out of the 31 scenes held out and not visible during the training phase, ensuring no related frames are observed during training. The SRDR dataset contains manually approximated illumination for each of the 31 different scenes from the YCB-M dataset, attempting to match the lighting in the real-world scenes. The object positions and camera trajectories, and angles are also matched. For each of the replicated scenes, each object is randomized from a collection of commonly applied textures based on existing literature from Table 4.3. Additionally, for each of the DR texture randomizations used, five different backgrounds are used to investigate background usage from the SRDR dataset, and one network is trained per texture and background combination. These backgrounds are of various complexities, ranging from non-photorealistic synthetic, photorealistic synthetic, and real-world scenes. Samples from the SRDR dataset backgrounds are in Chapter 6.

Dataset	cheezit	sugar	soup	mustard	tuna	pudding	gelatin	spam	banana	pitcher	bleach	bowl	mug	drill	wood	scissors	largeMarker	largeClamp	xLargeClamp	foam	Total Images
Train	899	880	914	1382	870	1075	857	892	1167	891	1049	904	1040	1492	1060	1034	839	925	930	1156	3935
Test	160	337	341	168	320	169	160	334	167	172	168	168	169	168	328	169	312	172	160	149	837

Table 7.15: Instances per class for a synthetic dataset replicating real-world scenes from the YCB-M dataset [67]. The scene replication process is described in more detail in Chapter 6 From the 31 scenes in the YCB-M dataset, 26 were separated for training and validation, and 5 were used as the test set. The split ensures that all objects are represented in the test set, and no frames from the test set appear in the training sets.

7.5.2 Real-Texture Equivalent

Table 7.16 shows the results for the object detection and segmentation tasks when evaluating the trained models on synthetic and real-world images. Since the synthetic scenes are replicated versions of the real-world test set, the synthetic evaluations contain the same poses, camera positions, and approximate illumination as the real-world versions.

Object Detection

When evaluating the model using the IRLab background and a synthetic test set, we see high performance across all object classes when evaluating on a similar target domain as seen in Table 7.16. It is worth noting that the bowl, power drill, and bleach cleanser are still more challenging to detect compared to some of the other categories, such as the mug, large marker, and mustard bottle, suggesting that the problem at hand is not highly trivial even in the target domain.

When evaluating the model on the real-world dataset from the YCB-M dataset for the object detection task, it is expected to see worse performance due to domain mismatch. We are evaluating our models on a different domain - meaning our synthetic models are evaluated on real-world images. Interestingly, the results in Table 7.16 show two objects achieved a higher performance in the real-world domain compared to synthetic ones. The two objects are the Cheez-It box (AP of 54.816 on the real-world dataset compared to 48.254 AP on the synthetic dataset) and the bleach bottle (55.857 AP on the real-world dataset compared to 45.961 AP on the synthetic dataset). The model may be overfitting on the object poses it sees for those two classes due to their shape, and a larger dataset may not have similar anomalies. The remaining classes all had worse performance on the real-world dataset.

Task	Test Set	cheezit	sugar	soup	mustard	tuna	pudding	gelatin	spam	banana	pitcher	bleach	bowl	mug	drill	wood	scissors	largeMarker	largeClamp	xLargeClamp	foam
Detection	Synthetic	48.254	66.536	77.467	86.561	71.060	83.327	83.327	60.928	77.618	68.464	45.961	31.609	92.735	42.196	72.611	65.686	89.552	65.413	53.110	70.361
Detection	Real-World	54.816	38.312	36.711	36.311	26.185	8.506	1.958	9.202	7.609	55.843	55.857	6.043	56.619	2.967	67.784	20.206	6.047	0.912	2.541	4.783
Segmentation	Synthetic	89.681	79.817	78.629	90.062	71.060	86.188	77.858	67.436	84.823	90.029	81.450	61.355	90.003	62.528	81.839	48.884	82.649	77.004	54.915	77.358
Segmentation	Real-world	64.646	46.632	32.447	42.193	20.262	7.361	1.883	7.044	0.007	41.073	63.259	6.043	56.619	0.062	70.814	2.228	5.049	0.020	1.652	4.473

Table 7.16: Per-Category object detection (bounding box) and semantic segmentation AP scores using models trained with the real-texture synthetic images from the SRDR dataset with the IRLab background described in Chapter 6. The models are evaluated using five of the replicated synthetic real-texture scenes (defined as "Synthetic") and their real-world equivalents (defined as "Real-World") from the YCB-M dataset [67]. A breakdown of the instances in the test set is in Table 7.15.

Semantic Segmentation

Generally, the same pattern follows where the network can predict correct segmentation masks for each class when evaluating on a similar target domain as shown in Table 7.16. However, scissors is one of the few classes that are more challenging to segment. One plausible hypothesis for this is due to the shape of the object itself and that it is generally more occluded than some of the other classes in the training set, as seen in Figure 7.11. The figure shows the first scene (left) and second scene (right), which contains the scissors. Note that a large portion of the samples containing this object is partially visible and is challenging to learn from, which may explain the lower performance compared to other classes.

When evaluating the model on the real-world dataset from the YCB-M dataset for the object segmentation task, it is expected to see worse performance due to domain mismatch. This expected result is shown in Table 7.16, where we see significant performance degradation when evaluating our models trained on synthetic images and evaluated on real-world images instead of the synthetic ones.

7.5.3 Varying Backgrounds

The types of backgrounds used are often overlooked when applying DR and may potentially influence the final task-based performance. Background features such as similar objects of interest could be false positives, or increased background clutter may act as distractor objects, challenging a given network during the training process. Tables 7.17 and 7.18 shows the results evaluating the detection and segmentation models using the same training dataset consisting of the same textures, poses, camera positions, and illumination except for changing the type of backgrounds used for each scene.

Interestingly, the types of backgrounds used for the tasks can affect the final task-based per-

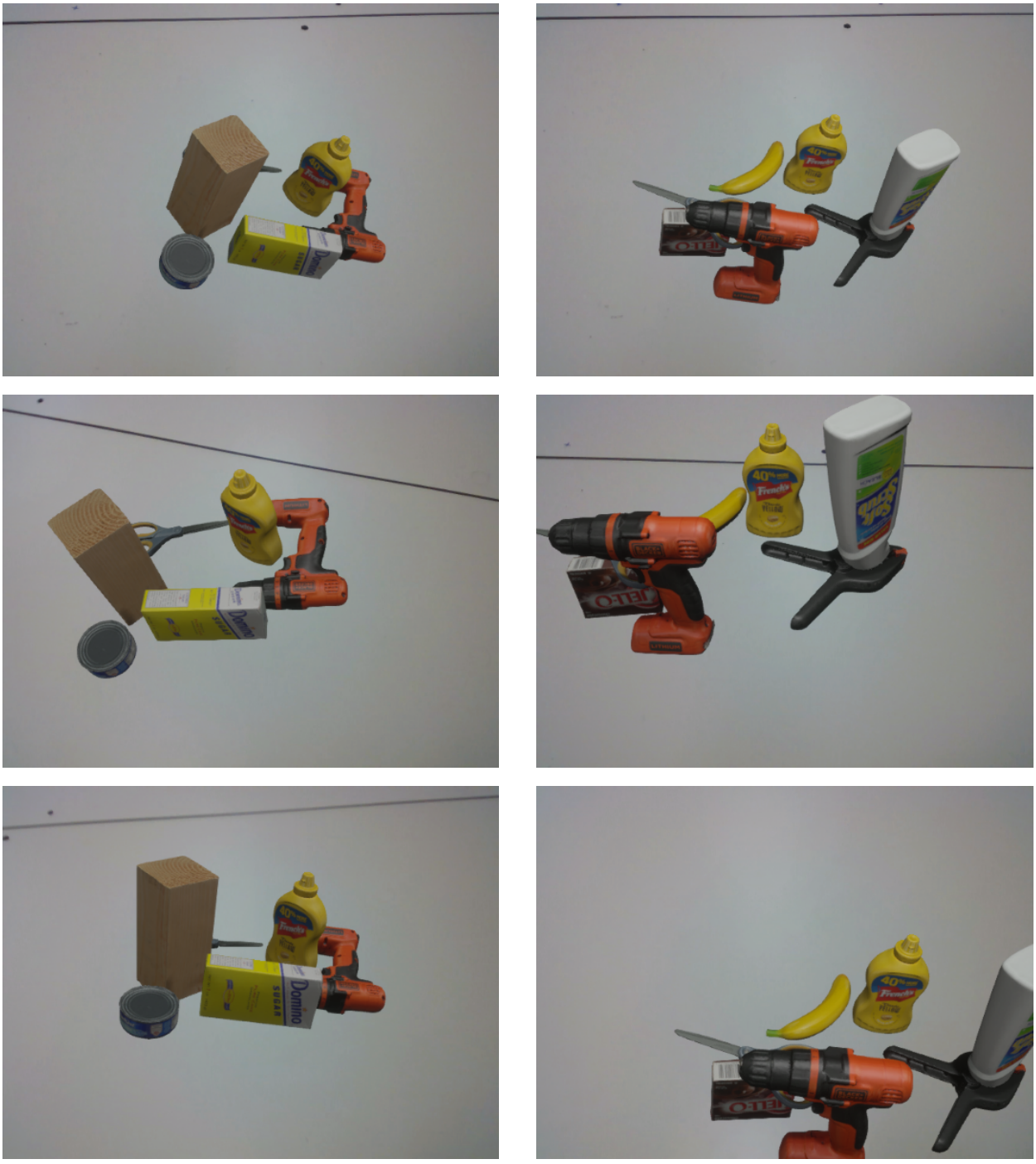


Figure 7.11: Figure showing several samples of the scissors class from the training set. The two scenes depicted on the left and right are the only scenes containing the scissors class. Note that the object is partially occluded in most instances and would be challenging to learn from these samples.

Dataset	AP	AP50	AP75	APs	APm	APl
Scenenet [132] (synthetic)	22.846	44.448	20.026	0.539	20.888	24.452
Photorealistic [74] (synthetic)	25.021	45.027	25.946	0.303	19.173	26.451
Structured3D [244] (synthetic)	28.141	51.781	26.25	0.813	24.172	**35.344**
Active Vision [4] (real world)	**29.813**	**58.452**	**26.942**	**0.982**	**25.963**	34.024
Real World	39.403	72.902	39.072	6.475	36.93	39.386

Table 7.17: Comparative results on a real-world test set of 837 images. All models were trained using Mask R-CNN using the ResNet-50-C4 backbone. Backgrounds of the synthetic training sets were replaced with images from synthetic and real datasets. Scores shown are for object detection (bounding box).

Dataset	AP	AP50	AP75	APs	APm	APl
Scenenet [132] (synthetic)	21.519	38.747	20.316	0.116	18.333	30.048
Photorealistic [74] (synthetic)	22.85	38.917	24.047	0.016	17.159	28.022
Active Vision [4] (real world)	26.532	46.073	25.5	**0.26**	**21.237**	**38.858**
Structured3D [244] (synthetic)	**26.797**	**46.098**	**26.383**	0.111	20.786	37.335
Real World	37.198	63.169	36.717	0.13	31.452	45.976

Table 7.18: Comparative results on a real-world test set of 837 images. All models were trained using Mask R-CNN using the ResNet-50-C4 backbone. Backgrounds of the synthetic training sets were replaced with images from synthetic and real datasets. Scores shown are for the segementation task.

formance. For example, the difference between the highest performer, the Active-Vision dataset, which comprises real-world images, and the lowest performer SceneNet, consisting of non-photorealistic synthetic images, is approximately 7 AP points worse. Similarly, the Scenenet dataset is just over 5 AP points worse than the highest-performing Structured3D dataset for the segmentation task. This result indicates that the types of backgrounds and complexity, in terms of realism and background clutter, can influence the final task-based performance.

7.5.4 DR Textures

In chapter 5, we found that the selection of textures applied to an object of interest's mesh can degrade task-based performance. For example, when training an object localization network to predict the x, y, z positions of an object of interest using DR data, patterned textures resulted in lower task-based error than the commonly applied Flat RGB textures. This section shows the results for solving object detection and segmentation tasks using commonly applied DR textures using both real-world images and synthetic images as backgrounds.

Photorealistic Dataset

Tables 7.19 and 7.20 show the AP scores across commonly used DR methods for solving detection and segmentation tasks, sorted in ascending order from lowest to highest AP scores. In this scenario, a photorealistic, highly cluttered background is used, and each of the commonly selected DR methods is applied to objects of interest as shown previously in Figure 6.7.

While we can see a difference between the selection of DR method for the detection and segmentation tasks in Tables 7.19 and 7.20, which usually results in higher performance when using patterned textures, the overall performance is relatively poor. One possible explanation for this is the high levels of clutter in the backgrounds for this particular dataset. For example, in Figure 6.7, we see several samples of the dataset where it is often difficult to discern foreground

Dataset	AP	AP50	AP75	APs	APm	APl
Gradient RGB Perlin	4.463	10.031	2.553	0.02	4.077	2.904
Flat RGB Perlin	4.694	10.252	2.712	0.011	4.56	2.129
Striped Perlin	4.93	10.52	3.324	0.16	5.533	0.991
Gradient RGB	5.144	10.542	3.651	0	4.322	2.683
Checkerboard Perlin	5.489	11.774	3.752	0.085	6.231	0.851
Flat RGB	5.522	11.549	4.154	0	5.043	2.281
Zig Zag Perlin	5.758	**14.227**	2.808	0.275	6.322	3.282
Zig Zag	6.097	13.623	4.007	**0.303**	5.879	**4.107**
Checkerboard RGB	6.353	14.104	4.148	0.242	6.434	2.248
Striped RGB	**6.521**	13.126	**4.741**	0.198	**6.721**	2.155
Synthetic Real Textured	25.021	45.027	25.946	0.303	19.173	26.451
Real World	39.403	72.902	39.072	6.475	36.93	39.386

Table 7.19: Table showing comparative results on a real-world test set of 837 images from the YCB-M dataset [67] for the object detection task. All models were trained using Mask R-CNN using the ResNet-50-C4 backbone and each of the texture DR datasets from the SRDR dataset described in Chapter 6. The Synthetic Real Textured dataset is the original object textures, and the Real World dataset is a model trained on the matched real-world dataset from YCB-M [67]. The results shown are from the models trained with synthetic backgrounds from the photorealistic dataset [74] from the SRDR dataset.

Dataset	AP	AP50	AP75	APs	APm	APl
Striped Perlin	3.087	5.947	4.239	0.006	3.527	0.731
Gradient RGB Perlin	3.323	6.113	4.178	0.001	3.26	2.178
Flat RGB Perlin	3.38	6.533	4.157	0	3.448	1.536
Flat RGB	3.412	6.49	3.893	0	3.253	1.355
Gradient RGB	3.761	6.764	4.192	0	3.634	1.315
Checkerboard Perlin	3.797	7.273	4.959	0.002	4.308	1.326
Zig Zag Perlin	3.998	8.001	4.71	0.011	4.188	**4.117**
Zig Zag	4.075	7.903	4.981	0.012	4.087	3.231
Striped RGB	4.077	8.043	5.092	0.005	4.385	1.134
Checkerboard RGB	**4.551**	**8.749**	**5.599**	**0.013**	**4.732**	1.492
Synthetic Real Textured	22.85	38.917	24.047	0.016	17.159	28.022
Real World	37.198	63.169	36.717	0.13	31.452	45.976

Table 7.20: Table showing comparative results on a real-world test set of 837 images from the YCB-M dataset [67] for the object segmentation task. All models were trained using Mask R-CNN using the ResNet-50-C4 backbone and each of the texture DR datasets from the SRDR dataset described in Chapter 6. The Synthetic Real Textured dataset is the original object textures, and the Real World dataset is a model trained on the matched real-world dataset from YCB-M [67]. The results shown are from the models trained with synthetic backgrounds from the photorealistic dataset [74] from the SRDR dataset.

objects of interest from the background objects. In this scenario, the usual DR textures selected in the literature would not be suitable when combined with heavily cluttered scenes.

Scenenet Dataset

Tables 7.21 and 7.22 show the AP scores across commonly used DR methods for solving detection and segmentation tasks, sorted in ascending order from lowest to highest AP scores. In this scenario, an unrealistic background with varying degrees of background clutter is used, and each of the commonly selected DR methods is applied to objects of interest as shown previously in Figure 6.8.

In both tasks, the models using patterned textures typically outperform non-patterned ones. In the detection task results presented in Table 7.21, the AP score for the highest performing method, Striped, is 10.936 AP and using the lowest-performing method, Gradient RGB Perlin, is 4.328 AP. Similarly, in the segmentation task in Table 7.22, the AP score for the highest performing method, Zig Zag Perlin, is 8.122 AP compared to the lowest-performing method, Flat RGB Perlin, is 3.789 AP. However, the AP scores between the top three performers for the detection and segmentation task is not as substantial. Take the detection task, where the top three performers are Striped, Zig Zag Perlin, and Zig Zag. Using Striped results in an AP score of 10.936 AP, while Zig Zag Perlin yields 10.502 AP, meaning a generally more complex pattern results in better performance for segmentation and detection tasks.

Compared to using the previous Photorealistic backgrounds, which are highly cluttered, using less cluttered scenes can increase the performance for both detection and segmentation tasks from 6.521 AP to 10.936 AP and from 4.551 AP to 8.122 AP using the highest performing DR methods, respectively.

Dataset	AP	AP50	AP75	APs	APm	APl
Gradient RGB Perlin	4.328	10.183	3.078	0.288	4.545	1.101
Flat RGB Perlin	4.804	10.602	3.558	0.185	5.186	1.201
Flat RGB	8.818	18.152	7.869	0.253	10.395	3.851
Gradient RGB	9.805	20.069	8.384	0.264	10.324	4.673
Striped Perlin	9.836	21.118	7.623	0.622	10.609	5.045
Checkerboard Perlin	10.049	22.926	7.255	0.484	11.875	6.427
Checkerboard	10.071	21.983	7.799	**0.645**	10.129	6.92
Zig Zag	10.276	22.153	7.579	0.462	10.004	5.961
Zig Zag Perlin	10.502	**23.068**	8.201	0.594	**13.102**	5.713
Striped	**10.936**	22.022	**9.002**	0.442	10.383	**8.329**
Synthetic Real Textured	22.846	44.448	20.026	0.539	20.888	24.452
Real World	39.403	72.902	39.072	6.475	36.93	39.386

Table 7.21: Table showing comparative results on a real-world test set of 837 images from the YCB-M dataset [67] for the object detection task. All models were trained using Mask R-CNN using the ResNet-50-C4 backbone and each of the texture DR datasets from the SRDR dataset described in Chapter 6. The Synthetic Real Textured dataset is the original object textures, and the Real World dataset is a model trained on the matched real-world dataset from YCB-M [67]. The results shown are from the models trained with synthetic backgrounds from the Scenenet dataset [132] from the SRDR dataset.

Dataset	AP	AP50	AP75	APs	APm	APl
Flat RGB Perlin	3.789	8.646	3.083	0.007	4.091	0.982
Gradient RGB Perlin	3.806	8.107	3.277	0.006	4.3	1.067
Flat RGB	6.992	13.474	6.983	0.007	8.617	2.377
Checkerboard	7.283	14.389	8.077	**0.058**	7.191	7.915
Zig Zag	7.512	15.327	7.512	0.06	7.842	5.281
Striped Perlin	7.579	14.999	7.815	0.056	8.087	6.634
Striped	7.801	15.857	7.747	0.023	7.51	7.838
Checkerboard Perlin	7.84	**15.901**	7.638	0.05	8.425	**9.185**
Gradient RGB	7.915	15.717	6.745	0.004	8.958	3.514
Zig Zag Perlin	**8.122**	15.651	**8.896**	0.042	**10.181**	8.058
Synthetic Real Textured	21.519	38.747	20.316	0.116	18.333	30.048
Real World	37.198	63.169	36.717	0.13	31.452	45.976

Table 7.22: Table showing comparative results on a real-world test set of 837 images from the YCB-M dataset [67] for the object segmentation task. All models were trained using Mask R-CNN using the ResNet-50-C4 backbone and each of the texture DR datasets from the SRDR dataset described in Chapter 6. The Synthetic Real Textured dataset is the original object textures, and the Real World dataset is a model trained on the matched real-world dataset from YCB-M [67]. The results shown are from the models trained with synthetic backgrounds from the Scenenet dataset [132] from the SRDR dataset.

Structured3D

Tables 7.23 and 7.24 show the AP scores across commonly used DR methods for solving detection and segmentation tasks, sorted in ascending order from lowest to highest AP scores. In this scenario, a synthetic, photorealistic, low clutter background is used, and each of the commonly selected DR methods is applied to objects of interest as shown previously in Figure 6.5.

Dataset	AP	AP50	AP75	APs	APm	APl
Checkerboard Perlin	13.922	30.41	11.483	0.458	15.2	14.481
Striped Perlin	13.986	30.827	10.512	0.487	14.189	13.604
Flat RGB Perlin	14.875	31.458	11.97	0.528	15.016	12.506
Zig Zag Perlin	15.723	34.197	11.754	0.393	16.219	13.522
Checkerboard	15.963	33.482	12.692	**2.393**	13.059	19.521
Gradient RGB	15.98	34.228	11.8	0.487	16.278	13.555
Gradient RGB Perlin	16.688	33.587	13.005	0.328	16.211	15.093
Flat RGB	16.826	34.664	14.173	0.088	16.287	13.347
Zig Zag	17.869	38.097	13.341	1.7	**16.567**	19.433
Striped	**18.681**	**38.954**	**15.271**	0.577	15.057	**22.293**
Synthetic Real Textured	28.141	51.781	26.25	0.813	24.172	35.344
Real World	39.403	72.902	39.072	6.475	36.93	39.386

Table 7.23: Table showing comparative results on a real-world test set of 837 images from the YCB-M dataset [67] for the object detection task. All models were trained using Mask R-CNN using the ResNet-50-C4 backbone and each of the texture DR datasets from the SRDR dataset described in Chapter 6. The Synthetic Real Textured dataset is the original object textures, and the Real World dataset is a model trained on the matched real-world dataset from YCB-M [67]. The results shown are from the models trained with synthetic backgrounds from the Structured3D dataset [244] from the SRDR dataset.

Dataset	AP	AP50	AP75	APs	APm	APl
Striped Perlin	10.82	21.731	10.909	0.023	10.427	14.921
Checkerboard Perlin	11.374	22.721	11.082	0.022	11.087	18.616
Flat RGB Perlin	11.915	22.813	11.726	0.014	11.056	17.139
Zig Zag Perlin	12.013	24.701	11.059	0.034	11.686	15.501
Gradient RGB Perlin	12.475	25.06	11.495	0.012	11.451	16.325
Checkerboard	13.092	25.74	12.824	0.035	9.601	22.779
Flat RGB	13.425	26.943	13.021	0.004	12.549	14.365
Gradient RGB	14.063	26.654	13.151	0.007	**12.851**	18.231
Zig Zag	14.428	**29.244**	13.252	**0.116**	11.921	22.409
Striped	**14.779**	28.31	**15.503**	0.056	10.567	**23.618**
Synthetic Real Textured	26.797	46.098	26.383	0.111	20.786	37.335
Real World	37.198	63.169	36.717	0.13	31.452	45.976

Table 7.24: Table showing comparative results on a real-world test set of 837 images from the YCB-M dataset [67] for the object segmentation task. All models were trained using Mask R-CNN using the ResNet-50-C4 backbone and each of the texture DR datasets from the SRDR dataset described in Chapter 6. The Synthetic Real Textured dataset is the original object textures, and the Real World dataset is a model trained on the matched real-world dataset from YCB-M [67]. The results shown are from the models trained with synthetic backgrounds from the Structured3D dataset [244] from the SRDR dataset.

Similar to previous findings, the highest performing DR method using this set of backgrounds is patterned textures (Striped and Zig Zag). The AP score for the highest performing method, Striped, is 18.681 AP compared to the lowest-performing, Checkerboard Perlin, is 13.922 AP for the detection task. The AP scores for the segmentation task for the highest performing, Striped, is 14.779 AP and using Striped Perlin as the lowest-performing, is 10.820 AP.

Interestingly, Flat RGB and Gradient RGB performed better than previous findings on other backgrounds, where they usually ranked in the bottom four of the ten implementations. While they do not outperform all patterned texture methods, they perform better than the generally higher ranked Checkerboard pattern in detection and segmentation, albeit by a very small margin. The AP score using Gradient RGB is 15.980 AP compared to the Checkerboard resulting in a score of 15.963 AP in the detection task. Compared to the segmentation task, the difference between the two is higher, where Gradient RGB scores 14.063 AP compared to the Checkerboard's 13.092 AP. However, Striped and Zig Zag are still the highest performers, and the consensus from previous experiments remains consistent, where using patterned textures achieves the highest performance.

In both the detection and segmentation tasks, we see that the AP scores for each applied texture outperform the models with the previous backgrounds used, where one was non-photorealistic, and the other was photorealistic but with a high degree of clutter. With all other parameters remaining consistent, such as poses, illumination, textures, and camera positions, only the backgrounds used in the training set differ. This result indicates a higher degree of realism and reduced background clutter boosts performance. Compared to the previously highest AP scores, which were achieved using Striped textures and the Scenenet background dataset for the detection task and Zig Zag Perlin for the segmentation task, the Striped pattern with the Structured3D background increases performance from 10.936 AP to 18.681 AP and from 8.122 AP to 14.779 AP respectively. This set of results highlights the importance of background selection as part of generating the training set.

Real-world Backgrounds

Tables 7.25 and 7.26 show the AP scores across commonly used DR methods for solving detection and segmentation tasks, sorted in ascending order from lowest to highest AP scores. In this scenario, real-world scenes with low clutter backgrounds are used, and each of the commonly selected DR methods is applied to objects of interest as shown previously in Figure 6.4.

Dataset	AP	AP50	AP75	APs	APm	APl
Flat RGB Perlin	11.956	25.449	10.302	0.429	13.98	6.784
Zig Zag Perlin	12.195	28.684	8.032	0.451	16.287	8.44
Gradient RGB Perlin	12.219	25.629	9.874	0.341	15.254	5.945
Checkerboard Perlin	12.467	27.496	9.63	1.033	17.098	5.549
Flat RGB	13.157	27.444	11.358	0	14.583	8.797
Striped	13.436	28.054	11.807	**3.481**	15.948	9.495
Striped Perlin	13.508	30.053	10.752	0.352	16.635	10.118
Gradient RGB	14.051	27.754	12.747	0	16.249	8.943
Checkerboard	14.741	31.725	12.615	2.046	16.427	11.273
Zig Zag	**16.354**	**34.339**	**13.937**	2.437	**17.309**	**13.453**
Synthetic Real Textured	29.813	58.452	26.942	0.982	25.963	34.024
Real World	39.403	72.902	39.072	6.475	36.93	39.386

Table 7.25: Table showing comparative results on a real-world test set of 837 images from the YCB-M dataset [67] for the object detection task. All models were trained using Mask R-CNN using the ResNet-50-C4 backbone and each of the texture DR datasets from the SRDR dataset described in Chapter 6. The Synthetic Real Textured dataset is the original object textures, and the Real World dataset is a model trained on the matched real-world dataset from YCB-M [67]. The results shown are from the models trained with real-world backgrounds from the Active-Vision dataset [4] from the SRDR dataset.

Dataset	AP	AP50	AP75	APs	APm	APl
Flat RGB Perlin	8.106	17.089	6.791	0.029	9.694	5.562
Gradient RGB Perlin	8.713	18.975	7.662	0.045	11.529	4.992
Flat RGB	8.95	18.159	7.954	0	12.162	5.19
Striped Perlin	9.203	19.534	8.401	0.029	10.235	10.423
Zig Zag Perlin	9.257	19.622	8.277	0.035	11.551	10.312
Checkerboard Perlin	9.577	19.661	9.421	0.018	11.23	9.273
Striped	9.686	19.182	9.665	0.116	10.686	10.262
Gradient RGB	10.117	19.915	8.478	0.005	13.451	6.897
Checkerboard	11.18	23.037	10.486	0.053	10.887	11.78
Zig Zag	**12.771**	**25.544**	**11.877**	**0.258**	**13.817**	**15.455**
Synthetic Real Textured	26.532	46.073	25.5	0.26	21.237	38.858
Real World	37.198	63.169	36.717	0.13	31.452	45.976

Table 7.26: Table showing comparative results on a real-world test set of 837 images from the YCB-M dataset [67] for the object segmentation task. All models were trained using Mask R-CNN using the ResNet-50-C4 backbone and each of the texture DR datasets from the SRDR dataset described in Chapter 6. The Synthetic Real Textured dataset is the original object textures, and the Real World dataset is a model trained on the matched real-world dataset from YCB-M [67]. The results shown are from the models trained with real-world backgrounds from the Active-Vision dataset [4] from the SRDR dataset.

Using real-world backgrounds closely resembles the previous experiments' results, where patterned textures achieved the highest AP scores for both the detection and segmentation tasks. The AP score for the lowest-performing model using Flat RGB Perlin is 11.956 AP, compared to the highest performing Zig Zag which is 16.354 AP. This result is a similar performance difference when looking at the Structured3D dataset comparing the lowest and highest performers of 13.922 AP and 18.681 AP for the detection task. This performance difference is similar to the segmentation task with real-world backgrounds, where the lowest performing Flat RGB Perlin scores 8.106 AP. The highest performing Zig Zag scores 12.771 AP. This result is similar to the Structured3D photorealistic backgrounds difference in performance in the segmentation task. The lowest performing textures resulted in an AP score of 10.820 AP, and the highest performing resulted in an AP score of 14.779 AP.

Like the Structured3D backgrounds, Gradient RGB outperformed one of the patterned textures by increasing the AP score compared to Striped Perlin for the detection task from 13.508 AP to 14.051 AP, and Striped for the segmentation task from 9.686 AP to 10.117 AP. While the performance difference is quite small, it still outperforms one of the patterned textures. Although, the remaining patterned textures still achieve a higher performance across both tasks. Using Zig Zag results in an AP score of 16.354 AP compared to the Gradient RGB's 14.051 AP for the detection task, and 12.771 AP compared to 10.117 AP when using Zig Zag patterns compared to Gradient RGB for the segmentation task.

Similar to Structured3D backgrounds, the AP scores achieved were higher than the non-photorealistic Scenenet low clutter backgrounds and photorealistic highly cluttered backgrounds, which echoes the findings that increased realism with low clutter would achieve higher overall performance.

7.6 Discussion

Overall, we have examined several stages of the DR pipeline by probing the poses used, backgrounds, and textures to conclude that the hypothesis that DR is task-agnostic. This section discusses our findings supporting this hypothesis and provides suggestions for improving the DR strategy for solving various tasks.

7.6.1 Poses

From the experiments conducted using random object poses and scene replication, it is evident that restricting the poses to scenarios that would be more plausible in the test set would be more beneficial. In section 7.5.3, we conducted an experiment that randomized both the position and orientation of four objects of interest and applied the different DR texture techniques to each of the four objects. We also replaced the image background with real-world images from the Active-Vision dataset for a total of 5400 training images. The highest performing technique for the detection and segmentation tasks was Checkerboard Perlin in this scenario. However, the difference between using the highest performing DR method compared to real-world images is significantly higher for both detection and segmentation tasks, with an AP score for the Checkerboard Perlin at 15.385 AP, and the real-world images at 69.479 AP, and 13.444 AP compared to 61.619 AP respectively. Compared to the replicated scenes using fewer training images and the same image backgrounds, albeit using poses that are more indicative of the target test set, the performance difference is lower, with the highest performing DR method resulting in an AP score of 16.354 AP, and the real-world a score of 39.403 AP for the detection task. For the segmentation task, the highest performing DR method resulted in an AP score of 12.771 AP, and the real-world model resulted in an AP score of 37.198 AP.

While the goal of the experiment is not to achieve comparable performance to the real-world test set, the performance gap is nonetheless reduced when using poses that are more similar

to the target test set, as indicated by a lower difference between the real-world and highest performing DR method for detection and segmentation, while also using fewer training samples at 3935 images for the training set. The results here support the work by [160], where the authors suggest enforcing constraints on the structure and context of the scene increases performance when solving 2D object detection in urban driving environments, compared to traditional DR, which places all objects and distractors randomly.

Complete randomization of poses may be computationally wasteful if specific orientations are not common in the target test data. We can infer the types of poses to generate based on the scenario of the problem we are trying to solve, such as including poses in orientations that would be physically plausible. Although enforcing constraints on the generated poses is more beneficial for generalization with fewer images.

7.6.2 Backgrounds

Randomizing the image backgrounds is typically part of the DR process, where in section 7.5.3 we conducted an experiment to investigate how performance varies when selecting a range of image backgrounds as part of the training set. The image backgrounds used ranged from synthetic non-photorealistic with little clutter, synthetic photorealistic with high clutter, and photorealistic with low clutter. We also looked at applying real-world image backgrounds, as used in some existing works in this field [45, 160, 192]. We may think of replacing the backgrounds in the datasets as a way to force a model to learn from the objects of interest rather than associate the background to the objects of interest, adding a unique background that is not visible in any other sample from the training set guarantees this. However, the types of backgrounds used may act as additional clutter or as distractor objects that may increase the difficulty of the learning process.

From the experiments conducted in section 7.5.4, using real-world image backgrounds instead of non-photorealistic synthetic backgrounds increases performance for both detection and

segmentation tasks from 22.846 AP to 29.813 AP and from 21.519 AP to 26.532 AP respectively when training models using the real-texture synthetic data. We typically see an increase in realism increases performance, where the highest performing models use the real-world backgrounds from the Active Vision dataset.

The Structured3D dataset increases performance over the highly cluttered Photorealistic dataset, achieving an AP score of 28.141 AP compared to the Photorealistic dataset of 25.021 AP for the detection task. The Structured3D dataset also resulted in an AP score of 26.797 AP compared to the Photorealistic score of 22.850 AP for the segmentation task. This difference in performance may be due to the high degree of background clutter in the Photorealistic dataset that is not as prevalent in the Structured3D dataset, which acts as distractor objects during the training process.

Replacing the backgrounds of the synthetic training data by segmenting the objects of interest from the background, then replacing individual pixels of each sample with random backgrounds from other datasets is a quick way of increasing data diversity. Although, it is necessary to note that this method eliminates shadows and illumination from interacting with the environment outside of the existing rendered objects of interest. It would be helpful to investigate further how this may affect task performance if we rendered the scenes alongside the objects rather than replacing the backgrounds of the images in the dataset.

7.6.3 Textures

In section 7.5, we looked at how textures play a role in affecting task-based performance in object detection and segmentation tasks. Across multiple image backgrounds and the existing DR textures used in the current literature, we found similar findings to what was presented in chapter 5, where using more complex patterned textures generally outperformed the commonly used Flat RGB method. We also find similar rankings across both the detection, segmentation, and the lo-

calization task presented in Chapter 5, which was solving a different task, using a different dataset, and network architecture. Table 7.27 shows the rankings produced for each of the tasks using the current DR texture techniques used, where we see a similar order across tasks. The table shows that task-based performance generally increases when using more complex patterned textures, showing that DR is task-agnostic.

Ranking	Detection	Segmentation	Localization
1	Zig Zag	Zig Zag	Zig Zag
2	Checkerboard	Checkerboard	Striped Perlin
3	Gradient RGB	Gradient RGB	Striped
4	Striped Perlin	Striped	Checkerboard Perlin
5	Striped	Checkerboard Perlin	Zig Zag Perlin
6	Flat RGB	Zig Zag Perlin	Checkerboard
7	Checkerboard Perlin	Striped Perlin	Flat RGB Perlin
8	Gradient RGB Perlin	Flat RGB	Gradient RGB Perlin
9	Zig Zag Perlin	Gradient RGB Perlin	Flat RGB
10	Flat RGB Perlin	Flat RGB Perlin	Gradient RGB

Table 7.27: Table showing the rankings for object detection from Table 7.25, semantic segmentation from Table 7.26, and localization task from Figure 5.16. A rank of 1 indicates the highest performing texture used when implementing DR. Rankings for the detection and segmentation tasks are using matched real-world scenes and real-world backgrounds [4] from the SRDR dataset described in Chapter 6. Rankings for the localization task are from the experiments conducted in Chapter 5. Despite solving different tasks, using different networks, and different datasets, it is more favorable selecting more complex patterned textures when using DR.

7.6.4 Illumination

Since using the SRDR dataset presented in chapter 6 for the experiments of this chapter, we manually approximated the illumination for each individual set of scenes that replicated the YCB-M real-world dataset. Illumination does play an important role in aiding transfer from synthetic to real. However, it is challenging to approximate this given a set of real images from the YCB-M dataset without information about the lighting accurately. One possible way to approximate the illumination better would be to optimize the lights' positions in the scene using a differentiable renderer, enabling more precise experimentation for lighting, which we could explore.

7.7 Conclusion

In this section, we conducted a comprehensive study on the generalizability of DR across multiple tasks. We found that DR is task-agnostic and behaves similarly when solving object detection, semantic segmentation, and localization tasks. We have shown this through experiments on complex scenes involving varying levels of occlusion, clutter and scenes with multiple object classes, unlike existing works which focus on single instances or primitive object shapes. We have shown that imposing constraints on the poses available in the training set increases task-based performance, showing that pose structure improves performance. Furthermore, we have shown that the selection of backgrounds plays an important role, and using unique backgrounds with increased realism also increases performance. Finally, the choice of texture DR techniques can significantly influence performance, as also discussed in chapter 5, however, we have shown that the use of DR is task-agnostic, and the use of more complex patterned textures generally increases performance.

While the existing DR methods achieve good performance in specific scenarios, it still requires the design and selection of base patterned textures to generate such as checkerboard, zig zag, or striped, as performed in the current literature. We have seen that DR methods typically

increase task-based performance when using more complex patterns for the objects of interest. The next chapter will present a novel framework for synthesizing textures that outperform the existing DR techniques by generating textures conditionally based on objects of interest.

Chapter 8

Conditional Domain Randomization: Synthesizing Textures via Image Patches

In the previous Chapter 7, we have seen that the selection of DR parameters such as object poses, types of backgrounds, and types of textures influences task-based performance. The performance across different tasks follows a similar ranking. More complex patterned textures such as Zig Zag or Checkerboard resulted in the highest performance for object detection and object segmentation tasks shown in experiments conducted in chapter 7. This result is consistent with the findings for the object localization task in less complex scenes in Chapter 5.

This chapter investigates an alternative method of generating DR textures by eliminating the decision-making on the types of textures to use in the DR process. For example, how should we manually define the DR textures to select from that yield the highest performance? The alternative approach is fast, easy to execute, and outperforms the most commonly used and highest performing DR textures for solving vision tasks on a challenging dataset.

Our approach uses randomly sampled cropped image patches from real-world objects as the textures for the 3D object meshes. We show that using textures from random classes of real-world

objects greatly improves performance over the most commonly used DR texture (Flat RGB) in an object detection task from 13.157 AP to 19.196 AP, and an object segmentation task from 8.950 AP to 17.074 AP. It also outperforms the best DR texture (Zig Zag) in detection and segmentation tasks from 16.354 AP to 19.196 AP, and from 12.771 AP to 17.074 AP, respectively.

Furthermore, we show that conditionally applying the textures using textures from the same object class further increases performance compared to the most commonly used DR texture (Flat RGB) going from 13.157 AP to 21.287 AP, and from 8.950 AP to 19.481 AP in object detection and object segmentation task, respectively. It also outperforms the best DR texture (Zig Zag) in detection and segmentation tasks, increasing from 16.354 AP to 21.287 AP, and from 12.771 AP to 19.481 AP, respectively.

Finally, we extend the idea to propose a conditional texture generation routine based on GANs that conditionally generate textures from similar object classes to the YCB-M dataset [67] and show that the synthetic data generation routine outperforms existing methods. The conditional texture generation routine is helpful in scenarios where we would like to increase the texture diversity (generate more textures) when we only have access to a limited amount of real-world data to sample cropped image patches randomly.

8.1 Introduction

There are numerous scenarios where we would like to use vision systems in the real world. Take, for example, a kitchen countertop, where we may have a robotic arm tasked to store some objects away. There are multiple steps in solving this problem: first, an object detection system must classify the type of objects, such as a cereal box, to select an appropriate grip that would not damage the contents. Second, the detection system must predict a rough estimate of the position of the object, which we can finally pass to a motion planner to plan an appropriate trajectory for picking up and moving the desired object. Understanding what and where things are plays an

important role in solving this task.

One of the most significant barriers to solving such problems is access to large amounts of high-quality annotated data describing where and what each object is in a scene. Synthetic data has helped us gather large quantities of labeled datasets for such scenarios; however, they typically underperform systems trained on real data due to domain gap. We have shown that DR is a promising method for helping to bridge the domain gap from synthetic to real. However, the current utilization typically requires manually defining a data distribution to sample values from and not relevant to object classes. For example, if we wanted to generate DR textures, we would have to decide what type of textures, such as flat RGB, checkerboard, or zig zag as shown in Figure 5.1 and apply the textures to objects in a synthetic scene.

Going back to our kitchen countertop examples, we may have object categories that share similar features: a cereal box, a cracker box, a pudding box, or a gelatin box. Visually, their packaging has similarities, such as bold colors and patterns, large text, and nutritional information. These visual features would differ from other kitchen objects such as bowls or plates, containing more muted colors or patterns and generally no text.

Current DR implementations sample the textures for each object from some statistical data distribution, which the practitioner manually determines. For example, the most commonly used texture in the existing literature is flat RGB, which is flat shades of a single color applied to objects in the scene. This approach lacks visual information that could be useful for solving vision tasks. For example, cans may appear more metallic,

This chapter introduces CDR: Conditional Domain Randomization, synthesizing more visually relevant textures for solving vision tasks using DR synthetic data. In the previous chapters 5 and 7, we have seen that texture complexity generally improves task-based performance in object localization, detection, and segmentation. A question arises: what type of complex texture aids the learning process for solving such vision tasks? For example, the current literature [15, 52, 155, 195, 202, 207, 237] generally uses flat RGB colors and less commonly used patterns such as

checkerboard patterns. However, the more complex patterns used in the existing literature, as seen in Table 4.3 are by no means an exhaustive set of possible complex textures. For example, since patterns increased task-based performance compared to flat RGB, we may have textures generated from triangles, stars, or dots if we focused on simple geometric shapes. A practitioner would determine the types of textures to generate and programmatically create a collection to apply to object meshes to create a training set. However, this would still not address whether this texture contains the complexity that typically results in higher task-based performance or if a texture is suitable for a given object.

We propose a strategy using CDR to improve task-based performance by focusing on visually relevant features for objects of interest, as shown in the top left of Figure 8.1. Here, we see several objects containing similar features to the objects in the real-world image in the bottom right. For example, the wooden block containing wooden grain, the Jell-o box and mustard bottle containing text, or the tuna can appearing more metallic and similar to a real-world can.

The improvement is to take real-world images of objects, which may be the same class or a random class, and sample from those images squared image patches to use as the textures in the DR process. For example, we would apply cropped image patches from random real-world boxes instead of using flat RGB as the textures for a cereal box when generating DR synthetic data. This approach on the cereal box would enforce more visually relevant features such as patterns or text that typically appear in objects of that class. An example of this is in Figure 8.1, where we see the traditional DR approach (top right), CDR (top left), and real-world images (bottom right) compared. This approach can work well given access to large amounts of real-world data or scenarios with a limited amount of real-world data.

We also present an approach to conditionally generate textures for a given object class using a conditional GAN trained on a small amount of real-world data. The texture generator is advantageous in scenarios where access to real-world data is limited. In this scenario, the texture generator can increase the texture diversity by sampling textures from the trained generator. Please

Figure 8.1: Comparison between CDR (top left) and traditional DR approaches (top right using Flat RGB). The CDR approach applies textures visually more similar to object classes in the target dataset. Bottom left is the synthetic real-texture versions, and bottom right is the real-world sample from the YCB-M dataset [67].

refer to the Appendix A for additional samples for the CDR and conditional GAN-based approach.

Our contributions are as follows:

- We present a novel approach for conditionally generating and applying class-specific DR textures by using image patches from real-world images.
- Our proposed approach outperforms the most widely used DR texture randomization method going from 13.157 AP to 21.287 AP and 8.950 AP to 19.481 AP in object detection and semantic segmentation tasks, respectively. We also outperform the highest performing DR texture going from 16.354 AP to 21.287 AP and 12.771 AP to 19.481 AP in object detection and semantic segmentation tasks, respectively.
- We propose a conditional GAN-based texture generator trained on a few real-world image patches to increase texture diversity and outperform the most commonly applied DR texture randomization method using flat RGB going from 13.157 AP to 20.287 AP and from 8.950 AP to 17.636 AP in object detection and semantic segmentation tasks, respectively. This approach also outperforms the best texture randomization method for object detection and segmentation tasks going from 16.354 AP to 20.287 AP and 12.771 AP to 17.636 AP, respectively.

8.2 Related Work

8.2.1 Domain Randomization

Typical DR approaches use a variety of textures to generate DR images for solving vision and robotics tasks. In previous chapters, we have seen that there is currently no consensus on the most suitable approach for selecting the type of textures to use in the randomization process. Table 4.3 highlights the various methods researchers have used for synthesizing DR images. Typically, these

textures are programmatically defined as a set of colors or patterns that we can sample our textures from. For example, while most works use simple textures [14, 15, 52, 81, 127, 147, 155, 159, 180, 195, 201, 202, 205–207, 219, 237], fewer works also incorporate more complex patterned textures [14, 81, 127, 155, 205–207], which we have now shown to be more beneficial in increasing task-based performance. The approach has worked to solve several tasks, particularly in robotics and vision; however, increased texture complexity would boost performance.

While existing approaches have used real-world images as part of the randomization process [15, 205, 232], these are typically used to randomize the backgrounds, such as the work by Tremblay et al. [205], where 10000 real-world images were used to randomize the background of the synthetic dataset. Approaches may also use real-world images to supplement DR data, as is the case with Bousmalis et al. [15], where they used 9.4 million unlabelled real-world images to develop a generative model to improve grasping performance. Currently, existing DR methods do not apply textures as patches sampled from real-world images. We hypothesize this approach would improve task-based performance due to complex properties with natural images that are more difficult to attain with artificially created ones. Such properties include patterns, text, or barcodes from boxes, metallic surfaces for cans, or wooden grains for a wooden block. These features can be attained using our proposed approach and is more difficult to produce through manually defining textures to sample from.

Adding contextual information within DR has recently been explored in work by Prakash et al. [160]. Here, the authors devised a method called Structured Domain Randomization and used an urban driving scene for solving 2D car detection. The authors introduced context into scene generation by placing objects more naturally in a scene per probability distributions from the problem at hand. Meaning placement of roads, signs, or cars would be positioned more naturally, instead of randomly in the camera frame. The authors have shown promising results towards improving the general procedure for DR by focusing on natural object placement but did not focus on texture randomization of the objects in the scene, sampling only from 9 standard colors (Flat RGB), and performing data augmentation such as lightness, roughness, and metallic properties.

We propose that adding visually relevant information to the object's textures would improve the DR process. Combining increased texture complexity from patch-based real-world images and visually relevant information would help improve task-based performance.

8.2.2 Image Synthesis

Synthesizing images is widely explored in the methods using generative adversarial networks [64] or variational autoencoders [98]. More recently, strides have been made towards generating higher quality and higher resolution images using improvements to generative models [18, 87, 88, 90, 140]. The methods have shown significant improvements in image quality, particularly at higher resolutions. Karras, Laine, and Aila [87] proposed an approach motivated by domain adaptation style transfer methods, where the authors proposed a novel generator architecture capable of disentangling high-level attributes such as poses or object classes and allows greater control of synthesized images such as controlling the synthesis of hair on human faces. Karras et al. [90] extended this further by modifying the generator further and training methodology, allowing the synthesis of images with fewer data while maintaining a high level of image quality.

The style-based approach is an appealing choice for synthesizing images over previous methods, which typically required more considerable amounts of data to achieve a similar level of performance. It is also quicker to train and with greater control of image synthesis of high-resolution images compared to alternative methods [18, 140]. For example, a comparative approach by Brock, Donahue, and Simonyan [18] to produce high-quality images at a resolution of 128x128 were evaluated on ImageNet ILSVRC 2012 [178], with a dataset size of 1.2 million images [18] compared to a few thousand training images in the approach by Karras et al. [90].

While the above approaches result in high-quality synthetic images, it is more challenging to generate a specific scene explicitly. For example, we would not be able to generate more complex scenes such as a tabletop scene containing precisely n amount of objects from a specific

camera angle and under specific illumination conditions as we would using a traditional renderer. Furthermore, we would not have access to labeled data that describes the scene using this particular approach to generating synthetic data.

8.2.3 Texture Generation

Texture synthesis outside of traditional rendering pipelines has been explored, particularly in the field of graphics [2, 34, 157, 226]. Typical texture generation pipelines would require artists to manually design and create high-quality images, which can often be time-consuming to generate large quantities of data.

Previously, models were broadly split into parametric and non-parametric techniques, where non-parametric techniques create new textures by repeat smaller-sized patches to synthesize a larger image texture [46, 47, 220], and parametric models use statistical measures for modeling the textures [158]. More recently, neural network generative-based approaches have gained popularity for synthesizing novel textures [2, 34, 157, 226].

For example, Xian et al. [226] used a GAN-based approach for synthesizing textures given an input sketch and texture patches. For example, given a hand-drawn sketch for a handbag, and a sample texture patch, output a handbag with the texture propagated over the sketch. Their method would output a sketch with the texture filling the sketch contours. The system is strongly tied to inputs containing sketch/texture patch pairs, limiting its potential use in scenarios outside of this setting. For example, texturing more complex 3D scenes would not be feasible using this approach. This approach performs well when generating textures for objects such as a wooden block or a mug; however, it would not be able to produce textures containing text, barcodes, which are visually relevant to household objects such as boxes, bottles, or cans.

In our scenario, feeding patches from real-world images to a conditional generator would allow us to maintain more visually relevant features that we can control when using conditional

GAN models, which is desirable when synthesizing textures class-conditioned on the fly.

8.3 Method

The following section details the method for implementing CDR, opening with an overview of the approach for generating textures from image patches. Following this, we describe the original datasets used for generating the image patches, the procedure for producing patches, and details on the conditional GAN approach. The section concludes with information on the training implementation.

8.3.1 Approach Overview

The proposed approach is a simple, fast, and effective method of producing complex patterned textures from real-world images via random sampling of patches. Figure 8.2 shows an overview of the approach:

- Collect images where an object occupies most of the image frame: Start by collecting a set of real-world images. In our scenario, we assume we have access to images with object-centric viewpoints, which occupy most of the image frame.
- Generate patches: Given a set of real-world images, uniformly sample squared patches of size $(n \text{x} n)$.
- Data augmentation: Perform data augmentation on the previously generated image patches. We perform random rotations and random flips to increase dataset diversity further.
- Scene generation: Apply the previously generated set of patches to generate DR synthetic data.

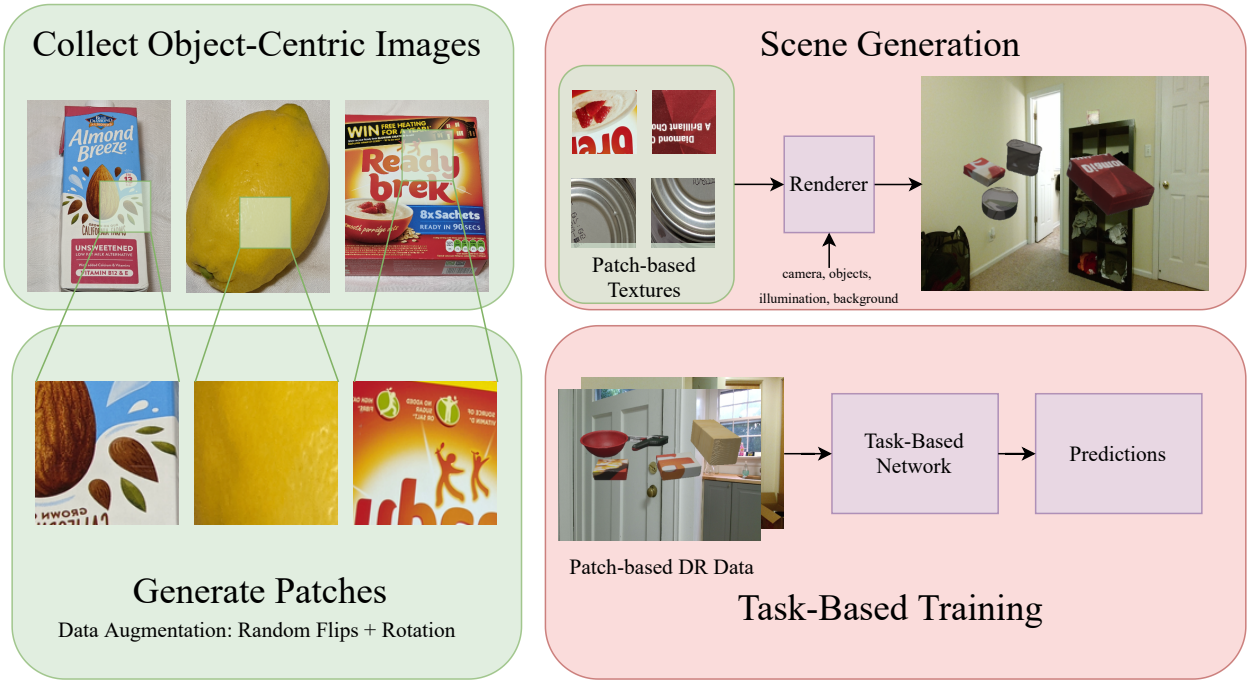


Figure 8.2: We present an approach for synthesizing synthetic images based on real-world image patches for solving detection and segmentation tasks. Natural images provides the desired texture complexity when using DR, while conditionally applying textures based on the objects of interest adds visually relevant information.

- Task-based training: We apply the dataset for solving vision tasks.

In previous chapters, we have seen that texture complexity plays a role in improving task-based performance, although introducing texture complexity can be more challenging than our proposed approach. For example, in the traditional case of DR, we would programmatically specify texture data distributions to sample textures, such as checkerboard, striped, or zig zag. Further complexity can be introduced by adding additional noise, such as using Perlin noise [152] combined with previous textures. While this approach can be beneficial, it requires multiple definitions of data distributions to sample our textures, and we must decide how to model texture complexity best.

In contrast, our approach suggests natural images inherently contain desired properties of texture complexity such as patterns on real-world boxes and outperforms existing DR methods used in the current literature. The process is inexpensive to execute due to the availability of large-scale datasets that we could sample our textures from and does not require handcrafted algorithms for texture generation routines.

8.3.2 Dataset

This section details the texture dataset creation process, which involves generating cropped image patches from real-world images. It also covers the generative approach for synthesizing textures and the creation of the DR synthetic datasets.

Image Collection

Our approach uses patches generated from a collection of two distinct real-world image datasets. The first dataset used is a set of household objects primarily used for training object detection systems [134]. Figure 8.3 shows the collection of household objects in the dataset, which contains

166 RGB images with 13 different object classes. Each image is of size 3264x1836 and contains annotations for 2D bounding boxes for each of the objects visible in a given scene. The textures generated from this set of images are referred to as Texture A.

Figure 8.3: Figure of all object classes used to generate image patches for Texture A. Image taken from [134].

The second dataset used is a novel object-centric household object dataset, which serves as a basis for our experiments' conditional application of textures. This dataset contains classes that are more similar to the target objects. Figure 8.4 shows several samples of the original images from this dataset, which contains 274 images with 7 object classes. Each image is of size 3024x4032 and contains class labels. To supplement this novel dataset, a collection of 15 real-world images of wood is added from Image*After [78]. The textures generated from this dataset will be referred to as Texture B.

Generating Patches

We randomly sample from a uniform distribution across a set of given images to generate patches of size nxn. In the case of patches for Texture A, we use the provided 2D bounding boxes to ensure that each patch generated is contained around the object and not the background. A sample of the

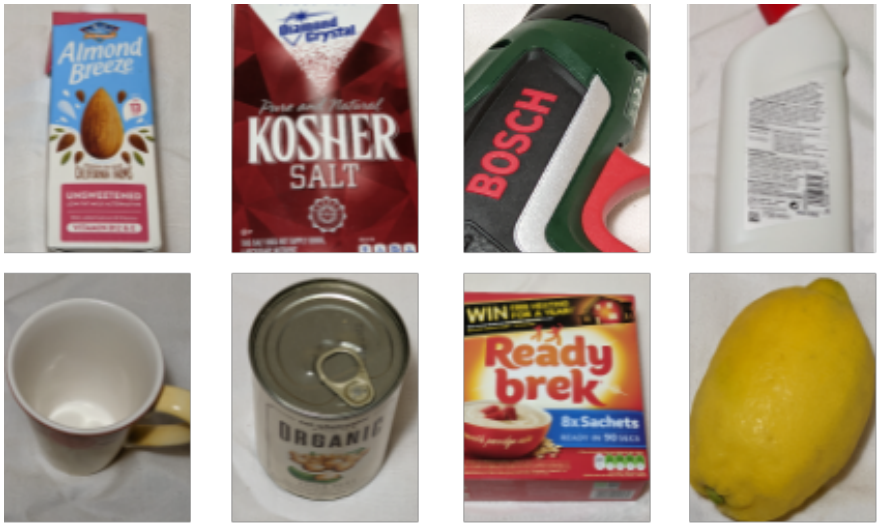


Figure 8.4: Samples real-world objects used to generate image patches for Texture B.

image patches from this dataset is in Figure 8.5, where we generate our textures of size 128x128.

Figure 8.5: Texture A: textures generated from image patches. Each patch is of size 128x128, and uniformly sampled from a set of real-world images.

Textures B did not require bounding box positions before cropping, as the dataset was collected to ensure the majority of the object is visible in the frame. Therefore, we uniformly

sample the random image patches across the set of images for Texture B. Figure 8.6 shows a sample of the textures of size 128x128, after being resized from the crops of size 512x512, generated using this method. The approach to generating patches outlined above ensures we capture visually relevant patches for objects in a scene. The necessity of first cropping at 512x512 was due to the object occupying the majority of a high-resolution image. The initial crops at 512x512 ensures the visual features we desire are visible.

Texture Generator

We use a generative model adopted from StyleGAN2-ADA [90] to synthesize new image patches based on the above data. StyleGAN2-ADA is currently a state-of-the-art method for image synthesizing for both conditional and unconditional modeling. Since we have a small number of real-world images, StyleGAN2-ADA is desirable as it can produce high-quality images when trained on limited data. Previous conditional and unconditional GAN-based approaches typically rely on using a more significant number of data [18, 64, 88, 138, 217]. Using StyleGAN2-ADA means we have control over the generated images based on input modalities, such as object classes in our scenario. This approach allows us to generate textures conditioned on the types of objects we have in our scenes.

We use the official StyleGAN2-ADA implementation as provided by the authors [89]. We use the same network architecture along with the standard loss functions for training the generator and discriminator. We use the default training configuration settings, which trains the GAN for 25k iterations, with a batch size of 32, and using the Adam optimizer with a base learning rate of 0.0025 on a single RTX Titan GPU. The size of the image patches used in both the conditional and unconditional models use a fixed size of 128x128, with a total of 10000 image patches for the unconditional models and 10000 image patches per class for the conditional model.

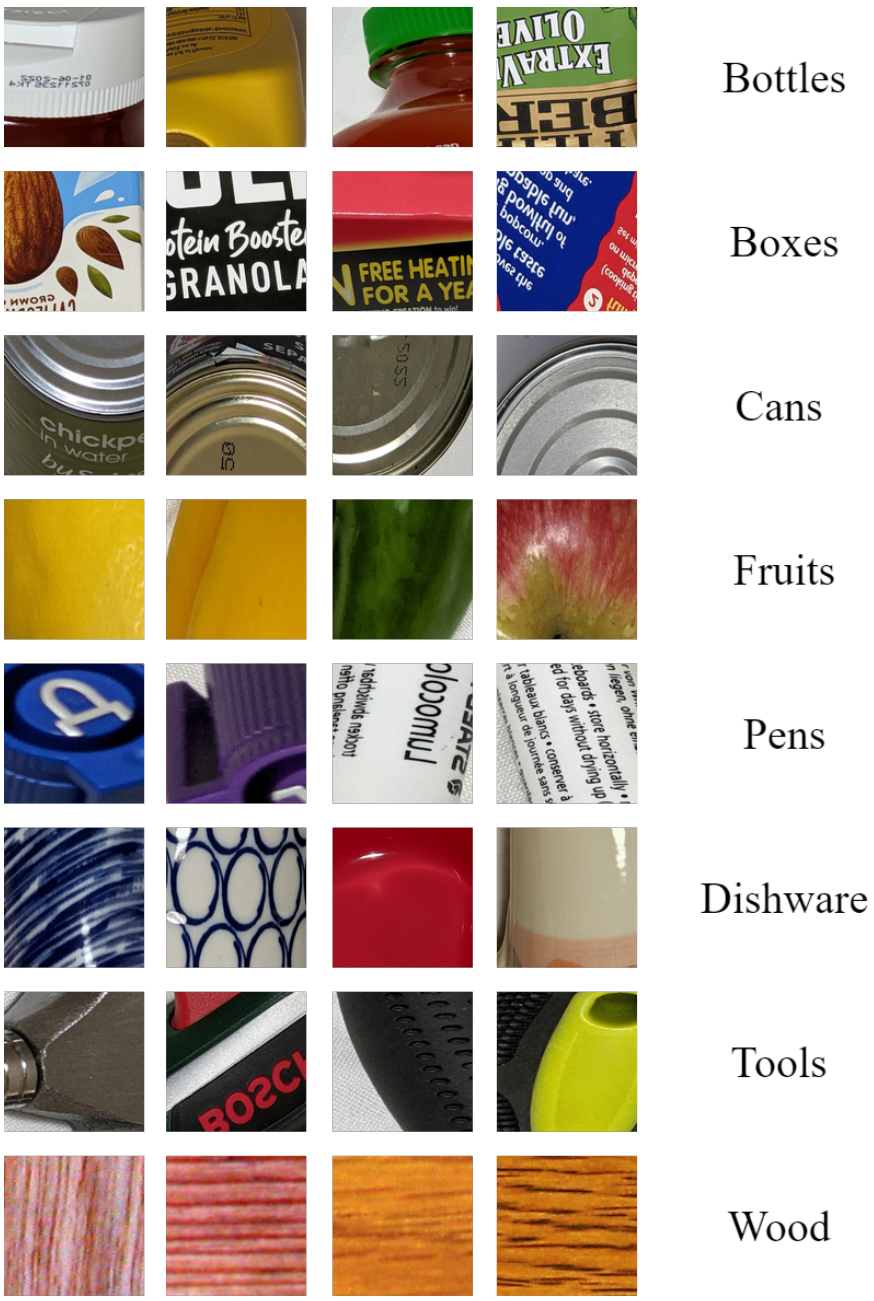


Figure 8.6: Texture B: textures generated from image patches. Each patch is of size 512x512 then resized to 128x128, and uniformly sampled from a set of real-world images.

Applying Patch-Based Textures

We use the SRDR plugin presented in Chapter 6 to replicate real-world scenes from the YCB-M dataset [67]. We fix object poses, illumination, backgrounds, and camera positions across the datasets while replacing the textures of the objects using the generated image patches as shown in Figure 8.1. Further details about the dataset statistics are shown in Chapter 6. We use 3935 synthetic images with the patch-based textures applied for the training set and 837 real-world test images. The backgrounds of the dataset are randomly sampled from the Active Vision dataset, which contains real-world images of indoor household scenes. The Active Vision background dataset resulted in one of the highest task-based performances in matched synthetic and real scenes across commonly used texture randomization methods in object detection and segmentation through experimentation in the previous chapter 6. The results from this set of experiments are in Tables 7.25 and 7.26.

8.3.3 Training Implementation

We use the same task-based network described in chapter 7 for solving detection and segmentation tasks, which uses Mask-RCNN with a ResNet-50 backbone, where features are extracted after the fourth convolutional block. Ensuring a fair comparison across previous experiments, we use the same hyperparameters across all studies, which uses images of size 640x480, a batch size of 4, and an RTX titan GPU. The models were trained using the SGD optimizer, with a base learning rate of 0.00025, a linear warmup factor of 0.001, weight decay of 0.0001, and momentum 0.9. All networks backbones were pre-trained using MSCOCO weights [116] and fined-tuned on the patch-based DR datasets for 25k iterations.

8.4 Experiments

We set out to compare our approach against existing DR texture methods in object detection and segmentation tasks. We compare our method against two baselines: the most commonly used DR texture method, which is flat RGB, and the highest performing DR texture method, zig zag. The baseline performance was evaluated through experimentation in the previous chapter 6. The results from this set of experiments are in Tables 7.25 and 7.26.

We run several experiments to investigate the effectiveness of our proposed method using patch-based textures from images. We do so by evaluating our trained object detection, and segmentation models on a test set of 837 images containing 20 distinct object categories from the YCB-M dataset [67]. The test set contains five scenes that are not visible during the training process and includes various objects differing in shape, size, and visual appearance. Evaluation of the object detection and segmentation tasks use the AP metrics outlined in Chapter 7.

To ensure fairness across all experiments, we fix all rendering parameters, including background, poses, illumination, object and camera positions, and training hyperparameters. The difference between each trained model is the method we apply the texture to the objects of interest. For each subsection, we outline the experimental setup, followed by the results and discussion.

8.4.1 Unconditional Real-World Image Patches

We previously hypothesized that texture complexity - desirable when using DR as seen in chapters 5 and 7 - is more naturally occurring in real images than in artificially created textures. To evaluate our approach in using real-world image patches as textures in DR, we use patch-based textures from two different datasets as described in section 8.3.2.

We apply patches of size 128x128 generated from Textures A and B unconditionally, meaning the textures applied to an object do not rely on a particular category. In the case of Texture

B, the patches were first generated using a crop of size 512x512, then further resized to 128x128 to ensure consistency in spatial dimensions. The reasoning is to ensure visually relevant features are visible in the image patches as shown in Figure 8.4. Using the textures this way is typical of the DR approach, where the applied textures are randomized with no enforcing class-specific information. For example, patches from boxes can be placed on cans. Tables 8.1 and 8.2 show the results for the object detection and segmentation tasks, where we are interested in analyzing texture complexity without additional class-specific information. The DR Baselines for Flat RGB and Zig Zag are from previous experimentation and results shown in Chapter 7 in Tables 7.25 and 7.26.

Dataset	AP	AP50	AP75	APs	APm	APl
DR Baseline: Flat RGB	13.157	27.444	11.358	0.0	14.583	8.797
DR Baseline: Zig Zag	16.354	34.339	13.937	**2.437**	17.309	13.453
Unconditional Texture A	17.33	36.477	14.418	0.766	16.075	18.164
Unconditional Texture B	**19.196**	**41.578**	**15.184**	0.416	**17.645**	**20.681**
Synthetic Real Texture	29.813	58.452	26.942	0.982	25.963	34.024
Real World	39.403	72.902	39.072	6.475	36.93	39.386

Table 8.1: Results from an object detection task using unconditional patch-based textures generated from real-world images.

We see a significant increase in performance over the most commonly applied method using Flat RGB textures in both tasks. In the detection task, we see a boost from 13.157 AP using Flat RGB to 17.330 AP using Texture A and 19.196 AP using Texture B. The boost in performance is also observed, although by a smaller amount when comparing real-world image patches over the highest performing patterned texture. In this case, we see an increase from 16.354 AP to 17.330 AP using Texture A and 19.196 AP using Texture B.

We observe a similar set of results when solving the segmentation task, where we increase the performance of the most commonly applied method from 8.950 AP to 14.370 AP using Texture

Dataset	AP	AP50	AP75	APs	APm	APl
DR Baseline: Flat RGB	8.95	18.159	7.954	0.0	12.162	5.19
DR Baseline: Zig Zag	12.771	25.544	11.877	**0.258**	**13.817**	15.455
Unconditional Texture A	14.37	27.324	13.391	0.033	12.444	22.388
Unconditional Texture B	**17.074**	**32.376**	**15.671**	0.03	13.506	**24.762**
Synthetic Real Texture	26.532	46.073	25.5	0.26	21.237	38.858
Real World	37.198	63.169	36.717	0.13	31.452	45.976

Table 8.2: Results from an object segmentation task using unconditional patch-based textures generated from real-world images.

A and 17.074 AP using Texture B. Similar to the detection results, the performance gain is smaller when comparing the highest performing patterned texture, where we see an increase from 12.771 AP to 14.370 AP using Texture A and 17.074 AP when using Texture B.

One possible explanation for the difference in performance between Texture A and B is the image sharpness of the patches generated from the real-world images. Texture A consists of real-world objects in natural scenes and patches within 2D bounding boxes that specify the objects' location. Unlike Texture A, Texture B contains high-resolution images with the objects mainly in the frame, resulting in sharper patches than those generated from Texture A.

In the approach of randomly generating textures via patches from real-world images unconditionally, we see that outperforms the most commonly used flat RGB textures and highest performing zig zag textures for object detection and segmentation tasks. The generated patches are a quick and easy way to integrate into existing systems. This approach would not require devising complex artificial texture distributions to sample DR textures as is currently the case with commonly used DR techniques, making it an appealing alternative.

8.4.2 Conditional Real-World Image Patches

We have previously seen that using patches from real-world images as the textures when performing DR increases task-based performance, outperforming current implementations. This experiment examines the addition of class-specific information as part of the randomization process. For example, we would only be using patches generated from real-world images of boxes on synthetic boxes or using patches generated from real-world images of cans on synthetic cans. To investigate using class-specific information, we use textures generated from the Texture B dataset. We do so as it contains similar object categories that we may sample our patch-based textures. While there is no direct mapping from each class in Texture B to the target real-world data from the YCB-M dataset, the categories have a broad overlap between them. For example, the Cheez-it box, sugar box, pudding box, and gelatin box would sample patch-based textures from the "box" category. Figure 8.4 shows the different available classes available in the Texture B dataset.

Tables 8.3 and 8.4 show the results when applying the image patches generated from Texture B conditionally, which injects additional visually relevant features into the training set. The conditional approach outperforms the existing DR methods by a greater margin than the unconditional application in detection and segmentation tasks. When comparing Texture set B for object detection and segmentation with DR textures applied conditionally and unconditionally, we see a performance increase from 19.196 AP to 21.287 AP for the detection task and 17.074 AP to 19.481 AP for the segmentation task. This result suggests that additional class-specific information when performing DR is beneficial.

The benefits of class-specific information are more evident when viewing AP scores for each object category as seen in Tables 8.5 and 8.6 for detection and segmentation tasks. It is particularly the case with objects categorized as boxes, such as the Cheez-it, sugar, pudding, and gelatin boxes. The texture patches conditionally applied to these objects visually share similar features such as the text, colors, and patterns. It is also the case with the wooden block. The patches from images applied to the synthetic wooden block contain more visually relevant features,

Dataset	AP	AP50	AP75	APs	APm	APl
DR Baseline: Flat RGB	13.157	27.444	11.358	0.0	14.583	8.797
DR Baseline: Zig Zag	16.354	34.339	13.937	**2.437**	17.309	13.453
Unconditional Texture A	17.33	36.477	14.418	0.766	16.075	18.164
Unconditional Texture B	19.196	41.578	15.184	0.416	17.645	20.681
Conditional Texture B	**21.287**	**42.42**	**19.112**	1.116	**19.794**	**22.397**
Synthetic Real Texture	29.813	58.452	26.942	0.982	25.963	34.024
Real World	39.403	72.902	39.072	6.475	36.93	39.386

Table 8.3: Results from an object detection task using conditional patch-based textures generated from real-world images from Texture B.

Dataset	AP	AP50	AP75	APs	APm	APl
DR Baseline: Flat RGB	8.95	18.159	7.954	0.0	12.162	5.19
DR Baseline: Zig Zag	12.771	25.544	11.877	**0.258**	13.817	15.455
Unconditional Texture A	14.37	27.324	13.391	0.033	12.444	22.388
Unconditional Texture B	17.074	32.376	15.671	0.03	13.506	24.762
Conditional Texture B	**19.481**	**35.868**	**18.578**	0.056	**15.586**	**27.246**
Synthetic Real Texture	26.532	46.073	25.5	0.26	21.237	38.858
Real World	37.198	63.169	36.717	0.13	31.452	45.976

Table 8.4: Results from an object segmentation task using conditional patch-based textures generated from real-world images from Texture B.

despite not appearing photorealistic or matching the target dataset, which outperforms the baseline DR methods using Flat RGB and Zig Zag by over 40 AP points in both detection and segmentation tasks.

Generally, conditionally applying Texture B outperforms using Texture B unconditionally in 14 out of the 20 available categories when evaluating real-world images. We show that combining increased texture complexity and class-specific information during the DR process significantly outperforms the existing DR baselines.

8.4.3 Size of Image Patches

As we have established the usability of generating textures from real-world image patches, we would like to analyze the influence on the image crops' size. The following experiments apply patches from Texture B with varying the initial dimensions of the crops. Figure 8.7 shows an example of the varying sizes of crops used per object category. Here, each image is cropped to the specified resolution and further resized to 128x128 to ensure we maintain the same dimension across experiments. We see that for larger objects, the final generated patches on larger sizes at 1024x1024 and 2048x2048 contain more visual information. However, smaller, thinner objects such as the banana or board marker have more information regarding the background due to the shape of the objects.

Tables 8.7 and 8.8 show the results for varying the size of the patches generated, and evaluated on real-images from the YCB-M dataset [67]. The performance increase between the lowest-performing (256x256) and highest-performing (1024x1024) models for the detection task is an AP score from 20.056 AP to 21.304 AP. In comparison, the performance increase between the lowest-performing (256x256) and highest-performing (512x512) models for the segmentation task is an AP score from 18.112 AP to 19.481 AP. The highest performing models differ between the two tasks, and the difference in performance between the four various-sized crops does not deviate

Dataset	cheezit	sugar	bowl	mug	foam	soup	mustard	tuna	pudding	gelatin	meat	banana	pitcher	bleach	drill	wood	scissors	largeMarker	largeClamp	xLargeClamp	AP
DR Baseline: Flat RGB	0	0.217	6.047	**37.963**	**33.116**	22.599	13.551	2.031	0	0.097	0.734	5.735	**56.616**	29.694	0.019	0.009	**47.651**	0.028	5.445	1.587	13.157
DR Baseline: Zig Zag	28.176	0.559	10.218	37.876	4.066	**39.957**	25.034	1.608	**19.527**	3.102	1.798	3.322	42.218	39.132	1.145	1.864	41.872	18.754	4.119	2.741	16.354
Unconditional Texture A	13.756	2.549	6.806	37.342	12.141	17.637	30.464	25.438	3.493	2.011	3.177	**6.456**	44.241	**51.139**	**5.108**	36.485	22.564	10.541	**9.19**	6.064	17.33
Unconditional Texture B	43.867	1.4	10.061	31.611	0.759	38.525	35.215	18.6	10.11	4.648	**6.801**	2.684	31.865	44.792	2.549	33.034	36.178	17.769	5.613	**7.849**	19.196
Conditional Texture B	**48.69**	**8.406**	**11.854**	30.349	1.129	35.128	**35.898**	**24.457**	16.993	**10.041**	4.828	2.803	39.003	44.942	1.794	**46.595**	39.169	**18.774**	0.931	3.963	**21.287**
Synthetic Real Texture	64.535	41.84	14.243	34.976	26.077	44.498	42.693	12.366	14.528	4.232	15.472	9.756	63.747	52.515	11.293	69.777	37.268	18.811	11.119	6.514	29.813
Real World	67.552	58.59	18.558	44.839	64.739	36.752	36.301	38.814	46.323	48.938	42.623	2.294	67.065	62.631	18.525	72.093	21.789	9.766	2.396	27.481	39.403

Table 8.5: Results from an object detection task across all object classes comparing baseline DR methods, Unconditional, and Conditional application of textures using our method.

Dataset	cheezit	sugar	bowl	mug	foam	soup	mustard	tuna	pudding	gelatin	meat	banana	pitcher	bleach	drill	wood	scissors	largeMarker	largeClamp	xLargeClamp	AP
DR Baseline: Flat RGB	0	0.106	7.352	55.512	**33.485**	22.792	15.417	1.071	0	0.073	0.315	0.004	24.438	15.583	0	0.009	**2.368**	0.002	0.057	0.412	8.95
DR Baseline: Zig Zag	32.667	0.235	13.723	54.748	4.076	35.098	24.682	1.122	15.651	2.609	1.495	0.021	16.177	35.762	0	1.921	1.578	11.933	0.06	1.872	12.771
Unconditional Texture A	12.861	1.902	9.124	**57.852**	12.314	15.399	32.465	17.92	2.553	2.459	3.334	**0.058**	13.955	**54.566**	**0.074**	38.892	0.459	7.106	**0.176**	3.935	14.37
Unconditional Texture B	45.189	0.799	**14.312**	55.717	0.833	**36.318**	**37.294**	15.296	8.547	6.076	**5.723**	0.012	18.51	45.945	0.048	32.285	0.563	12.557	0.117	**5.342**	17.074
Conditional Texture B	**53.949**	**7.395**	13.606	54.818	1.219	30.56	36.552	**18.033**	**17.349**	**9.908**	4.051	0.019	**28.282**	51.56	0.048	**44.891**	1.664	**13.346**	0.018	2.354	**19.481**
Synthetic Real Texture	68.528	40.695	17.536	56.665	25.258	36.69	43.233	8.152	9.218	4.206	16.247	0.034	42.156	72.895	0.033	71.154	1.294	11.447	0.141	5.068	26.532
Real World	72.559	57.089	23.322	55.958	67.798	40.244	43.658	36.674	46.313	47.315	40.707	0.05	45.242	74.432	0.635	70.856	0.035	3.934	0.038	17.104	37.198

Table 8.6: Results from an object segmentation task across all object classes comparing baseline DR methods, Unconditional, and Conditional application of textures using our method.

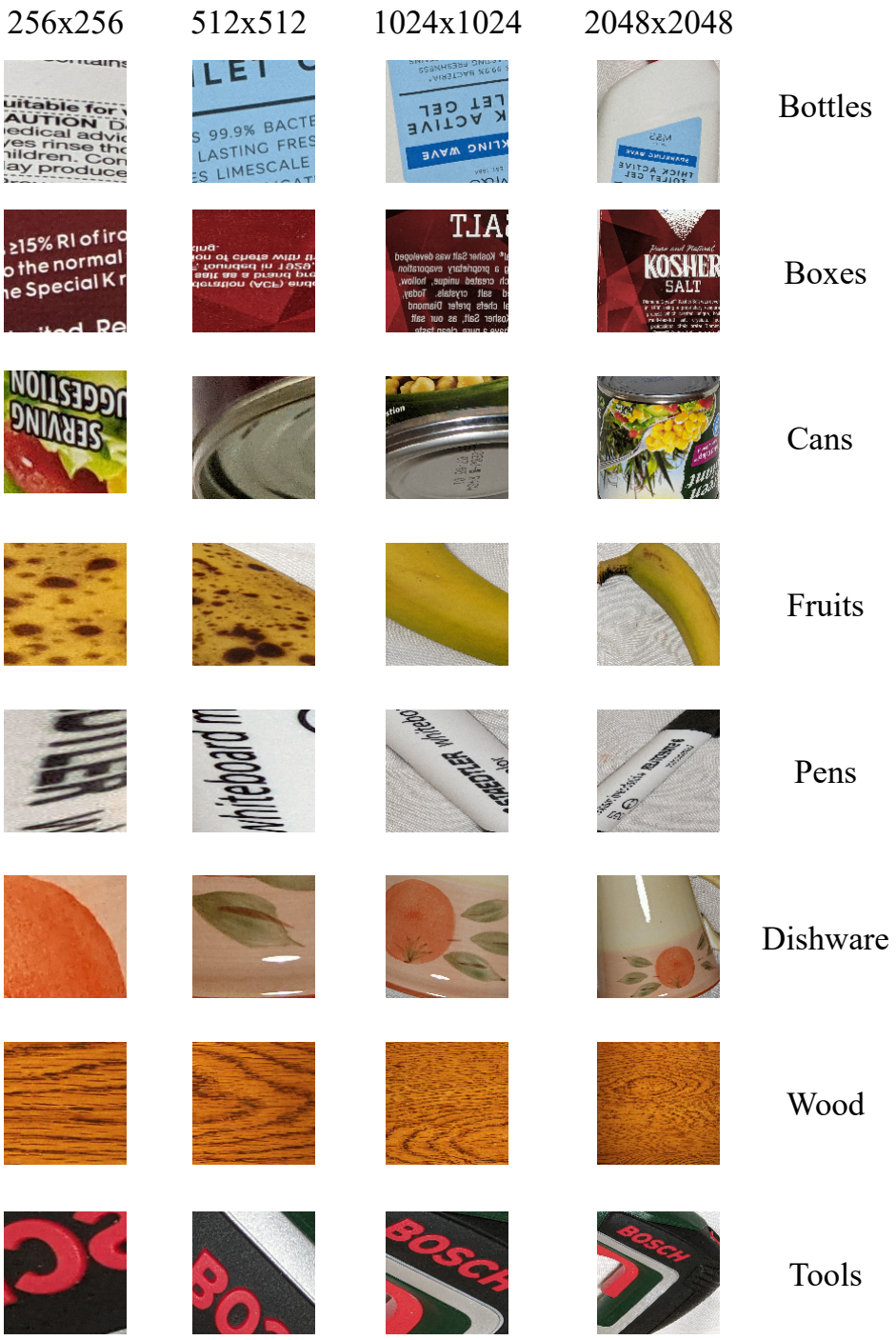


Figure 8.7: Samples of patch-based textures generated from Texture B. Each of the initial crops of varying sizes is resized to 128x128, to ensure the same spatial dimension is preserved across all experiments. Larger objects contain more visual information in larger crop sizes. However, smaller and thinner objects such as the board marker contains more of the background.

drastically. Based on these results, there does not appear to be conclusive evidence for strongly favoring a set of cropped image sizes using Texture set B.

Dataset	AP	AP50	AP75	APs	APm	APl
Conditional Texture B 256x256	20.056	40.385	17.193	0.851	18.489	21.917
Conditional Texture B 2048x2048	20.282	41.617	15.203	0.703	18.422	22.77
Conditional Texture B 512x512	21.287	**42.42**	**19.112**	**1.116**	**19.794**	22.397
Conditional Texture B 1024x1024	**21.304**	42.136	18.355	0.496	18.73	**23.729**
Synthetic Real Texture	29.813	58.452	26.942	0.982	25.963	34.024
Real World	39.403	72.902	39.072	6.475	36.93	39.386

Table 8.7: Results for the object detection task when varying the patch size.

Dataset	AP	AP50	AP75	APs	APm	APl
Conditional Texture B 256x256	18.112	32.951	17.569	0.035	14.319	25.961
Conditional Texture B 1024x1024	18.594	34.123	18.027	**0.103**	13.052	31.017
Conditional Texture B 2048x2048	19.398	34.204	**18.698**	**0.073**	14.292	**31.723**
Conditional Texture B 512x512	**19.481**	**35.868**	18.578	0.056	**15.586**	27.246
Synthetic Real Texture	26.532	46.073	25.5	0.26	21.237	38.858
Real World	37.198	63.169	36.717	0.13	31.452	45.976

Table 8.8: Results for the object segmentation task when varying the patch size.

8.4.4 GAN-Based Image Patches

We explore the possibility of generating patch-based textures using generative models, allowing the synthesis of novel textures to increase texture dataset diversity when using a limited amount of real-world data. The following subsections outline the experimental setup and results when using a conditional and unconditional GAN model.

Dataset	cheezit	sugar	bowl	mug	foam	soup	mustard	tuna	pudding	gelatin	meat	banana	pitcher	bleach	drill	wood	scissors	largeMarker	largeClamp	xLargeClamp	AP
Conditional Texture B 256x256	38.435	6.844	7.827	**40.794**	**6.955**	28.842	**40.782**	10.334	6.618	4.514	**11.973**	2.713	43.806	**45.384**	4.526	49.468	27.784	15.45	2.227	**5.835**	20.056
Conditional Texture B 2048x2048	44.291	**13.318**	11.196	40.173	4.862	28.117	37.417	17.702	5.91	**10.67**	6.655	4.359	29.926	42.806	1.922	**51.203**	30.258	**19.37**	**4.188**	1.301	20.282
Conditional Texture B 512x512	48.69	8.406	**11.854**	30.349	1.129	**35.128**	35.898	**24.457**	**16.993**	10.041	4.828	2.803	39.003	44.942	1.794	46.595	39.169	18.774	0.931	3.963	21.287
Conditional Texture B 1024x1024	**52.283**	11.963	8.58	40.551	2.685	27.269	38.584	13.047	8.813	3.741	7.678	**5.455**	**52.541**	37.09	**5.28**	46.257	**40.092**	18.553	2.251	3.367	**21.304**
Synthetic Real Texture	64.535	41.84	14.243	34.976	26.077	44.498	42.693	12.366	14.528	4.232	15.472	9.756	63.747	52.515	11.293	69.777	37.268	18.811	11.119	6.514	29.813
Real World	67.552	58.59	18.558	44.839	64.739	36.752	36.301	38.814	46.323	48.938	42.623	2.294	67.065	62.631	18.525	72.093	21.789	9.766	2.396	27.481	39.403

Table 8.9: Results for the object detection task when varying the patch size across all object classes.

Dataset	cheezit	sugar	bowl	mug	foam	soup	mustard	tuna	pudding	gelatin	meat	banana	pitcher	bleach	drill	wood	scissors	largeMarker	largeClamp	xLargeClamp	AP
Conditional Texture B 256x256	47.099	4.574	11.09	**54.997**	**7.823**	30.25	**38.379**	6.483	5.813	4.271	**9.004**	0.014	22.84	49.823	0.021	**54.069**	0.965	9.502	0.134	**5.098**	18.112
Conditional Texture B 2048x2048	51.972	9.279	12.751	54.788	2.791	26.879	36.692	9.903	8.246	4.155	7.911	**0.028**	27.095	47.596	0.042	53.83	1.76	13.304	0.138	2.724	18.594
Conditional Texture B 1024x1024	**58.111**	**13.998**	**18.309**	53.27	5.114	28.489	37.167	14.428	5.451	**11.379**	7.511	0.019	22.566	42.796	0.022	52.915	1.373	**13.515**	**0.169**	1.363	19.398
Conditional Texture B 512x512	53.949	7.395	13.606	54.818	1.219	**30.56**	36.552	**18.033**	**17.349**	9.908	4.051	0.019	**28.282**	**51.56**	**0.048**	44.891	**1.664**	13.346	0.018	2.354	**19.481**
Synthetic Real Texture	68.528	40.695	17.536	56.665	25.258	36.69	43.233	8.152	9.218	4.206	16.247	0.034	42.156	72.895	0.033	71.154	1.294	11.447	0.141	5.068	26.532
Real World	72.559	57.089	23.322	55.958	67.798	40.244	43.658	36.674	46.313	47.315	40.707	0.05	45.242	74.432	0.635	70.856	0.035	3.934	0.038	17.104	37.198

Table 8.10: Results for the object segmentation task when varying the patch size across all object classes.

Unconditional GAN

We use the GAN model described in Section 8.3.2 using Texture A and Texture B in an unconditional regime. A total of 10000 patches of size 128x128 are used for each dataset. For Texture B, the patches are initially at size 512x512 to include visually interesting features, then further resized to 128x128 to maintain spatial resolution. Figure 8.8 shows samples using Texture A, and Figure 8.9 shows generated samples from Texture B.

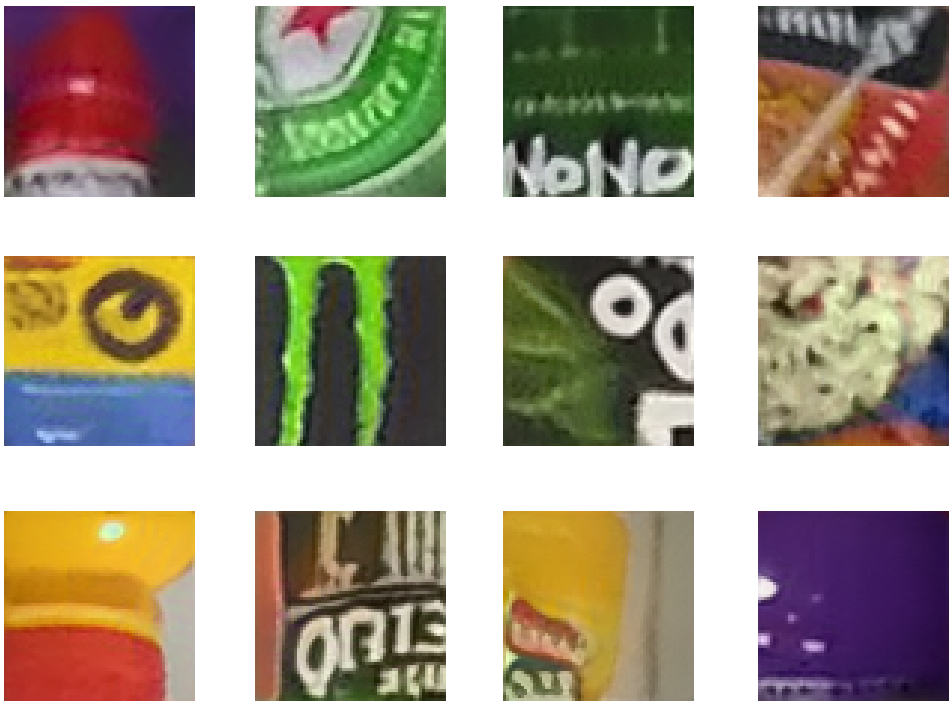


Figure 8.8: Samples generated from the unconditional GAN model from Texture A.

Tables 8.11 and 8.12 show the results for the detection and segmentation tasks, where we see both unconditional GAN models outperform the baseline DR methods of using Flat RGB and the highest performing Zig Zag complex pattern.

The performance between real-world image patches from Texture A and the textures generated from the unconditional model achieves comparable performance, with a slight increase in performance when using the GAN-based approach in detection (17.330 AP to 17.345 AP) and segmentation (14.370 AP to 14.987 AP) tasks. Similarly, when using Texture B, comparable per-

Figure 8.9: Samples generated from the unconditional GAN model from Texture B.

Dataset	AP	AP50	AP75	APs	APm	APl
DR Baseline: Flat RGB	13.157	27.444	11.358	0	14.583	8.797
DR Baseline: Zig Zag	16.354	34.339	13.937	**2.437**	17.309	13.453
Unconditional Texture A	17.33	36.477	14.418	0.766	16.075	18.164
Unconditional GAN Texture A	17.345	35.88	13.827	0.322	**18.504**	16.897
Unconditional GAN Texture B	18.708	38.841	**16.278**	0.511	17.289	18.617
Unconditional Texture B	**19.196**	**41.578**	15.184	0.416	17.645	**20.681**
Synthetic Real Texture	29.813	58.452	26.942	0.982	25.963	34.024
Real World	39.403	72.902	39.072	6.475	36.93	39.386

Table 8.11: Results from the object detection task using the unconditional GAN-based texture application.

Dataset	AP	AP50	AP75	APs	APm	APl
DR Baseline: Flat RGB	8.95	18.159	7.954	0	12.162	5.19
DR Baseline: Zig Zag	12.771	25.544	11.877	**0.258**	13.817	15.455
Unconditional Texture A	14.37	27.324	13.391	0.033	12.444	22.388
Unconditional GAN Texture A	14.987	28.542	14.068	0.056	**14.656**	21.724
Unconditional GAN Texture B	16.151	29.457	**15.847**	0.031	11.936	**25.727**
Unconditional Texture B	**17.074**	**32.376**	15.671	0.03	13.506	24.762
Synthetic Real Texture	26.532	46.073	25.5	0.26	21.237	38.858
Real World	37.198	63.169	36.717	0.13	31.452	45.976

Table 8.12: Results from the object segmentation task using the unconditional GAN-based texture application.

formance is achieved with the detection task, where using the real-world image patches achieves an AP score of 19.196 AP compared to the GAN-based approach at 18.708 AP. The performance gap is wider for the segmentation task, with the real-world image patches model achieves an AP score of 17.074 AP and the GAN-based approach of 16.151 AP. A possible explanation is that the GAN model trained with Texture B did not achieve a similar quality in sample generation to those trained with Texture A, as seen in samples in Figures 8.8 and 8.9. However, in both unconditional models, the number of iterations could be further increased to reduce the FID score and improve the quality of the generated samples.

It is also worth noting that Texture A contains fewer images compared to Texture B, at 166 initial real-world images compared to 274 for Texture B. This difference in initial dataset size may indicate that the generative model approach may be suitable in scenarios where the amount of real-world data is limited, as it would be able to generate more unique textures compared to the starting number of images.

This approach to synthesizing textures is an attractive alternative to manually defining com-

plex data distributions to sample DR textures. The GAN model can be used as a texture generator to create an arbitrarily large, unique, and diverse texture dataset encompassing texture complexity beneficial in DR.

Conditional GAN

We use the GAN model described in Section 8.3.2 using Texture B in a conditional regime. A total of 10000 patches of size 128x128 are used for each class. Samples from the different classes are in Figure 8.7. For Texture B, the patches are initially at size 512x512 to include visually interesting features, then further resized to 128x128 to maintain spatial resolution. Figures from patches generated using the conditional GAN model are in Figure 8.10.

Tables 8.13 and 8.14 show the results when evaluating the dataset using textures generated from the conditional GAN model for detection and segmentation tasks. Using the conditional GAN-based approach to generating textures, we outperform the baseline DR methods, in addition to the unconditional real-world application and unconditional GAN-based approach. This result is the case for detection and segmentation tasks, reinforcing the idea that additional class-specific information is beneficial during the DR process. Although, it is worth noting that the performance increase between unconditional Texture B and the Conditional GAN with Texture B is smaller, going from 19.196 AP to 20.287 AP, and from 17.074 AP to 17.636 AP for the detection and segmentation tasks, respectively. One possible explanation is that the quality of the generated samples is not using Conditional GAN with Texture B is not high enough. Further training to improve the quality of the generated samples may increase the performance difference.

Comparing the conditional GAN-based approach using Texture B against the unconditional GAN-based method using Texture B, we see a performance gain going from an AP score of 18.708 AP to 20.287 AP for the detection task and from 16.151 AP to 17.636 AP for the segmentation task. Similar to the previous set of results using the unconditional GAN-based approach, the model may be improved by training for longer to reduce the FID score and improve the quality of

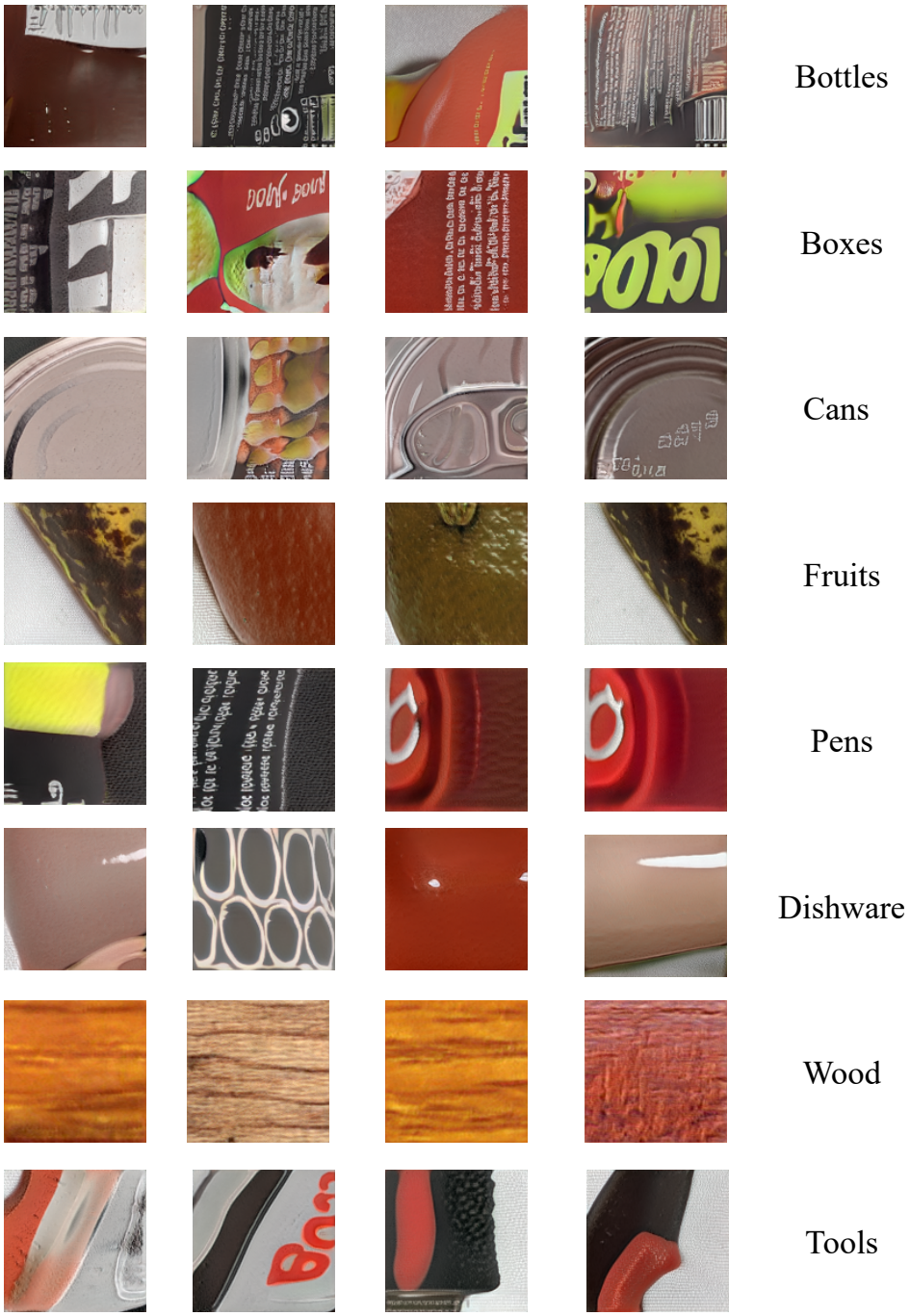


Figure 8.10: Samples generated from the conditional GAN model.

Dataset	AP	AP50	AP75	APs	APm	APl
DR Baseline: Flat RGB	13.157	27.444	11.358	0	14.583	8.797
DR Baseline: Zig Zag	16.354	34.339	13.937	**2.437**	17.309	13.453
Unconditional Texture A	17.33	36.477	14.418	0.766	16.075	18.164
Unconditional GAN Texture A	17.345	35.88	13.827	0.322	**18.504**	16.897
Unconditional GAN Texture B	18.708	38.841	**16.278**	0.511	17.289	18.617
Unconditional Texture B	19.196	41.578	15.184	0.416	17.645	**20.681**
Conditional GAN Texture B	**20.287**	**41.715**	16.112	2.244	17.733	19.467
Synthetic Real Texture	29.813	58.452	26.942	0.982	25.963	34.024
Real World	39.403	72.902	39.072	6.475	36.93	39.386

Table 8.13: Results from the object detection task using the conditional GAN-based texture application.

Dataset	AP	AP50	AP75	APs	APm	APl
DR Baseline: Flat RGB	8.95	18.159	7.954	0	12.162	5.19
DR Baseline: Zig Zag	12.771	25.544	11.877	**0.258**	13.817	15.455
Unconditional Texture A	14.37	27.324	13.391	0.033	12.444	22.388
Unconditional GAN Texture A	14.987	28.542	14.068	0.056	**14.656**	21.724
Unconditional GAN Texture B	16.151	29.457	15.847	0.031	11.936	25.727
Unconditional Texture B	17.074	**32.376**	15.671	0.03	13.506	24.762
Conditional GAN Texture B	**17.636**	32.26	**17.613**	0.1	12.505	**26.166**
Synthetic Real Texture	26.532	46.073	25.5	0.26	21.237	38.858
Real World	37.198	63.169	36.717	0.13	31.452	45.976

Table 8.14: Results from the object segmentation task using the conditional GAN-based texture application.

the generated samples.

Overall, there are trade-offs when using the GAN-based approach for generating textures. While the performance is comparable to using real-world image patches, we may also use it to increase dataset diversity using a limited amount of real-world images. However, we must factor in the additional cost of training a separate network to generate patch-based textures. Depending on the resolution of images, the number of images, and the number of GPUs, training time may exceed several days. However, the result is a texture generator that is capable of producing more diverse textures.

8.5 Conclusion

This chapter presented a novel approach for generating textures for DR using textures generated from real-world image patches for solving object detection and segmentation tasks. Our key findings show that the beneficial texture complexity can easily be attained using natural images rather than artificially creating complexity using synthetic data. We show that we can further increase performance over existing DR implementations by conditionally applying the patch-based textures to objects of interest, demonstrating that additional class-specific information is beneficial during the DR process. Functionally, the method is a fast, simple, and high-performing solution to creating complex DR textures compared to the usual approach in the current literature. Furthermore, we show that we can create a conditional generative model capable of producing comparable textures to real-world image patches with a small amount of real-world data. We envisage that the conditional GAN could be used to increase dataset diversity when access to real-world data is limited, allowing increased diversity in textures capable of being generated on the fly during a training process.

Chapter 9

Conclusions and Future Work

The thesis is motivated by the appeal of learning purely from synthetic data when applied to real-world scenarios. The benefits of synthesizing large amounts of high-quality annotated data at a fraction of the time required for real-world counterparts are sometimes diminished when deployed in the real world due to the domain shift between synthetic and natural data. This thesis focused on overcoming several shortcomings when bridging the gap between synthetic and real-world data domains using DR. Such shortcomings include a trial-and-error approach to applying DR for vision and robotics tasks. Here, a practitioner must decide which parameters to randomize to solve some arbitrary task and perform task-based training and evaluation to know if the generated DR data is suitable. It is also unclear if the design choices apply equally across multiple tasks. For example, would similar textures in a dataset for solving object localization be suitable for solving object segmentation? Finally, DR often requires defining a distribution to sample parameters to generate data. For example, should we use a uniform data distribution to sample values for our colored textures, or are there alternatives that remove that portion of the decision-making and ease the execution?

We address the challenges by working towards a more robust form of DR. Firstly, we provide methods for measuring differences between DR and realistic data distributions to eliminate

task-based training and evaluation. The technique can rank texture randomization methods used in the existing literature and predict the performance of an object localization task ranking with good agreement without task-based training and evaluation.

To further probe research questions in sim-to-real DR settings, we generated the SRDR dataset, a large domain randomized dataset containing 291K frames of realistic household objects widely used in robotics and vision benchmarking. The SRDR dataset builds on the YCB-M dataset [67] by generating DR synthetic versions of each YCB-M image (from a single camera's viewpoint) using a variety of DR texture types used in the existing literature and five different backgrounds environments. Using the SRDR dataset in conjunction with the real-world images from the YCB-M dataset [67] is highly beneficial in cross-domain training, evaluation, and comparison investigations due to the access of matched synthetic-real scenes, a variety of texture randomization methods used within DR literature, and diverse environments. Furthermore, we provide software tools built upon the work by To et al. [200] to enable their program to generate repeatable DR scenes using scene description files. This tool would allow researchers to create their own DR synthetic data provided access to 3D models and annotations describing the scene.

To address the challenge of design choices when using DR across various vision tasks, we present a comprehensive study evaluating DR's generalizability and robustness in sim-to-real settings by randomizing poses, textures, and backgrounds in cluttered and occluded scenes. We use the SRDR dataset from chapter 6, and the YCB-M dataset [67] to facilitate the cross-domain sim-to-real DR study for solving object detection and segmentation tasks. We find that the selection of DR textures, backgrounds, and object poses does not drastically change performance ranking when evaluating models trained on DR synthetic data and evaluating on real-world data across object detection and segmentation tasks. These findings indicate DR behaves similarly across multiple tasks. The design choices we suggest from this study when generating new DR data is a focus on creating a wide range of object poses that would be similar to the target pose data distribution. The study indicates that using textures containing complex patterns such as a checkerboard or zig zag yields higher task-based performance, which reflects our texture selection rankings from previous

work in chapter 5. Finally, using a robust set of non-repeatable image backgrounds from random real-world images such as the Active-Vision dataset [4], or a photorealistic synthetic dataset such as Structured3D [244].

A final challenge the thesis tackles is regarding the sampling techniques for DR data. Specifically, the design choices regarding the distribution to sample textures from and the selection of the types of complex textures previously shown as desirable in chapters 5 and 7. The final work presents a novel method for conditionally generating and applying DR textures using patches from real-world images, outperforming the most commonly applied DR texture and the highest performing DR texture, as determined in our previous findings in chapter 7. We outperform the most widely used DR texture randomization method going from 13.157 AP to 21.287 AP and 8.950 AP to 19.481 AP in object detection and semantic segmentation tasks, respectively. We also outperform the highest performing DR texture going from 16.354 AP to 21.287 AP and from 12.771 AP to 19.481 AP in object detection and semantic segmentation tasks, respectively. Using readily available real-world images [134] means the approach is fast, easy to implement, and removes decisions on manually defining texture generation routines to generate textures for generating DR synthetic images. A further improvement is presented to address low texture diversity when using a small number of real-world images to generate patches. We propose a conditional GAN-based texture generator trained on a few real-world image patches to increase texture diversity and outperform the most commonly applied DR texture randomization method going from 13.157 AP to 20.287 AP and 8.950 AP to 17.636 AP in object detection and semantic segmentation tasks. This approach also outperforms the best texture randomization method for object detection and segmentation tasks going from 16.354 AP to 20.287 AP and 12.771 AP to 17.636 AP, respectively.

The next section 9.1 is a summary of the chapters presented in the thesis, followed by a summary of the key contributions in section 9.1.1, a discussion of the limitations and future directions for the research are in section 9.2.

9.1 Summary

In Chapters 2 and 3, we presented an overview of the potential benefits that arise from using synthetic data in a deep learning pipeline. Of importance is the ability to generate large quantities of high-quality annotated data that data-driven supervised methods typically rely on to solve a given task. This feature is especially beneficial in the vision domain due to the reliance on data-driven approaches in the field [151, 193]. One can synthesize data from multiple sensors such as RGB or depth maps, along with pixel-level annotations for solving object localization, detection, and segmentation tasks.

In Chapter 4, we presented an overview for the Domain Randomization method, covering the core concepts, traditional algorithm, and frameworks in use. We looked at the current state-of-the-art approaches for using Domain Randomization, including how it is being in terms of the types of tasks it is being used to solve and the general procedure for doing so. We highlight the essential role that textures have in the Domain Randomization process and find that researchers use a variety of different textures in the randomization process, with Flat RGB being the most commonly used. However, it is unclear which approach would be most suitable for solving a given task. This variation in applying this strategy sets the central theme for the thesis, which is how we may best apply DR techniques across multiple tasks, with an emphasis on the types and generation of textures used in the process.

The following questions arise from differences in the types of tasks and applications of Domain Randomization:

- How does the Domain Randomization process affect the underlying data distributions?
- How does Domain Randomization generalize across different computer vision tasks?
- How can we improve the traditional approach to Domain Randomization?

In Chapter 5 we address the first question of how the Domain Randomization process affects data distributions by formulating a novel framework for quantifying the differences between synthetic and real-equivalent datasets in the feature space. The quantification of the differences in data distributions allowed us to evaluate the most widely used texture Domain Randomization methods and found that we can predict task-based performance without the additional expense of task-based training and evaluation. We find that the most commonly used textures in the Domain Randomization literature achieved the worst task-based performance and demonstrate that more complex patterns typically increase task-based performance.

In Chapter 6 we introduced a large Domain Randomized dataset to enable researchers to probe questions surrounding the use of Domain Randomization in both robotics and vision tasks. The dataset contains all the commonly applied texture Domain Randomization methods and replicates real-world scenes from the YCB-M dataset [67]. The YCB-M dataset is a real-world annotated object-centric dataset for solving vision and robotics tasks. For the SRDR dataset, we use all images and annotations from one of the cameras from the YCB-M dataset and generate DR versions of them using the texture randomization techniques in the current literature. For each DR image, we use backgrounds from five different background environments ranging from synthetic, photorealistic synthetic, and real-world images [4, 74, 132, 244], and use the 3D object models provided by YCB [23] to match the object positions of the YCB-M dataset. Since the SRDR dataset replicates scenes from the YCB-M dataset, researchers can incorporate a mix of synthetic, Domain Randomized synthetic, and real-world data in their investigations. They may select the images from the SRDR dataset with particular textures, backgrounds, or a variety of them, making it useful for cross-domain applications when combined with their real-world counterparts from the YCB-M dataset.

The YCB-M dataset, which we used the annotations from to create matched data, contains 31 scenes of various scene complexity, including clutter, occlusion, and varying object geometries allowing greater flexibility of the types of settings to use. Finally, we provide software tools built upon the work by To et al. [200] to enable their program to generate repeatable DR scenes using

scene description files. This tool would allow researchers to create their own DR synthetic data provided access to 3D models and annotations describing the scene. To show the flexibility of this dataset and plugin, the works conducted in Chapters 3 and 8 use both the SRDR dataset and SRDR plugin for training models in object detection and semantic segmentation.

Chapter 7 addresses the second question of how Domain Randomization behaves across multiple tasks in complex scenes. Unlike some works using primitive shapes when applying Domain Randomization, we investigate the method's usability in complex scenes involving various objects, object geometry, occlusion, and clutter across multiple tasks. We comprehensively evaluate various stages of the Domain Randomization process in complex settings across object detection and semantic segmentation tasks by investigating object poses, image backgrounds, and texture randomization methods. We find that imposing constraints on object poses to viewpoints more similar to the target data distribution increases task-based performance across multiple tasks. Additionally, current literature does not evaluate the wide range of possibilities to randomize object backgrounds. We find that increased realism, in the form of real-world images or photorealistic synthetic images with low clutter, is more beneficial than non-realistic and highly cluttered image backgrounds. Finally, we demonstrate that Domain Randomization is task-agnostic, achieving similar rankings in task-based performance across object localization, detection, and segmentation tasks, where we find that using more complex patterned textures still achieves the highest task-based performance across multiple tasks.

Finally, we address the third question surrounding the improvement of current Domain Randomization methods in Chapter 8. Previously, we have confirmed the importance of texture complexity for improving task-based performance. In this chapter, we present our alternative approach to texture synthesis for use in Domain Randomization. We devise an alternative conditional patch-based method for generating Domain Randomized textures that outperform the existing texture generation methods used in the current literature in complex scenes for object detection and semantic segmentation tasks. This quick, inexpensive, and easy to implement approach shows that we can attain texture complexity from natural images instead of artificially creating texture

complexity to sample Domain Randomized textures. We show that additional conditional, class-specific information in the form of conditionally applying the textures further increases task-based performance. Furthermore, we proposed a conditional GAN-based method that operates on a small amount of real-world data to create patch-based textures from real-world images. This texture synthesis method can increase texture diversity further while including the required texture complexity that Domain Randomization methods favor.

9.1.1 Key Contributions

In summary, this thesis has presented the following:

- We present a novel framework based on the Wasserstein and FID for quantifying data distributions between synthetic and real-equivalent allowed us to evaluate the performance of a localization task when using various Domain Randomization methods currently used in the existing literature, without the additional expense of task-based training and evaluation.
- We presented a large multi-task dataset for Domain Randomization, tailored to probing questions surrounding cross-domain investigations via a combination of synthetic, Domain Randomized synthetic using commonly applied techniques, and real-world images from the YCB-M dataset [67].
- We provide the software plugin to enable out-of-the-box Domain Randomization scene replication via scene configuration files to enable reproducibility and finer control of generating Domain Randomized scenes.
- Current works do not analyze the usability of Domain Randomization across multiple tasks. We perform an exhaustive investigation into the generalizability of Domain Randomization across object localization, detection, and semantic segmentation tasks. We demonstrate that constraints on object poses, texture complexity, and background complexity can increase

task-based performance. We show that Domain Randomization is task-agnostic and generally behaves similarly across multiple tasks.

- We present an alternative approach to synthesizing textures for Domain Randomization, without the need for manually defining texture data distributions to sample textures from. Our system uses patches from real-world images and outperforms existing methods by as much as double from the most commonly used procedures in complex scenes for solving object detection and semantic segmentation tasks. We find that not only can we attain texture complexity from natural images, but we can also further increase performance by conditionally applying Domain Randomization to objects in the scene.
- We devised a conditional generative-based approach using a small number of real-world patches from images for synthesizing textures that would increase data diversity.

9.2 Limitations and Future Work

Chapter 5 proposes a novel method of quantifying differences in data distributions between realistic (real-textured) and Domain Randomized data using commonly applied texture randomization techniques. The approach is currently focused on using the different types of texture randomization methods. However, it would be interesting to explore the impact of several factors such as textures, illumination, camera, or object positions simultaneously. The work also focuses on using a single object in non-complex scenes. An extension would be to scale up the number of objects and tweak the above combination of DR parameters to understand its effects better when measuring the difference between synthetic and realistic data distributions. It would be interesting to note if there is a point where the ranking estimates no longer agree with the performance rankings from solving a given task.

The SRDR dataset in Chapter 6 uses annotations from the YCB-M dataset [67] to generate DR scenes that match the real world. The reliance on the YCB-M annotations limits the scope

of the dataset we may create. It would be helpful to generate matched real-world scenes under different conditions. Such conditions could be controlled, matched, illumination in the real-world and synthetic world. This type of data would enable further experimentation into lighting as part of the randomization process. We could also generate scenes only containing objects of similar classes, such as only boxes or only cans, to better understand how DR behaves when our dataset contains similar shapes or textures.

The study conducted regarding the generalization of DR across multiple tasks in Chapter 7 revealed insights into the types of design choices to improve task-based performance across detection and segmentation tasks. While DR is used to solve such tasks, it would be helpful to know if these approaches can be used to solve other vision tasks that do not typically utilize DR, such as image denoising, deblurring, or object tracking.

We could perform further error analysis to investigate the differences between the synthetic and real-world objects by selecting the cases where the system fails to transfer to the real world and find patterns in these situations. For example, we may find that objects of a specific geometric shape perform higher than those more complex. Furthermore, we may visually investigate failure cases where the synthetic-based approach fails to detect or segment a given object. However, a system trained on real-world images produces the correct result. We may notice clusters or other issues with the failure cases that may further our knowledge on the topic and lead to methods to overcome these cases.

This chapter also finds that the background selection played a substantial role in impacting task-based performance when trained on DR synthetic data and evaluated on real-world data. It would be interesting to investigate this further by re-creating several real-world environments of varying complexities in the synthetic world to explore further the influence of background selection on the problem.

In Chapter 8, we presented an approach for conditionally generating textures from randomly sampled cropped image patches from real-world objects and applied them to 3D object

meshes. This approach improves the existing texture generation routines used within DR literature. Further, we propose a conditional GAN-based approach that could be beneficial when operating in low dataset regimes. A potential extension to this work could be to crop and apply the textures to objects in a more natural way, for example, only selecting entire surfaces of real-world objects without the possibility of including pixels from the real-world background or an edge of an object. It would be interesting to investigate how this would influence task-based performance and further guide DR texture generation methods.

The thesis focused on a central theme of texture Domain Randomization and its effects on the learning process and performance across multiple tasks. In addition to exploring the avenues mentioned above, another natural progression of the work is to incorporate studies into illumination on the randomization process. There have been exciting developments towards scene relighting and estimating illumination in real-world scenes, which can be highly beneficial in synthetic scene generation [48]. Recent trends towards neural network-based approaches to rendering could enable us to optimize positions or intensity of synthetic lighting conditions based on real-world target data. This illumination estimation method, combined with our proposed conditional GAN-based approach, could lead to more suitable Domain Randomized data to help bridge the gap between synthetic and real. We could further extend the idea of adding conditional information to illumination, where synthetic scenes would be created using an approximation of lighting conditions on a small amount of real-world data.

Appendix A

Additional Image Samples For Conditional Domain Randomization

The following Figures A.1 and A.2 include additional samples using the CDR approach for generating patch-based textures from real-world images, and the conditional GAN-based approach described in Chapter 8.

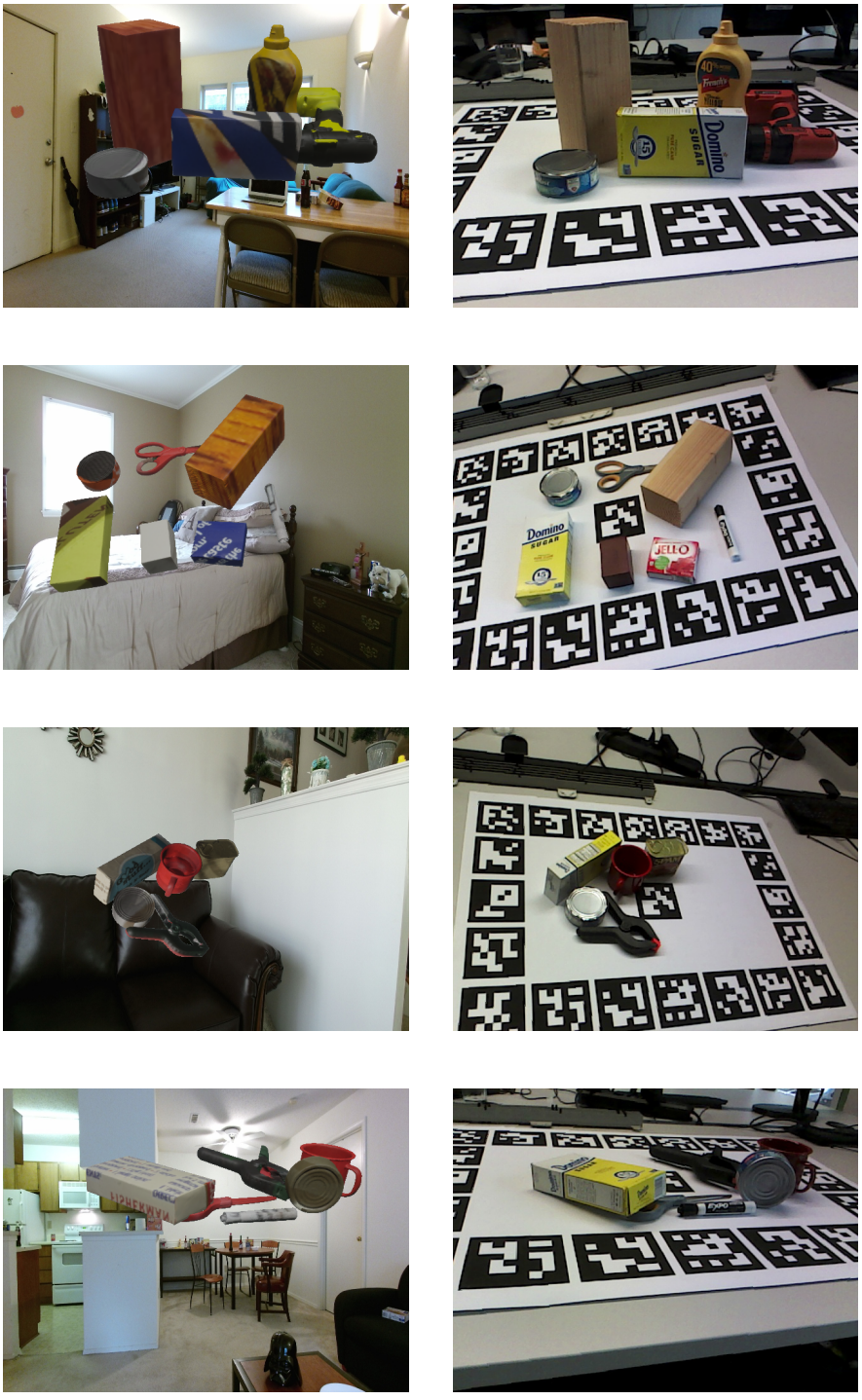


Figure A.1: Additional CDR samples using the proposed method presented in Chapter 8. The left column shows the CDR samples, while the right column shows the real-world images from the YCB-M dataset [67].

Figure A.2: Additional conditional GAN-based samples using the proposed method presented in Chapter 8. The left column shows the conditional GAN-based approach samples, while the right column shows the real-world images from the YCB-M dataset [67].

References

[1] Leon A .Gatys, Alexander S Ecker, and Matthias Bethge. “Texture and art with deep neural networks”. In: *Current Opinion in Neurobiology* 46 (2017). Computational Neuroscience, pp. 178–186.

[2] Aibek Alano. et al. “User-controllable Multi-texture Synthesis with Generative Adversarial Networks”. In: *Proceedings of the 15th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 4: VISAPP,* SciTePress, 2020, pp. 214–221.

[3] Raghad Alghonaim and Edward Johns. “Benchmarking Domain Randomisation for Visual Sim-to-Real Transfer”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2021.

[4] Phil Ammirato et al. “A Dataset for Developing and Benchmarking Active Vision”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2017.

[5] Mohammad Ani, Hector Basevi, and Aleš Leonardis. “Quantifying the Use of Domain Randomization”. In: *2020 25th International Conference on Pattern Recognition (ICPR)*. 2021, pp. 6128–6135.

[6] Martin Arjovsky, Soumith Chintala, and Léon Bottou. “Wasserstein Generative Adversarial Networks”. In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Re-

search. International Convention Centre, Sydney, Australia: PMLR, Aug. 2017, pp. 214–223.

[7] Mathieu Aubry et al. “Seeing 3D Chairs: Exemplar Part-Based 2D-3D Alignment Using a Large Dataset of CAD Models”. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 3762–3769.

[8] Maria-Florina Balcan et al. *How much data is sufficient to learn high-performing algorithms? Generalization guarantees for data-driven algorithm design*. 2021. arXiv: 1908.02894.

[9] Pedro Ballester and Ricardo Matsumura Araujo. “On the Performance of GoogLeNet and AlexNet Applied to Sketches”. In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. AAAI’16. Phoenix, Arizona: AAAI Press, 2016, pp. 1124–1128.

[10] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. “SURF: Speeded Up Robust Features”. In: *Computer Vision – ECCV 2006*. Ed. by Aleš Leonardis, Horst Bischof, and Axel Pinz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 404–417.

[11] Harkirat Singh Behl et al. “AutoSimulate: (Quickly) Learning Synthetic Data Generation”. In: *16th European Conference Computer Vision (ECCV 2020)*. Aug. 2020.

[12] James F. Blinn. “Models of Light Reflection for Computer Synthesized Pictures”. In: *Proceedings of the 4th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’77. San Jose, California: Association for Computing Machinery, 1977, pp. 192–198.

[13] Jeannette Bohg et al. “Data-Driven Grasp Synthesis—A Survey”. In: *IEEE Transactions on Robotics* 30.2 (Apr. 2014), pp. 289–309.

[14] João Borrego et al. “Applying Domain Randomization to Synthetic Data for Object Category Detection”. In: arXiv:1807.09834 (2018). arXiv: 1807.09834.

[15] Konstantinos Bousmalis et al. “Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping”. In: (2017). arXiv: 1709.07857.

[16] Wieland Brendel and Matthias Bethge. “Approximating CNNs with Bag-of-local-Features models works surprisingly well on ImageNet”. In: *International Conference on Learning Representations*. 2019.

[17] Lorenzo Brigato and Luca Iocchi. “A close look at deep learning with small data”. In: *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE. 2021, pp. 2490–2497.

[18] Andrew Brock, Jeff Donahue, and Karen Simonyan. “Large Scale GAN Training for High Fidelity Natural Image Synthesis”. In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. 2019.

[19] Tom Brown et al. “Language Models are Few-Shot Learners”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901.

[20] Joy Buolamwini and Timnit Gebru. “Gender Shades: Intersectional Accuracy Disparities in Commercial Gender Classification”. In: *Proceedings of the 1st Conference on Fairness, Accountability and Transparency*. Ed. by Sorelle A. Friedler and Christo Wilson. Vol. 81. Proceedings of Machine Learning Research. New York, NY, USA: PMLR, Feb. 2018, pp. 77–91.

[21] Daniel J. Butler et al. “A Naturalistic Open Source Movie for Optical Flow Evaluation”. In: *Computer Vision – ECCV 2012*. Ed. by Andrew Fitzgibbon et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 611–625.

[22] Yohann Cabon, Naila Murray, and Martin Humenberger. *Virtual KITTI 2*. 2020. arXiv: 2001.10773.

[23] Berk Calli et al. “Yale-CMU-Berkeley dataset for robotic manipulation research”. In: *International Journal of Robotics Research* 36.3 (2017), pp. 261–268.

[24] Michael Calonder et al. “BRIEF: Binary Robust Independent Elementary Features”. In: vol. 6314. Sept. 2010, pp. 778–792.

[25] Jaquin Quiñonero Candela. *Managing Your Identity on Facebook With Face Recognition Technology*. https://about.fb.com/news/2017/12/managing-your-identity-on-facebook-with-face-recognition-technology/. 2017.

[26] Caogang. *caogang/wgan-gp*. Nov. 2017. URL: https://github.com/caogang/wgan-gp.

[27] CHAI3D. URL: https://www.chai3d.org/download/doc/html/chapter15-material.html.

[28] Supriyo Chakraborty et al. "Interpretability of deep learning models: A survey of results". In: *2017 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computed, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*. 2017, pp. 1–6.

[29] Alan Chalmers and Andrej Ferko. "Levels of Realism: From Virtual Reality to Real Virtuality". In: *Proceedings of the 24th Spring Conference on Computer Graphics*. SCCG '08. Budmerice, Slovakia: Association for Computing Machinery, 2008, pp. 19–25.

[30] Alan Chalmers and Andrej Ferko. "Levels of Realism: From Virtual Reality to Real Virtuality". In: *Proceedings of the 24th Spring Conference on Computer Graphics*. SCCG '08. Budmerice, Slovakia: Association for Computing Machinery, 2008, pp. 19–25.

[31] Zhengping Che et al. *D^2-City: A Large-Scale Dashcam Video Dataset of Diverse Traffic Scenarios*. 2019. arXiv: 1904.01975.

[32] Liang-Chieh Chen et al. "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs". In: *IEEE transactions on pattern analysis and machine intelligence* 40.4 (2017), pp. 834–848.

[33] Qifeng Chen and Vladlen Koltun. "Photographic Image Synthesis with Cascaded Refinement Networks". In: *ICCV*. Oct. 2017, pp. 1520–1529.

[34] Wenting Chen et al. "Texture Deformation Based Generative Adversarial Networks for Face Editing". In: *PRICAI 2019: Trends in Artificial Intelligence* abs/1812.09832 (2019).

[35] Corinna Cortes and V. Vapnik. “Support-Vector Networks”. In: *Machine Learning* 20 (2004), pp. 273–297.

[36] Erwin Coumans and Yunfei Bai. *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. http://pybullet.org. 2020.

[37] Gabriela Csurka. “A Comprehensive Survey on Domain Adaptation for Visual Applications”. In: *Domain Adaptation in Computer Vision Applications*. Ed. by Gabriela Csurka. Cham: Springer International Publishing, 2017, pp. 1–35.

[38] Xiyang Dai et al. “Dynamic Head: Unifying Object Detection Heads with Attentions”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 7373–7382.

[39] Deepdrive. *Deepdrive Simulator*. https://github.com/deepdrive/deepdrive. 2018.

[40] Jia Deng et al. “ImageNet: A large-scale hierarchical image database”. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition* (2009), pp. 248–255.

[41] Jeevan Devaranjan, Amlan Kar, and Sanja Fidler. “Meta-Sim2: Learning to Generate Synthetic Datasets”. In: *ECCV*. 2020.

[42] Mahesh M Dhananjaya, Varun Ravi Kumar, and Senthil Yogamani. *Weather and Light Level Classification for Autonomous Driving: Dataset, Baseline and Active Learning*. 2021. arXiv: 2104.14042.

[43] Jeff Donahue et al. “Decaf: A deep convolutional activation feature for generic visual recognition”. In: *International conference on machine learning*. PMLR. 2014, pp. 647–655.

[44] Alexey Dosovitskiy et al. “CARLA: An Open Urban Driving Simulator”. In: *Proceedings of the 1st Annual Conference on Robot Learning*. 2017, pp. 1–16.

[45] D. Dwibedi, I. Misra, and M. Hebert. “Cut, Paste and Learn: Surprisingly Easy Synthesis for Instance Detection”. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. Los Alamitos, CA, USA: IEEE Computer Society, Oct. 2017, pp. 1310–1319.

[46] A.A. Efros and T.K. Leung. “Texture synthesis by non-parametric sampling”. In: *Proceedings of the Seventh IEEE International Conference on Computer Vision*. Vol. 2. 1999, 1033–1038 vol.2.

[47] Alexei A. Efros and William T. Freeman. “Image Quilting for Texture Synthesis and Transfer”. In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’01. New York, NY, USA: Association for Computing Machinery, 2001, pp. 341–346.

[48] Farshad Einabadi, Jean-Yves Guillemaut, and Adrian Hilton. “Deep Neural Models for Illumination Estimation and Relighting: A Survey”. In: *Computer Graphics Forum* 40.6 (2021), pp. 315–331.

[49] Unreal Game Engine. URL: https://www.unrealengine.com/.

[50] Unreal Game Engine. *Digital Humans: MetaHuman Creator*. URL: https://www.unrealengine.com/en-US/digital-humans.

[51] M. Everingham et al. “The Pascal Visual Object Classes Challenge: A Retrospective”. In: *International Journal of Computer Vision* 111.1 (Jan. 2015), pp. 98–136.

[52] Peter R Florence, Lucas Manuelli, and Russ Tedrake. “Dense Object Nets: Learning Dense Visual Object Descriptors By and For Robotic Manipulation”. In: *CoRL*. 2018. arXiv: 1806.08756v2.

[53] *Free 3D Home Planner: Design a House Online: Planner5D*. URL: https://planner5d.com/.

[54] Adrien Gaidon et al. “Virtual Worlds as Proxy for Multi-Object Tracking Analysis”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 4340–4349.

[55] Yaroslav Ganin and Victor Lempitsky. “Unsupervised Domain Adaptation by Backpropagation”. In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 2015, pp. 1180–1189.

[56] Yaroslav Ganin et al. “Synthesizing Programs for Images using Reinforced Adversarial Learning”. In: *ICML*. 2018.

[57] Andreas Geiger, Philip Lenz, and Raquel Urtasun. “Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012.

[58] Robert Geirhos et al. “ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness.” In: *International Conference on Learning Representations*. 2019.

[59] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. “Dropblock: A regularization method for convolutional networks”. In: *arXiv preprint arXiv:1810.12890* (2018).

[60] Ross Girshick. “Fast R-CNN”. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. Dec. 2015.

[61] Ross Girshick et al. “Rich feature hierarchies for accurate object detection and semantic segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.

[62] Zhiqiang Gong, Ping Zhong, and Weidong Hu. “Diversity in Machine Learning”. In: *IEEE Access* 7 (2019), pp. 64323–64350.

[63] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[64] Ian Goodfellow et al. “Generative Adversarial Nets”. In: *Advances in Neural Information Processing Systems 27*. 2014, pp. 2672–2680.

[65] H. Gouraud. “Continuous Shading of Curved Surfaces”. In: *IEEE Transactions on Computers* C-20.6 (1971), pp. 623–629.

[66] Manik Goyal et al. “Dataset Augmentation with Synthetic Images Improves Semantic Segmentation”. In: *Computer Vision, Pattern Recognition, Image Processing, and Graphics*. Ed. by Renu Rameshan, Chetan Arora, and Sumantra Dutta Roy. Singapore: Springer Singapore, 2018, pp. 348–359.

[67] Till Grenzdörffer, Martin Günther, and J. Hertzberg. “YCB-M: A Multi-Camera RGB-D Dataset for Object Recognition and 6DoF Pose Estimation”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)* (2020), pp. 3650–3656.

[68] Ishaan Gulrajani et al. “Improved Training of Wasserstein GANs”. In: *NIPS*. 2017.

[69] Yanming Guo et al. “Deep learning for visual understanding: A review”. In: *Neurocomputing* 187 (2016). Recent Developments on Deep Big Vision, pp. 27–48.

[70] Kaiming He, Ross Girshick, and Piotr Dollár. “Rethinking imagenet pre-training”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 4918–4927.

[71] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pp. 770–778.

[72] Kaiming He et al. “Mask r-cnn”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2961–2969.

[73] Stefan Hinterstoisser et al. “On Pre-trained Image Features and Synthetic Images for Deep Learning”. In: *ECCV*. Jan. 2019, pp. 682–697.

[74] Tomáš Hodaň et al. “Photorealistic Image Synthesis for Object Instance Detection”. In: *IEEE International Conference on Image Processing (ICIP)* (2019).

[75] Weixiang Hong et al. “Conditional Generative Adversarial Network for Structured Domain Adaptation”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2018, pp. 1335–1344.

[76] Gary B. Huang et al. *Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments*. Tech. rep. 07-49. University of Massachusetts, Amherst, Oct. 2007.

[77] Haroon Idrees et al. “Multi-source Multi-scale Counting in Extremely Dense Crowd Images”. In: *2013 IEEE Conference on Computer Vision and Pattern Recognition*. 2013, pp. 2547–2554.

[78] Image*After. *Wooden Textures*. http://www.imageafter.com/category.php?category=woods. 2019.

[79] Phillip Isola et al. "Image-to-Image Translation with Conditional Adversarial Networks". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), pp. 5967–5976.

[80] Mona Jalal et al. "SIDOD: A Synthetic Image Dataset for 3D Object Pose Recognition With Distractors". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (2019), pp. 475–477.

[81] Stephen James, Andrew J Davison, and Edward Johns. "Transferring End-to-End Visuomotor Control from Simulation to Real World for a Multi-Stage Task". In: *CoRL* (2017).

[82] Stephen James et al. "RLBench: The Robot Learning Benchmark & Learning Environment". In: *IEEE Robotics and Automation Letters* (2020).

[83] Liming Jiang et al. "TSIT: A Simple and Versatile Framework for Image-to-Image Translation". In: *ECCV*. 2020.

[84] Daniel Kappler, Jeannette Bohg, and Stefan Schaal. "Leveraging big data for grasp planning". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2015, pp. 4304–4311.

[85] Amlan Kar et al. "Meta-Sim: Learning to Generate Synthetic Datasets". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (2019), pp. 4550–4559.

[86] Andrej Karpathy. *CS231n: Convolutional Neural Networks for Visual Recognition*. https://cs231n.github.io/convolutional-networks/. 2016.

[87] Tero Karras, Samuli Laine, and Timo Aila. "A Style-Based Generator Architecture for Generative Adversarial Networks". In: *CVPR*. 2019.

[88] Tero Karras et al. "Progressive Growing of GANs for Improved Quality, Stability, and Variation". In: *ICLR* (2018).

[89] Tero Karras et al. *StyleGAN2-ADA Pytorch Implementation*. https://github.com/NVlabs/stylegan2-ada-pytorch. 2020.

[90] Tero Karras et al. “Training Generative Adversarial Networks with Limited Data”. In: *Thirty-fourth Conference on Neural Information Processing Systems*. Advances in neural information processing systems (NeurIPS). Morgan Kaufmann Publishers, 2020.

[91] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. “Neural 3D Mesh Renderer”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 3907–3916.

[92] Hiroharu Kato et al. “Differentiable Rendering: A Survey”. In: *arXiv* abs/2006.12057 (2020).

[93] Kenji Kawaguchi, Leslie Pack Kaelbling, and Yoshua Bengio. “Generalization in deep learning”. In: *arXiv preprint arXiv:1710.05468* (2017).

[94] Wadim Kehl et al. “SSD-6D: Making RGB-Based 3D Detection and 6D Pose Estimation Great Again”. In: *2017 IEEE International Conference on Computer Vision (ICCV)* (2017), pp. 1530–1538.

[95] Samin Khan et al. “ProcSy: Procedural Synthetic Dataset Generation Towards Influence Factor Studies Of Semantic Segmentation Networks”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. June 2019.

[96] Taeksoo Kim et al. “Learning to Discover Cross-Domain Relations with Generative Adversarial Networks”. In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, Aug. 2017, pp. 1857–1865.

[97] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *International Conference on Learning Representations (ICLR)* abs/1412.6980 (2015).

[98] Diederik P. Kingma and Max Welling. “Auto-Encoding Variational Bayes”. In: *International Conference on Learning Representations (ICLR)* abs/1312.6114 (2014).

[99] Nathan Koenig and Andrew Howard. “Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. Sendai, Japan, Sept. 2004, pp. 2149–2154.

[100] Nathan Koenig and Andrew Howard. “Design and use paradigms for gazebo, an open-source multi-robot simulator”. In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Vol. 3. IEEE, pp. 2149–2154.

[101] H. Kolivand et al. “Photorealistic rendering: a survey on evaluation”. In: *Multimedia Tools and Applications* 77 (2018), pp. 25983–26008.

[102] Eric Kolve et al. “AI2-THOR: An Interactive 3D Environment for Visual AI”. In: *ArXiv* abs/1712.05474 (2017).

[103] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012.

[104] Oliver Kroemer, S. Niekum, and G. Konidaris. “A Review of Robot Learning for Manipulation: Challenges, Representations, and Algorithms”. In: *J. Mach. Learn. Res.* 22 (2021), 30:1–30:82.

[105] Neeraj Kumar et al. “Describable Visual Attributes for Face Verification and Image Search”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.10 (2011), pp. 1962–1977.

[106] Ares Lagae et al. “A Survey of Procedural Noise Functions”. In: *Computer Graphics Forum* 29.8 (2010), pp. 2579–2600.

[107] Kuan-Ting Lai et al. “VIVID: Virtual Environment for Visual Deep Learning”. In: *Proceedings of the 26th ACM International Conference on Multimedia*. MM ’18. Seoul, Republic of Korea: Association for Computing Machinery, 2018, pp. 1356–1359.

[108] Brenden M. Lake et al. “Building machines that learn and think like people”. In: *Behavioral and Brain Sciences* 40 (2017), e253.

[109] Y. Le Cun et al. “Handwritten Digit Recognition: Applications of Neural Net Chips and Automatic Learning”. In: *Neurocomputing*. Ed. by Françoise Fogelman Soulié and Jeanny Hérault. Berlin, Heidelberg: Springer Berlin Heidelberg, 1990, pp. 303–318.

[110] Y. Lecun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[111] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *nature* 521.7553 (2015), pp. 436–444.

[112] Sergey Levine et al. “Learning Hand-Eye Coordination for Robotic Grasping with Large-Scale Data Collection”. In: *2016 International Symposium on Experimental Robotics*. Ed. by Dana Kulić et al. Cham: Springer International Publishing, 2017, pp. 173–184.

[113] Chuan Li. *OpenAI’s GPT-3 Language Model: A Technical Overview*. https://lambdalabs.com/blog/demystifying-gpt-3. 2020.

[114] Tzu-Mao Li et al. “Differentiable Monte Carlo Ray Tracing through Edge Sampling”. In: *ACM Trans. Graph.* 37.6 (Dec. 2018).

[115] Tsung Yi Lin et al. “unsupervised cross-domain image generation”. In: *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017* 2017-Janua (2017), pp. 936–944. arXiv: 1612.03144.

[116] Tsung-Yi Lin et al. “Microsoft COCO: Common Objects in Context”. In: *Computer Vision – ECCV 2014*. Ed. by David Fleet et al. Cham: Springer International Publishing, 2014, pp. 740–755.

[117] Ming-Yu Liu, Thomas Breuel, and Jan Kautz. “Unsupervised Image-to-Image Translation Networks”. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS’17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 700–708.

[118] Shichen Liu et al. “Soft Rasterizer: A Differentiable Renderer for Image-Based 3D Reasoning”. In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 7707–7716.

[119] Weibo Liu et al. “A survey of deep neural network architectures and their applications”. In: *Neurocomputing* 234 (2017), pp. 11–26.

[120] Ziwei Liu et al. “Deep Learning Face Attributes in the Wild”. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 3730–3738.

[121] Peter Longhurst, Patrick Ledda, and Alan Chalmers. “Psychophysically Based Artistic Techniques for Increased Perceived Realism of Virtual Environments”. In: *Proceedings of the 2nd International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa*. AFRIGRAPH ’03. Cape Town, South Africa: Association for Computing Machinery, 2003, pp. 123–132.

[122] Gilles Louppe, Joeri Hermans, and Kyle Cranmer. “Adversarial Variational Optimization of Non-Differentiable Simulators”. In: *The 22nd International Conference on Artificial Intelligence and Statistics, AISTATS 2019, 16-18 April 2019, Naha, Okinawa, Japan*. Ed. by Kamalika Chaudhuri and Masashi Sugiyama. Vol. 89. Proceedings of Machine Learning Research. PMLR, 2019, pp. 1438–1447.

[123] David G. Lowe. “Distinctive Image Features from Scale-Invariant Keypoints”. In: *International Journal of Computer Vision* 60 (2 2004), pp. 91–110.

[124] Laurens van der Maaten and Geoffrey Hinton. “Visualizing Data using t-SNE”. In: *Journal of Machine Learning Research* 9.86 (2008), pp. 2579–2605.

[125] Lilian Weng Maciek Chociej Peter Welinder. “ORRB: OpenAI Remote Rendering Backend”. In: *eprint arXiv*. June 2019. eprint: arXiv:1906.11633.

[126] Gary Marcus. “Deep learning: A critical appraisal”. In: *arXiv preprint arXiv:1801.00631* (2018).

[127] Jan Matas, Stephen James, and Andrew J Davison. “Sim-to-Real Reinforcement Learning for Deformable Object Manipulation”. In: *CoRL*. 2018. arXiv: 1806.07851v2.

[128] N. Mayer et al. “A Large Dataset to Train Convolutional Networks for Disparity, Optical Flow, and Scene Flow Estimation”. In: *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[129] Nikolaus Mayer et al. “What Makes Good Synthetic Training Data for Learning Disparity and Optical Flow Estimation?” In: *International Journal of Computer Vision* 126.9 (Apr. 2018), pp. 942–960.

[130] Brianna Maze et al. “IARPA Janus Benchmark - C: Face Dataset and Protocol”. In: *2018 International Conference on Biometrics (ICB)*. 2018, pp. 158–165.

[131] B McCane et al. “On Benchmarking Optical Flow”. In: *Computer Vision and Image Understanding* 84.1 (2001), pp. 126–143.

[132] John McCormac et al. “SceneNet RGB-D: Can 5M Synthetic Images Beat Generic ImageNet Pre-training on Indoor Segmentation?” In: *2017 IEEE International Conference on Computer Vision (ICCV)* (2017), pp. 2697–2706.

[133] Stephan Meister and Daniel Kondermann. “Real versus realistically rendered scenes for optical flow evaluation”. In: *2011 14th ITG Conference on Electronic Media Technology*. 2011, pp. 1–6.

[134] Douglas De Rizzo Meneghetti et al. “Annotated image dataset of household objects from the RoboFEI@Home team”. In: IEEE Dataport, 2020.

[135] Michele Merler et al. *Diversity in Faces*. 2019. arXiv: 1901.10436.

[136] Microsoft. *Visual Object Tagging Tool (VoTT)*. https://github.com/microsoft/VoTT. 2017.

[137] S. Minaee et al. “Image Segmentation Using Deep Learning: A Survey”. In: *IEEE Transactions on Pattern Analysis & Machine Intelligence* 01 (Feb. 5555), pp. 1–1.

[138] Mehdi Mirza and Simon Osindero. “Conditional generative adversarial nets”. In: *arXiv preprint arXiv:1411.1784* (2014).

[139] Thomas M. Mitchell. *Machine Learning*. 1st ed. USA: McGraw-Hill, Inc., 1997.

[140] Takeru Miyato et al. “Spectral Normalization for Generative Adversarial Networks”. In: *ICLR*. 2018.

[141] Stylianos Moschoglou et al. “AgeDB: The First Manually Collected, In-the-Wild Age Database”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (2017), pp. 1997–2005.

[142] Pushmeet Kohli Nathan Silberman Derek Hoiem and Rob Fergus. “Indoor Segmentation and Support Inference from RGBD Images”. In: *ECCV*. 2012.

[143] Thu Nguyen-Phuoc et al. “RenderNet: A deep convolutional network for differentiable rendering from 3D shapes”. In: *NeurIPS*. 2018.

[144] NVIDIA. *NVIDIA Isaac Simulator*. URL: https://developer.nvidia.com/isaac-sim.

[145] Niall O’Mahony et al. “Deep Learning vs. Traditional Computer Vision”. In: *Advances in Computer Vision*. Ed. by Kohei Arai and Supriya Kapoor. Cham: Springer International Publishing, 2020, pp. 128–144.

[146] Niall O’Mahony et al. “Deep learning vs. traditional computer vision”. In: *Science and Information Conference*. Springer. 2019, pp. 128–144.

[147] OpenAI et al. *Learning Dexterous In-Hand Manipulation*. 2018. arXiv: 1808.00177.

[148] OpenCV. *Computer Vision Annotation Tool (CVAT)*. https://github.com/openvinotoolkit/cvat. 2018.

[149] Sinno Jialin Pan and Qiang Yang. “A Survey on Transfer Learning”. In: *IEEE Transactions on Knowledge and Data Engineering* 22.10 (2010), pp. 1345–1359.

[150] Taesung Park et al. “Semantic Image Synthesis with Spatially-Adaptive Normalization”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019.

[151] F. Pereira, P. Norvig, and A. Halevy. “The Unreasonable Effectiveness of Data”. In: *IEEE Intelligent Systems* 24.02 (Mar. 2009), pp. 8–12.

[152] Ken Perlin. “Improving Noise”. In: *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’02. San Antonio, Texas: ACM, 2002, pp. 681–682.

[153] Bui Tuong Phong. “Illumination for computer generated pictures”. In: *Communications of the ACM* 18 (1975), pp. 311–317.

[154] Pedro O Pinheiro. “Unsupervised Domain Adaptation with Similarity Learning”. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2018, pp. 8004–8013.

[155] Lerrel Pinto et al. “Asymmetric Actor Critic for Image-Based Robot Learning”. In: *Robotics: Science and Systems XIV*. 2018. arXiv: 1710.06542v1.

[156] François Pitié, Anil C. Kokaram, and Rozenn Dahyot. “Automated colour grading using colour distribution transfer”. In: *Computer Vision and Image Understanding* 107.1 (2007). Special issue on color image processing, pp. 123–137.

[157] Tiziano Portenier, S. Bigdeli, and O. Goksel. “GramGAN: Deep 3D Texture Synthesis From 2D Exemplars”. In: *ArXiv* abs/2006.16112 (2020).

[158] J. Portilla and Eero P. Simoncelli. “A Parametric Texture Model Based on Joint Statistics of Complex Wavelet Coefficients”. In: *International Journal of Computer Vision* 40 (2004), pp. 49–70.

[159] Samira Pouyanfar et al. “ROADS : Randomization for Obstacle Avoidance and Driving in Simulation”. In: *Computer Vision and Pattern Recognition Workshops*. 2019.

[160] Aayush Prakash et al. “Structured Domain Randomization: Bridging the Reality Gap by Context-Aware Synthetic Data”. In: *2019 International Conference on Robotics and Automation (ICRA)* (2018), pp. 7249–7255.

[161] Weichao Qiu et al. “UnrealCV: Virtual Worlds for Computer Vision”. In: *ACM Multimedia Open Source Software Competition* (2017).

[162] Mahdi Rad and Vincent Lepetit. *BB8: A Scalable, Accurate, Robust to Partial Occlusion Method for Predicting the 3D Poses of Challenging Objects without Using Depth*. Tech. rep. arXiv: 1703.10896v2.

[163] Paul Rademacher et al. "Measuring the Perception of Visual Realism in Images". In: *Proceedings of the 12th Eurographics Conference on Rendering*. EGWR'01. London, UK: Eurographics Association, 2001, pp. 235–248.

[164] Inioluwa Deborah Raji and Joy Buolamwini. "Actionable Auditing: Investigating the Impact of Publicly Naming Biased Performance Results of Commercial AI Products". In: *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*. AIES '19. Honolulu, HI, USA: Association for Computing Machinery, 2019, pp. 429–435.

[165] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 779–788.

[166] E. Reinhard et al. "Color transfer between images". In: *IEEE Computer Graphics and Applications* 21.5 (2001), pp. 34–41.

[167] Erik Reinhard et al. "On Visual Realism of Synthesized Imagery". In: *Proceedings of the IEEE* 101.9 (2013), pp. 1998–2007.

[168] Shaoqing Ren et al. "Faster r-cnn: Towards real-time object detection with region proposal networks". In: *Advances in neural information processing systems* 28 (2015), pp. 91–99.

[169] Colin Rennie et al. "A Dataset for Improved RGBD-Based Object Detection and Pose Estimation for Warehouse Pick-and-Place". In: *IEEE Robotics and Automation Letters* 1 (2016), pp. 1179–1185.

[170] Stephan R. Richter, Hassan Abu AlHaija, and Vladlen Koltun. "Enhancing Photorealism Enhancement". In: *arXiv:2105.04619* (2021).

[171] Stephan R. Richter et al. “Playing for Data: Ground Truth from Computer Games”. In: *European Conference on Computer Vision (ECCV)*. Ed. by Bastian Leibe et al. Vol. 9906. LNCS. Springer International Publishing, 2016, pp. 102–118.

[172] Mike Roberts et al. “Hypersim: A Photorealistic Synthetic Dataset for Holistic Indoor Scene Understanding”. In: *International Conference on Computer Vision (ICCV) 2021*. 2021.

[173] E. Rohmer, S. P. N. Singh, and M. Freese. “CoppeliaSim (formerly V-REP): a Versatile and Scalable Robot Simulation Framework”. In: *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*. 2013.

[174] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-net: Convolutional networks for biomedical image segmentation”. In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

[175] Rasmus Rothe, Radu Timofte, and Luc Van Gool. “Deep Expectation of Real and Apparent Age from a Single Image Without Facial Landmarks”. In: *International Journal of Computer Vision* (Aug. 2016).

[176] Ethan Rublee et al. “ORB: An efficient alternative to SIFT or SURF”. In: *2011 International Conference on Computer Vision*. 2011, pp. 2564–2571.

[177] Nataniel Ruiz, Samuel Schulter, and Manmohan Chandraker. “Learning To Simulate”. In: *International Conference on Learning Representations*. 2019.

[178] Olga Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. In: *Int. J. Comput. Vision* 115.3 (Dec. 2015), pp. 211–252.

[179] Stuart Russell and Peter Norvig. “Artificial Intelligence: A Modern Approach, Global Edition 4th”. In: *Foundations* 19 (2021), p. 23.

[180] Fereshteh Sadeghi and Sergey Levine. “CAD2RL: Real Single-Image Flight without a Single Real Image”. In: (2016). arXiv: 1611.04201.

[181] Kate Saenko. *Taming Dataset Bias via Domain Adaptation*. 2021. URL: http://introtodeeplearning.com/.

[182] Jürgen Schmidhuber. "Deep learning in neural networks: An overview". In: *Neural Networks* 61 (2015), pp. 85–117.

[183] Akash Sengupta, Ignas Budvytis, and Roberto Cipolla. "Synthetic Training for Accurate 3D Human Pose and Shape Estimation in the Wild". In: *BMVC*. 2020.

[184] Zhiqiang Shen et al. "Object detection from scratch with deep supervision". In: *IEEE transactions on pattern analysis and machine intelligence* 42.2 (2019), pp. 398–412.

[185] Yosuke Shinya, Edgar Simo-Serra, and Taiji Suzuki. "Understanding the Effects of Pre-Training for Object Detectors via Eigenspectrum". In: *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*. 2019, pp. 1931–1941.

[186] Ashish Shrivastava et al. "Learning from simulated and unsupervised images through adversarial training". In: *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*. Vol. 2017-Janua. 2017, pp. 2242–2251. arXiv: 1612.07828.

[187] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *ICLR* abs/1409.1556 (2015).

[188] Suraj Pal Singh and Shano Solanki. "Recommender System Survey: Clustering to Nature Inspired Algorithm". In: *Proceedings of 2nd International Conference on Communication, Computing and Networking*. Ed. by C. Rama Krishna, Maitreyee Dutta, and Rakesh Kumar. Singapore: Springer Singapore, 2019, pp. 757–768.

[189] Piotr Skalski. *Make Sense*. https://github.com/SkalskiP/make-sense/. 2019.

[190] Hwanjun Song et al. "Learning from noisy labels with deep neural networks: A survey". In: *arXiv preprint arXiv:2007.08199* (2020).

[191] Shuran Song et al. "Semantic Scene Completion from a Single Depth Image". In: *Proceedings of 30th IEEE Conference on Computer Vision and Pattern Recognition* (2017).

[192] Hao Su et al. “Render for CNN: Viewpoint estimation in images using CNNs trained with rendered 3D model views”. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 2686–2694.

[193] Chen Sun et al. “Revisiting Unreasonable Effectiveness of Data in Deep Learning Era”. In: *ICCV*. Oct. 2017, pp. 843–852.

[194] Shiliang Sun, Honglei Shi, and Yuanbin Wu. “A survey of multi-source domain adaptation”. In: *Information Fusion* 24 (2015), pp. 84–92.

[195] Martin Sundermeyer. “Implicit 3D Orientation Learning for 6D Object Detection from RGB Images”. In: *ECCV*. 2018.

[196] Christian Szegedy et al. “Going deeper with convolutions”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 1–9.

[197] Unity Technologies. URL: https://unity.com/.

[198] Bugra Tekin, Sudipta N Sinha, and Pascal Fua. *Real-Time Seamless Single Shot 6D Object Pose Prediction*. Tech. rep.

[199] Maoqing Tian et al. “Eliminating Background-Bias for Robust Person Re-Identification”. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018*. 2018.

[200] Thang To et al. *NDDS: NVIDIA Deep Learning Dataset Synthesizer*. https://github.com/NVIDIA/Dataset_Synthesizer. 2018.

[201] Josh Tobin et al. “Domain Randomization and Generative Models for Robotic Grasping”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2018), pp. 3482–3489.

[202] Josh Tobin et al. “Domain randomization for transferring deep neural networks from simulation to the real world”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2017), pp. 23–30.

[203] Emanuel Todorov, Tom Erez, and Yuval Tassa. “MuJoCo: A physics engine for model-based control”. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, pp. 5026–5033.

[204] Jonathan Tremblay, Thang To, and Stan Birchfield. “Falling Things: A Synthetic Dataset for 3D Object Detection and Pose Estimation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. June 2018.

[205] Jonathan Tremblay et al. “Deep Object Pose Estimation for Semantic Robotic Grasping of Household Objects”. In: *CoRL*. 2018. arXiv: 1809.10790v1.

[206] Jonathan Tremblay et al. “Synthetically Trained Neural Networks for Learning Human-Readable Plans from Real-World Demonstrations”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)* (2018), pp. 1–5.

[207] Jonathan Tremblay et al. “Training Deep Networks with Synthetic Data: Bridging the Reality Gap by Domain Randomization”. In: *CVPR Workshop on Autonomous Driving*. 2018.

[208] Eric Tzeng et al. “Adapting Deep Visuomotor Representations with Weak Pairwise Constraints”. In: *arXiv preprint* arXiv: 1511.07111 (2017). arXiv: 1511.07111.

[209] tzutalin. *LabelImg*. https://github.com/tzutalin/labelImg. 2018.

[210] Grant Van Horn and Pietro Perona. “The devil is in the tails: Fine-grained classification in the wild”. In: *arXiv preprint arXiv:1709.01450* (2017).

[211] Gül Varol et al. “Learning from Synthetic Humans”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 4627–4635.

[212] Kentaro Wada. *labelme: Image Polygonal Annotation with Python*. https://github.com/wkentaro/labelme. 2016.

[213] Fei Wang et al. “The Devil of Face Recognition Is in the Noise”. In: *Computer Vision – ECCV 2018*. Ed. by Vittorio Ferrari et al. Cham: Springer International Publishing, 2018, pp. 780–795.

[214] Mei Wang and Weihong Deng. “Deep visual domain adaptation: A survey”. In: *Neurocomputing* 312 (2018), pp. 135–153.

[215] Miao Wang et al. “Example-Guided Style-Consistent Image Synthesis from Semantic Labeling”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019.

[216] Q. Wang et al. “Learning From Synthetic Data for Crowd Counting in the Wild”. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019), pp. 8190–8199.

[217] Ting-Chun Wang et al. “High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018.

[218] Ting-Chun Wang et al. “Video-to-Video Synthesis”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2018.

[219] Daniel Ward, Peyman Moghadam, and Nicolas Hudson. “Deep Leaf Segmentation Using Synthetic Data”. In: *BMVC*. 2019. arXiv: 1807.10931.

[220] Li-Yi Wei and Marc Levoy. “Fast Texture Synthesis Using Tree-Structured Vector Quantization”. In: *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’00. USA: ACM Press/Addison-Wesley Publishing Co., 2000, pp. 479–488.

[221] Ronald J. Williams. “Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning”. In: *Mach. Learn.* 8.3–4 (May 1992), pp. 229–256.

[222] Yuxin Wu et al. *Detectron2*. https://github.com/facebookresearch/detectron2. 2019.

[223] Jonas Wulff et al. “Lessons and Insights from Creating a Synthetic Optical Flow Benchmark”. In: *Computer Vision – ECCV 2012. Workshops and Demonstrations*. Ed. by Andrea Fusiello, Vittorio Murino, and Rita Cucchiara. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 168–177.

[224] Bernhard Wymann et al. *TORCS: The open racing car simulator*. 2015.

[225] Fei Xia et al. "Interactive Gibson Benchmark: A Benchmark for Interactive Navigation in Cluttered Environments". In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 713–720.

[226] Wenqi Xian et al. "TextureGAN: Controlling Deep Image Synthesis with Texture Patches". In: *CVPR*. June 2018, pp. 8456–8465.

[227] Fanbo Xiang et al. "SAPIEN: A SimulAted Part-based Interactive ENvironment". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020, pp. 11094–11104.

[228] Yu Xiang et al. *PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes*. Tech. rep. arXiv: 1711.00199v3.

[229] Xuezhong Xiao and Lizhuang Ma. "Color Transfer in Correlated Color Space". In: *Proceedings of the 2006 ACM International Conference on Virtual Reality Continuum and Its Applications*. VRCIA '06. Hong Kong, China: Association for Computing Machinery, 2006, pp. 305–309.

[230] Xu Xie et al. "VRGym: A Virtual Testbed for Physical and Interactive AI". In: *Proceedings of the ACM Turing Celebration Conference - China*. ACM TURC '19. Chengdu, China: Association for Computing Machinery, 2019.

[231] Zhenfeng Xue, Weijie Mao, and Liang Zheng. *Learning to simulate complex scenes*. 2020. arXiv: 2006.14611.

[232] M Yan, S Tyree, and J Kautz. "Sim-to-Real Transfer of Accurate Grasping with Eye-In-Hand Observations and Continuous Control". In: *NIPS Workshop on Acting and Interacting in the Real World: Challenges in Robot Learning* (2017).

[233] Zili Yi et al. "DualGAN: Unsupervised Dual Learning for Image-to-Image Translation". In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 2868–2876.

[234] Fisher Yu et al. “BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning”. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020, pp. 2633–2642.

[235] Ekim Yurtsever et al. “A Survey of Autonomous Driving: Common Practices and Emerging Technologies”. In: *IEEE Access* 8 (2020), pp. 58443–58469.

[236] Chiyuan Zhang et al. “Understanding deep learning (still) requires rethinking generalization”. In: *Communications of the ACM* 64.3 (2021), pp. 107–115.

[237] Fangyi Zhang et al. “Adversarial discriminative sim-to-real transfer of visuo-motor policies”. In: *I. J. Robotics Res.* 38 (2018).

[238] Fangyi Zhang et al. “Sim-to-real Transfer of Visuo-motor Policies for Reaching in Clutter: Domain Randomization and Adaptation with Modular Networks”. In: *arXiv preprint* arXiv: 1709.05746 (2017). arXiv: 1709.05746.

[239] Quanshi Zhang and Song-Chun Zhu. “Visual interpretability for deep learning: a survey”. In: *Frontiers of Information Technology & Electronic Engineering* 19 (2018), pp. 27–39.

[240] Shuai Zhang et al. “Deep Learning Based Recommender System: A Survey and New Perspectives”. In: *ACM Comput. Surv.* 52.1 (Feb. 2019).

[241] Yi Zhang et al. “Unrealstereo: Controlling hazardous factors to analyze stereo vision”. In: *2018 International Conference on 3D Vision (3DV)*. IEEE. 2018, pp. 228–237.

[242] Yingying Zhang et al. “Single-Image Crowd Counting via Multi-Column Convolutional Neural Network”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 589–597.

[243] H. Zhao et al. “Pyramid Scene Parsing Network”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, July 2017, pp. 6230–6239.

[244] Jia Zheng et al. “Structured3D: A Large Photo-realistic Dataset for Structured 3D Modeling”. In: *Proceedings of The European Conference on Computer Vision (ECCV)*. 2020, pp. 519–535.

[245] Jun-Yan Zhu et al. “Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks”. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 2242–2251.

[246] Peihao Zhu et al. “SEAN: Image Synthesis With Semantic Region-Adaptive Normalization”. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020, pp. 5103–5112.

[247] Zhe Zhu et al. “Traffic-Sign Detection and Classification in the Wild”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 2110–2118.

[248] Christian Zimmermann and Thomas Brox. “Learning to Estimate 3D Hand Pose from Single RGB Images”. In: *IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 4913–4921.