



Learning to Estimate 3D Object Pose from Synthetic Data

Sergey Zakharov

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:

Prof. Dr. Daniel Cremers

Prüfende der Dissertation:

1. Priv.-Doz. Dr. Slobodan Ilic
2. Prof. Dr. Vincent Lepetit

Die Dissertation wurde am 15.07.2020 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 28.10.2020 angenommen.

Abstract

3D object pose estimation is one of the most important problems in computer vision with applications in robotics, augmented reality, autonomous driving, medicine and many other areas. This thesis introduces a number of deep learning-based solutions to this task with a focus on industrial usability. The first part presents pose estimation methods of various complexity relying on manifold learning, dense correspondences, and differentiable rendering. This is followed by works explicitly tackling the domain gap problem when training from synthetic data using the introduced reverse domain adaptation and adversarial domain randomization techniques.

In the first part of the thesis, we address the core problem of 3D object pose estimation. We gradually increase the problem’s complexity, starting from a simple 3D rotation estimation, continuing further to estimating the 6D pose by adding a translation component, and finishing with a full-fledged 9D pose estimation pipeline predicting both the pose, scale, and shape of the object. The first method aims at learning a discriminative feature space that allows to effectively retrieve both the pose and class using scalable nearest neighbor search methods. Subsequently, we improve the solution by introducing a multi-task pipeline, which combines the strengths of manifold learning and regression making the entire pipeline end-to-end. Next, we propose a 6D pose estimation solution based on dense correspondences and a fast PnP solver. This technique significantly improves the quality of poses, is more robust to occlusions and clutter, while remaining real time. Moreover, we introduce a deep learning-based refiner that further improves the pose quality. The last method provides a solution to the more ambitious 9D pose estimation problem. Here, we not only estimate the rotation and translation of the object, but also its scale and shape. It utilizes a more advanced dense correspondence network, differentiable SDF shape database, and a newly introduced differentiable SDF renderer. Furthermore, we present a new challenging dataset for 6D pose estimation and a number of benchmarks to test current state-of-the-art detectors with respect to the properties desired in the industry.

The majority of deep learning models learn from labeled data, which are expensive and often infeasible to acquire. The sound alternative is to use available synthetic CAD models to generate free training data and annotations. However, models trained on such data perform poorly on real images due to a significant domain gap. The second part of this thesis, introduces two methods tackling this problem. The first method is based on an idea of the reverse domain adaptation in depth and RGB data. Instead of the common pipeline that maps synthetic images into the real domain, we train networks to map real images back to the synthetic domain. This not only allows to improve the overall performance of the system, but also introduces some practical benefits, i.e., the downstream network can be trained purely on synthetic data and never

Abstract

has to be retrained, as opposed to the standard pipelines. The second method utilizes an adversarial procedure to make the task network robust to the possible appearance changes defined by specifically designed differentiable modules. The procedure uses an introduced deception network that modifies input images while still preserving their geometrical meaning. The task network is trained together with the deception network utilizing a min-max procedure and, as a result, it becomes much more robust to possible appearance changes.

Zusammenfassung

Die 3D-Objektposenschätzung ist eine der wichtigsten Probleme im Bereich des maschinellen Sehens mit einer Reihe von Anwendungen in Robotik, Augmented Reality, autonomen Fahren, Medizin und vielen anderen Bereichen. Diese Arbeit befasst sich mit den Aufgaben der 3D-Objektposenschätzung und Domain Adaptation basierend auf Deep-Learning Methoden. Der erste Teil präsentiert Ansätze von unterschiedlicher Komplexität zur Bestimmung einer Pose, welche auf Manifold Learning, dichten Punktkorrespondenzen und differenzierbarem Rendering beruhen. Darauf folgen eine Reihe von Arbeiten, welche sich explizit mit dem Domain-Gap Problem auseinandersetzen, welches entsteht wenn auf synthetischen Daten trainiert wird. Es wird versucht, dieses mit Hilfe der vorgestellten Techniken, Reverse Domain Adaptation und Adversarial Domain Randomization zu lösen.

Der erste Teil der Arbeit thematisiert das Kernproblem der Posenschätzung von 3D-Objekten. Wir erhöhen die Komplexität des Problems schrittweise, angefangen mit einer einfachen 3D-Rotationsschätzung, weiter bis zur Schätzung der 6D-Pose und endend mit einer vollwertigen 9D-Posenschätzungspipeline, die sowohl die Pose als auch den Maßstab und die Form des Objekts schätzt. Die erste Methode zielt darauf ab, einen Funktionsraum zu lernen, der es ermöglicht, sowohl die Pose als auch die Klasse unter Verwendung skalierbarer Nearest-Neighbour-Suchmethoden effektiv abzurufen. Anschließend verbessern wir die Lösung durch die Einführung einer Multi-Task-Pipeline, die die Stärken des Multitask-Lernens und der Posenregression kombiniert und die gesamte Pipeline durchgängig macht. Als nächstes schlagen wir eine Lösung zur 6D-Posenschätzung vor, die auf dichten Punktkorrespondenzen und einem schnellen PnP-Löser basiert. Diese Technik verbessert die Qualität von Posen erheblich, ist robuster gegenüber Okklusionen und echtzeitfähig. Darüber hinaus führen wir einen Deep-Learning-basierten Refiner ein, der die Posenqualität weiter verbessert. Die letzte Methode bietet eine Lösung für das anspruchsvollere Problem der 9D-Posenschätzung. Hier schätzen wir nicht nur die Rotation und Translation des Objekts, sondern auch dessen Maßstab und Form. Es verwendet ein fortschrittlicheres dichtes Korrespondenznetzwerk, eine differenzierbare SDF-Formdatenbank und einen neu eingeführten differenzierbaren SDF-Renderer. Darüber hinaus präsentieren wir einen neuen Datensatz für die 6D-Posenschätzung und eine Reihe von Benchmarks, um die aktuellen Detektoren hinsichtlich der in der Branche gewünschten Eigenschaften zu testen.

Die meisten Deep-Learning-Modelle lernen aus gelabelten Daten, die teuer und oft nicht realisierbar sind. Eine Alternative hierzu besteht darin, verfügbare synthetische CAD-Modelle zu verwenden, um kostenlose Trainingsdaten zu generieren. An solchen Daten trainierte Modelle weisen jedoch aufgrund eines erheblichen Domain-Gaps eine schlechte Leistung bei realen Bildern auf. Im zweiten Teil dieser Arbeit werden zwei

Zusammenfassung

Methoden vorgestellt, mit denen dieses Problem gelöst werden kann. Die erste Methode basiert auf einer Idee der umgekehrten Domain Adaptation in Tiefen- und RGB-Daten. Anstelle der üblichen Pipeline, die synthetische Bilder auf die reale Domäne abbildet, trainieren wir Netzwerke, um reale Bilder wieder auf die synthetische Domäne abzubilden. Dies ermöglicht nicht nur die Verbesserung der Gesamtleistung des Systems, sondern bringt auch einige praktische Vorteile mit sich, d. H. das nachgeschaltete Netzwerk kann rein auf synthetischen Daten trainiert werden und muss im Gegensatz zu den Standard-Pipelines nie umgestellt werden. Die zweite Methode verwendet Adversarial Domain Randomization, um das Task-Netzwerk gegenüber möglichen Änderungen des Erscheinungsbilds robust zu machen, die durch speziell entworfene differenzierbare Module definiert werden. Das Verfahren verwendet das Deception-Netzwerk, welches Eingabebilder modifiziert und dabei ihre geometrische Bedeutung beibehält. Das Task-Netzwerk wird zusammen mit dem Deception-Netzwerk unter Verwendung eines Min-Max-Verfahrens trainiert und dadurch viel robuster gegenüber möglichen Änderungen des Erscheinungsbildes.

Acknowledgments

First of all, I would like to thank PD Dr. Slobodan Ilic and Dr. Andreas Hutter for providing me an opportunity to conduct this work in collaboration with Siemens Corporate Technology. I would like to thank PD Dr. Slobodan Ilic additionally for giving me an explicit research direction and for guiding me throughout the thesis. I am very grateful to Wadim Kehl, with whom I have actively collaborated throughout these years and who introduced me to the software frameworks and methodologies that enabled me to have a solid starting point. I also want to thank Benjamin Planche and Mai Bui, my collaborators, for the fun research time spent together, responsiveness and willingness to help with any of the upcoming problems. I'm happy to have my PhD mate Haowen Deng, with whom I started and finished this journey, for the refreshing conversations, always being there to help, and shared suffering. Further, I want to thank Prof. Dr. Vincent Lepetit for serving on my committee and Prof. Dr. Daniel Cremers for serving as a committee's chairman.

I would also like to express gratitude to the other colleagues from the Department for Digital Perception at Siemens and the Chair for Computer Aided Medical Procedures at TUM: Mira Slavcheva, Tolga Birdal, Fabian Bauer, Adrian Haarbach, Andres Sanchez, Ivan Shugurov, Roman Kaskman, Agnieszka Tomczak, Ivan Pavlov, Mahdi Hamad, PD Dr. Federico Tombari, Fabian Manhardt, Helisa Dharmo, Johanna Wald, Sailesh Conjeti, Christian Rupprecht, Iro Laina, Anees Kazi, Gerome Vivar, Nikolas Brasch, Huseyin Coskun, Mahdi Saleh, Ashkan Khakzar, Fernando Navarro, Magda Paschali, Yida Wang, Jakob Mayr, Oliver Scheel and others for the great time and discussions. Moreover, a big thanks goes to the Machine Learning team at Toyota Research Institute: Adrien Gaidon, Arjun Bhargava, Dennis Park, Chao Fang, Rares Ambrus, Sudeep Pillai, Quincy Chen, Vitor Guizilini, Kuan-Hui Lee, Jie Li, and Allan Raventos for the great and fruitful internship time (and all the free food) which led to a publication.

Last but not least, I want thank my caring friends Stefan & Olga Gavranovic, and Denys & Sasha Korzh, who made my life in Munich much easier and happier and with whom I share so many beautiful memories. And finally, I want to express gratitude to my family, my parents Galina and Nikolai, my sister Ksenia, and my wife Alina, for their unconditional love, support, and advice.

– Sergey

Contents

Abstract	iii
Zusammenfassung	v
Acknowledgments	vii
Contents	ix
List of Figures	xiii
List of Tables	xix
1 Introduction	1
1.1 Challenges of Industrial Computer Vision	1
1.2 Contributions	4
1.3 Outline	6
2 Background	9
2.1 Neural Networks	9
2.1.1 Fully Connected Neural Networks	10
2.1.2 Convolutional Neural Networks	16
2.1.3 Optimization	17
2.2 Projective Geometry	19
2.2.1 Rigid Body Transformations	19
2.2.2 Pinhole Camera Model	21
2.3 Pose Estimation	22
2.3.1 Template-Based Methods	22
2.3.2 Correspondence-Based Methods	24
2.3.3 Direct Pose Regression Methods	27
2.3.4 Pose Refinement	27
2.4 Domain Adaptation	29
I Pose Estimation	33
3 3D Pose Estimation Based on Manifold Learning	37
3.1 Introduction	37

CONTENTS

3.2	Methodology	39
3.2.1	Loss Function	40
3.2.1.1	Triplet Loss with Dynamic Margin	41
3.2.1.2	Multitask Loss	42
3.2.2	Dataset Generation	42
3.2.2.1	In-plane Rotations	44
3.2.2.2	Treating Rotation-Invariant Objects	45
3.2.2.3	Surface Normals	46
3.2.2.4	Background Noise Generator	46
3.3	Evaluation	47
3.3.1	Tests on In-plane Rotations	48
3.3.2	Tests on the Dynamic Margin Triplet Loss	49
3.3.3	Tests on Background Noise Types	50
3.3.4	Tests on Input Image Channels	51
3.3.5	Tests on Larger Datasets	51
3.3.6	Combining Manifold Learning and Regression	52
3.3.6.1	Multi-Task Learning vs Single-Task Learning	52
3.3.6.2	Influence of Network Architecture	53
3.3.6.3	Feature Visualization	53
3.3.6.4	Scalability	55
3.3.6.5	Sensitivity to Regularization Parameter λ	55
3.4	Conclusion	56
4	6D Pose Estimation Based on Dense Correspondences	57
4.1	Introduction	57
4.2	Methodology	58
4.2.1	Data Preparation	58
4.2.1.1	Correspondence Mapping	60
4.2.1.2	Online Data Generation and Augmentation	61
4.3	Dense Object Detection Pipeline	61
4.4	Deep Model-Based Pose Refinement	62
4.5	Training Details	64
4.6	Evaluation	64
4.6.1	Datasets	65
4.6.2	Evaluation Metrics	65
4.6.3	Single Object Pose Estimation	66
4.6.4	Multiple Object Pose Estimation	68
4.6.5	Ablation Study	68
4.6.5.1	RANSAC Iterations	68
4.6.5.2	Runtime Analysis	69
4.6.5.3	Correspondence Quality	69
4.6.5.4	Refinement	70
4.7	Conclusion	71

5	9D Pose Estimation for Autolabeling	73
5.1	Introduction	73
5.2	Methodology	75
5.2.1	Coordinate Shape Space	75
5.2.2	Differentiable SDF Rendering	76
5.2.3	3D Autolabeling Pipeline	78
5.2.3.1	Initialization and Optimization	79
5.2.3.2	Verification and CSS Retraining	81
5.3	Experimental Evaluation	82
5.3.1	Correctness of Autolabels	83
5.3.2	Ablation	84
5.3.3	Autolabeling for 3D Object Detection	85
5.4	Conclusion	86
6	RGB-D 6D Pose Estimation Dataset	87
6.1	Introduction	87
6.2	Related Datasets	89
6.3	HomebrewedDB Dataset Creation	90
6.3.1	Calibration of RGB-D Sensors	91
6.3.2	Sequence Acquisition	91
6.3.3	3D Model Reconstruction	92
6.3.4	Depth Correction	93
6.3.5	Creation of Ground Truth Annotations	93
6.3.6	Accuracy of Ground Truth Poses	94
6.4	Benchmarks and Experiments	95
6.4.1	Evaluation Metrics	95
6.4.2	Scalability Benchmark	95
6.4.3	Scene Benchmarks	96
6.4.4	Domain Adaptation Benchmark	97
6.4.5	Drawbacks of Training on Real Data	97
6.5	Conclusion	99
II	Domain Adaptation in Depth and RGB	101
7	Reverse Domain Adaptation	105
7.1	Introduction	105
7.2	Methodology	107
7.2.1	GAN-Based Architecture for Depth	107
7.2.2	Multi-Modal U-Net Architecture for RGB	109
7.2.3	Learning from Purely Geometrical CAD Data	112
7.2.3.1	Synthetic Data Generation	112
7.2.3.2	Online Data Augmentation	113

CONTENTS

7.3	Experimental Evaluation	117
7.3.1	Experimental Setup	117
7.3.2	Real Depth \rightarrow Synthetic Depth	118
7.3.2.1	Comparison to Usual Domain Adaptation GANs	119
7.3.2.2	Ablation of the Solution Components	120
7.3.3	Real RGB \rightarrow Synthetic Normals / Depth	121
7.3.3.1	Ablation of the Solution Components	122
7.4	Conclusion	123
8	Network-Driven Domain Randomization	125
8.1	Introduction	125
8.2	Methodology	126
8.2.1	Deception Modules	127
8.2.1.1	Background Module (BG)	128
8.2.1.2	Distortion Module (DS)	128
8.2.1.3	Noise Module (NS)	129
8.2.1.4	Light Module (L)	129
8.2.2	Optimization Objective	130
8.2.3	Training Procedure	130
8.3	Experimental Evaluation	130
8.3.1	Adaptation Tests	131
8.3.1.1	Classification on MNIST	132
8.3.1.2	Classification and Pose Estimation on LineMOD	132
8.3.2	Generalization Tests	133
8.3.3	Ablation Studies	135
8.3.3.1	Deception Modules	135
8.3.3.2	Input Modalities	136
8.3.4	Real-World Scenario	136
8.4	Conclusion	137
9	Conclusion and Outlook	139
9.1	Summary	139
9.2	Limitations and Future Work	140
A	Authored and Co-Authored Publications	141
	Bibliography	143

List of Figures

1.1	6D pose estimation on a tablet. Our RGB dense pose object detector (DPOD) from Chapter 4 estimates 6D pose of the object in real time running on a tablet.	2
1.2	3D object pose estimation. Given a collection of 3D object models, the goal is to estimate the pose for each of the objects of interest in RGB / RGB-D image.	3
1.3	Domain gap. Synthetic data are very different from real images coming from commodity sensors not only due to the sensor-specific properties, but also due to external illumination and object appearance changes. The problem becomes even more difficult when objects have no texture.	4
2.1	Neuron representations: Artificial neuron model is a very rough approximation of the biological model.	9
2.2	Network structures. Going from a single to multiple layers.	10
2.3	Activation functions. Step function and its smooth alternatives.	11
2.4	Sample neural network to illustrate the forward and backward propagation processes.	12
2.5	Possible loss function landscapes. In many real-world applications we have to deal with multidimensional non-convex loss functions.	13
2.6	Receptive fields. Groups of pixels are considered for the next layer, as opposed to treating each pixel individually.	16
2.7	Feature maps and features. Convolutional layers consist of feature maps each defined by a set of shared weights (features) and a single shared bias.	17
2.8	Pooling layer. Used to decrease the number of parameters while keeping the relative spatial structure.	18
2.9	Pinhole camera model. Image is taken from [1].	21
2.10	Template-based methods. A database of templates covers possible object views. Each template is compared to the input image and the best fitting template is used as a pose estimate.	23
2.11	Correspondence-based methods. Given a set of 1-to-1 correspondences between the input image and 3D object model, we can recover the pose by aligning correspondences minimizing a predefined error metric.	25
2.12	Direct pose regression. A neural network maps an input image containing the object directly to its pose estimate.	26

LIST OF FIGURES

2.13 **Domain adaptation methods.** An overview of different method classes tackling the domain gap problem. 30

2.14 **3D Object Pose Estimation.** Given an input image, the aim is to recognize object instances and estimate their poses. This allows to project them back onto the image. 35

3.1 **Pipeline description.** Given an input image patch \mathbf{x}_i , we create corresponding triplets $(\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k)$ and pairs $(\mathbf{x}_i, \mathbf{x}_j)$ to train our model on manifold embedding creating robust feature descriptors. Additionally, the pose can be regressed directly by introducing the regression head as demonstrated in our multi-task pipeline extension. The pose \mathbf{q} can then be obtained either by a direct pose regression or using the resulting feature descriptor for nearest neighbor search in the descriptor database. 39

3.2 **CNN input format.** Triplets are used to learn a well-separated manifold, whereas pairs make the mapping invariant to various imaging conditions. 40

3.3 **Triplet loss with dynamic margin.** A better separation achieved by setting different inter- and intra-class margins. 41

3.4 **Different sampling types.** each vertex represents a camera position from which the object is rendered. 42

3.5 **Patch extraction.** the object of interest (shown in yellow) is covered by the cube of 40 cm^3 in dimension; only RGB and depth data covered by the cube is taken to generate a single patch. 43

3.6 **Datasets.** The training set S_{train} consists of both real and synthetic (fine sampling); the test set S_{test} consists of the real data not used for the training set S_{train} 44

3.7 **In-plane rotations.** At each vertex extra views are rendered by rotating the camera around the axis pointing at the object center. 45

3.8 **Sampling points for different objects types.** Vertices represent camera positions from which the object is rendered. 45

3.9 **Background noise types** for synthetic data shown for different channels, i.e., RGB, depth, and normals. 46

3.10 **Test set samples mapped to a 3D descriptor space.** Each color represents a separate object. 48

3.11 **Comparison of triplet loss** with (DM) and without (SM) dynamic margin for the 3D output descriptor. 49

3.12 **Comparison of triplet loss** with (DM) and without (SM) dynamic margin for 32D output descriptor. 49

3.13 **Comparison of four different background noise modalities** without any real data used for training. 50

3.14 **Comparison of three modalities** representing different input image channels used in training. 51

3.15	Feature space comparison. By using a multi-task learning framework, we are able to improve feature descriptors learned for object pose estimation. Depicted here is the feature visualization using left: PCA and right: t-SNE [2] for five objects of the LineMOD [3] dataset.	52
3.16	Average time and median angular error of nearest neighbor pose retrieval, regression and our approach.	54
3.17	Sensitivity of λ in our loss function $L_{mtl} = (1 - \lambda)L_{reg} + \lambda L_{nn}$. Depicted is the influence of different weighting parameters on the mean angular error for regression as well as nearest neighbor pose retrieval. . . .	55
4.1	Example output of the DPOD method. Given a single RGB image, we regress its ID mask and its 2D-3D correspondences. PnP+RANSAC is then applied to estimate the final pose. The green bounding box shows the ground truth pose, while the blue one corresponds to the estimated pose. The almost perfect overlap of the bounding boxes indicates that estimations are very accurate.	58
4.2	Pipeline description. Given an input RGB image, the correspondence block, featuring an encoder-decoder neural network, regresses the object ID mask and the correspondence map. The latter one provides us with explicit 2D-3D correspondences, whereas the ID mask estimates which correspondences should be taken for each detected object. The respective 6D poses are then efficiently computed by the pose block based on PnP+RANSAC.	59
4.3	Correspondence model. Given a 3D model of interest (1), we apply a 2 channel correspondence texture (2) to it. The resulting correspondence model (3) is then used to generate GT maps and estimate poses.	60
4.4	Refinement architecture. The network predicts a refined pose given an initial pose proposal. Crops of the real image and the rendering are fed into two parallel branches. The difference of the computed feature tensors is used to estimate the refined pose.	63
4.5	Qualitative results. Poses predicted with the proposed approach on (a) the LineMOD dataset and (b) the OCCLUSION dataset. Green bounding boxes correspond to ground truth poses, bounding boxes of other colors to predicted poses. For both datasets predicted poses are very close to correct poses.	65
4.6	Qualitative correspondence quality. Comparison of ground truth (left), predicted (center) UV maps and heat maps (right) of absolute errors.	70

LIST OF FIGURES

5.1 **Our pipeline for 3D object autolabeling.** Left: off-the-shelf 2D detections are fed into our Coordinate Shape Space (CSS) network to predict surface coordinates and a shape vector. We backproject the coordinates to LIDAR in the camera frustum and decode the shape vector into an object model. Then, we establish 3D-3D correspondences between the scene and model to estimate an initial affine transformation. Right: We iteratively refine the estimate via differentiable geometric and visual alignment. . . . 74

5.2 **CSS representation.** Top: Car models from the PD dataset [4]. Bottom: The same cars in the CSS representation: decoded shape vector \mathbf{z} colored with NOCS. 75

5.3 **Surface projection.** DeepSDF outputs the signed values \mathbf{s} for query locations \mathbf{x} . Normals \mathbf{n} can be computed analytically by a single backward pass. Given the signed values and normals, we project the query locations onto the object surface points \mathbf{p} . Only exterior points are visualized. . . . 76

5.4 **Oriented tangent discs** (right) represent the surface geometry more accurately than billboard ones. We reduced spatial sampling and diameters for better emphasis. 77

5.5 **Automatic annotation pipeline.** We fetch frames from the dataset and separately process each 2D detection using our CSS network and differentiable optimization procedure. Afterwards, we perform a verification to discard incorrect autolabels before saving them into our CSS label pool. Once all frames are processed, we retrain our CSS network and begin the next loop over the dataset. 79

5.6 **Synthetic PD dataset.** (a) Cars from the PD dataset that were used to train our DeepSDF shape space. (b) Top: Random RGB frame. Bottom: Patches used for CSS training. 80

5.7 **Qualitative results of our labeling pipeline.** We mark the ground truth cuboids in red and our predictions in blue. We can achieve rather tight fits that lead to cuboids of slightly different sizes compared with the ground truth. 81

5.8 Data input modalities: (a) input RGB image, (b) rendered normal map, (c) rendered NOCS. Light module outputs: (d, e, f). 82

5.9 **NOCS prediction quality** of our network over consecutive loops for the same patch. Initially, the predictions are rather noisy because of the synthetic domain gap. Within each subsequent autolabeling loop the predictions become more accurate overall. 83

5.10 **Detections from the autolabel-trained detectors.** We draw local 3D frames to identify correct orientation. 85

6.1 **HomebrewedDB scene examples.** Our dataset features 13 RGB-D annotated scenes of various difficulty. The reconstructed 3D models of the objects are rendered on top of RGB images with obtained ground truth poses. 88

6.2 **Rendered reconstructed 3D models** of HomebrewedDB. 91

6.3	Sample RGB images from the sequences presented in the HomebrewedDB dataset. Complexity of the scenes varies in terms of number and size of objects, levels of occlusion and clutter.	92
6.4	Sample RGB images from sequences belonging to the domain adaptation benchmark. From left to right: original sequence, illumination benchmark sequence, texture change benchmark sequence.	93
6.5	Realism gap. Top: synthetic textureless models. Middle: synthetic textured models. Bottom: real images. Networks trained on one type of data significantly underperform when tested on another domain due to a large visual discrepancy.	103
7.1	Real \rightarrow synthetic mapping. Trained on augmented data from 3D models, our network G can map real scans (either depth or RGB) to the synthetic domain (either depth or surface normals). The pre-processed data can then be handed to various recognition methods (T^S) to improve their performance.	105
7.2	Training of the depth-processing network G_{DEP}. Following the conditional GAN architecture, a generator G_{DEP} is trained against a discriminator D to recover the original noiseless image from a randomly augmented, synthetic one. Its loss combines similarity losses \mathcal{L}_g and \mathcal{L}_f , the conditional discriminator loss \mathcal{L}_d , and optionally a feature-similarity loss \mathcal{L}_t if a task-specific method T^s is provided at training time.	108
7.3	Training of the RGB multi-modal network G_{RGB}. Taking full advantage of available synthetic data e.g., texture-less CAD models, G consists of a custom multi-modal pipeline with self-attentive distillation, trained to recover noiseless geometrical and semantic modalities from randomly augmented synthetic samples.	110
7.4	G_{DEP} augmentation, training, and testing results: (A) Depth augmentation examples (BG background noise; FG foreground distortion; OC occlusions) for different noise amplitudes or types; (B) Validation results , showing how our solution learns during training to recover the clean images (<i>input</i>) from their augmented versions (<i>augm.</i>); (C) Test results on real data (compared to ground-truth GT).	111
7.5	G_{RGB} augmentation and training results. On the <i>left</i> , we demonstrate how normal maps are step by step transformed into complex, random color images by our online augmentation pipeline. On the <i>right</i> , we present how G_{RGB} is trained on these images, learning to map them back to the noiseless geometrical information.	112
7.6	G_{RGB} qualitative results. Intermediary and final mappings when applying G_{RGB} trained on purely synthetic data to real samples, on LineMOD [3] and T-LESS [5] datasets.	113

LIST OF FIGURES

7.7 **Qualitative results of depth domain adaptation GANs [6, 7, 8] on LineMOD [3].** First row contains indicative real images from the target domain; second row contains the synthetic depth images provided as sources; followed by the corresponding GANs outputs below. 117

8.1 **Training pipeline.** Training is performed in two alternating phases. **Phase 1:** The weights of the deception network are updated, while those of the recognition network are frozen. The recognition network’s objective is maximized instead of being minimized, forcing the deception network to produce increasingly confusing images. **Phase 2:** The generated deceptive images provided by the deception network, whose weights are now frozen, are passed to the recognition network and its weights are updated such that the loss is minimized. As a result of this min-max optimization, the input images are automatically altered by the deception network, forcing the recognition network to be robust to these domain changes. . . 126

8.2 **Architecture of the deception networks used for the presented experiments.** For the case of **MNIST** classification, three deception modules are used: the *distortion module* applying elastic deformations on the image, the *BG/FG module* responsible for generating background and foreground colors, and the *noise module* additionally distorting the image by applying slight noise. The **LineMOD** dataset requires a more sophisticated treatment and features four deception modules: noise and distortion (applied on depth channel only), modules similar to the previous case, pixel-wise BG module and light module generating different illumination conditions based on the Phong model. 127

8.3 **Deceptive images x^d over consecutive iterations.** The output becomes increasingly more complex for T 128

8.4 **Example samples of the MNIST modalities.** MNIST (Source), MNIST-M (Target), and MNIST-COCO (Generalization) on the left; and example augmentation images generated by PixelDA and our method respectively. 129

8.5 **Example samples of the LineMOD modalities.** Synthetic (Source), Real (Target), and Extended (Generalization) on the left; and example augmentation images generated by PixelDA and our method respectively. 130

8.6 **Deceptive augmentations.** Augmentations applied for the SYNTIA \rightarrow Cityscapes scenario. 137

List of Tables

3.1	Test setups. Each underlined entry represents the tested parameter for a given test.	47
3.2	Comparison of the network trained without in-plane rotations (baseline) with the one trained using in-plane rotations (baseline+).	48
3.3	Angular error histogram computed using the samples of the BigBIRD test set for a single nearest neighbor.	52
3.4	Angular error of the baseline method (<i>NN</i>), regression (<i>R</i>) and our approach (<i>Rmt</i> , <i>NNmt</i>).	53
3.5	Comparison between the classification and angular accuracy of the baseline method, <i>NN</i> , and our results on 15 objects of the LineMOD dataset.	54
4.1	Pose estimation performance. Comparison of our approach to the other RGB detectors on the LineMOD dataset. The table reports the percentages of correctly estimated poses w.r.t. the ADD score. Among the methods trained on synthetic data, our method shows the best results significantly surpassing the former state-of-the-art. The variant of our method trained on real data again demonstrates outstanding performance outperforming most of the competitors. Moreover, our new refinement pipeline improves the estimated poses even further and shows the best overall results.	66
4.2	Pose estimation for multiple objects. Comparison of our approach on real data to the other RGB detectors on the OCCLUSION dataset. The table reports percentages of correctly estimated poses w.r.t. the ADD score.	67
4.3	Detection performance for multiple objects. Comparison of the state-of-the-art mean average precision (mAP) scores on the OCCLUSION dataset.	67
4.4	RANSAC iterations test. The effect of the number of RANSAC iterations on the overall ADD score.	68
4.5	Runtime comparison. Time-efficiency of our approach with respect to the other state-of-the-art approaches.	68
4.6	Runtime analysis. Runtime of the proposed approach for all models of the LineMOD dataset.	69

LIST OF TABLES

4.7 **Quantative correspondence quality.** Correspondence quality for real and synthetic data estimated in terms of mean and median absolute errors, and standard deviation. 70

4.8 **Comparison of deep learning-based refinement methods:** Our refinement approach shows the overall best ADD score with respect to the latest state-of-the art method DeepIM [9]. 71

5.1 **Cuboid autolabel quality** when inputting into the CSS network (a) 2D ground truth boxes, (b) RCNN detections, and (c) Mask-RCNN detections. We run two self-improving loops to slowly incorporate more labels into the pool. 83

5.2 **The performance comparison** of the 3D object detectors trained on the true KITTI labels vs. our autolabels. Concerning the BEV metric, the detectors trained on autolabels alone achieve the results equal to the current state of the art. In the case of the 3D AP metric, the competitive results are achieved in both considered variants at the IoU 0.5 threshold. . 84

5.3 **Ablation study** over each optimization variable and each separate loss. . 86

6.1 **Differences between the depth of object rendered models at the ground truth poses and the captured depth (in mm).** μ_δ and σ_δ is the mean and the standard deviation of the differences, $\mu_{|\delta|}$ and $med_{|\delta|}$ is the mean and the median of the absolute differences. 94

6.2 **Scalability benchmark.** Results of object detection and pose estimation. 96

6.3 **Result of object detection and pose estimation presented on two benchmarks:** (1) per scene benchmark spanning over 11 scenes of the dataset and the (2) domain adaptation benchmark evaluating the detector’s generalization capabilities. 97

6.4 **Pose estimation results** in terms of ADD 10% metric on LineMOD sequences (LM) and HomebrewedDB (HB) sequence with the same objects. 98

7.1 **Quantitative results on different tasks and datasets:** (A) Classification accuracy of different instances of the IC network over a subset of T-LESS (5 objects); (B) Classification and angular accuracy of different instances of the ICPE method over LineMOD (15 objects). Instances were trained on various data modalities (noiseless synthetic for T^S ; synthetic augmented for T^A ; or real for T^R) and tested on the real datasets X_{test}^R with different pre-processing (none; pre-processing by G_{DEP}^A trained on synthetic augmented data; or by G_{DEP}^R same method trained using real data). 118

7.2	Comparison to opposite domain-adaptation GANs. Given the two recognition tasks “(A) Instance Classification on T-LESS (5 objects)” and “(B) Instance Classification and Pose Estimation on LineMOD (15 objects)” (defined for the experiment in Table 7.1), we train several modalities of the networks T against diverse domain-adaptation GANs trained on real data X_{train}^R (50% of the datasets), and compare their final accuracy with our results.	119
7.3	Quantitative ablation study. Given the two recognition tasks “(A) Instance Classification on T-LESS (5 objects)” and “(B) Instance Classification and Pose Estimation on LineMOD (15 objects)”, we train both networks on noiseless data and evaluate them on the outputs of different modalities of G_{DEP}^A . Each is either trained (A) with different augmentation combinations (BG background noise; FG foreground distortion; OC occlusions; SI sensor simulation); or (B) with different loss combinations ($\mathcal{L}_d + \mathcal{L}_g$ vanilla GAN loss; \mathcal{L}_f foreground-similarity loss; \mathcal{L}_t task-specific loss).	120
7.4	Quantitative comparison of recognition pipelines , depending on the available training data, for the task of localized instance classification on T-LESS [5] (11 objects).	121
7.5	Quantitative comparison of recognition pipelines , depending on the available training data, for the task of localized instance classification and pose estimation (ICPE) on LineMOD [3] with method T from Chapter 3.122	122
7.6	Architectural ablation study , with the “ICPE on LineMOD” task.	123
8.1	Baseline tests. While performing slightly worse than the leading state-of-the-art domain adaptation methods using target data, we still manage to achieve very competitive performance without access to target data.	132
8.2	Generalization tests. Our method generalizes well to the extended datasets, while the adaptation methods underperform due to overfitting.	133
8.3	Module ablation. Evaluation of the importance of the deception network’s modules. BG – background, NS – noise, DS – distortion, L – light.	134
8.4	Input modality ablation. Performance evaluation based on the input data type used: depth, RGB, or RGB-D.	135
8.5	Real-world application. Segmentation performance on SYNTHIA \rightarrow Cityscapes benchmark based on Intersection over Union (IoU) tested on 16 (mIoU) and 13 (mIoU*) classes of the Cityscapes dataset. Our method outperforms <i>source</i> and <i>unguided</i> by a significant margin and remains competitive to the methods relying on the target data.	136

1 Introduction

Vision is one of the most important human senses. It allows us to reason about the world, interact with objects and other people, and it supervises almost every activity we do in our lives. Transferring vision to computers would potentially allow to perform this manifold of tasks without any human intervention. This is exactly what computer vision aims to do and why it is so important.

In recent years, computer vision has made its way into many businesses, spanning such fields as robotics, augmented reality, autonomous driving, healthcare, etc. This success was made possible largely due to the occurrence and rapid development of deep learning – the field of machine learning devoted to models based on artificial neural networks. However, despite the remarkable progress, the design of the modern deep learning-based computer vision pipelines still leaves many challenges unresolved, coming either from the high performance expectations in highly dynamic real-world environments or from the innate limitations of the deep learning systems.

1.1 Challenges of Industrial Computer Vision

First considered to be trivial, computer vision stays largely unresolved up until this very day and is proved to be one of the major driving forces and main applications for dozens of algorithms. But what makes it so difficult? People can very easily reason about the surrounding environment, which includes recognizing objects, understanding their semantic meaning and relationship to the neighbouring objects, approximating their size and location with respect to the eyes. For example, we can easily position and recognize a human's face under an astonishing number of variations in expression, viewpoint, illumination, and even under heavy occlusions. We also seem to have a capacity to remember an unlimited number of faces and only need to see it once to remember. This stellar performance is a merit of the human brain and, unfortunately, much is still unknown about how human vision works and how the brain perceives the visual contribution. Computers, on the other hand, only see images as arrays of pixels ultimately representing different colors. Given a large collection of numerical values, a computer has to understand and reason about the contents and there is no straightforward way to do it.

Early computer vision solutions were based on the hand-crafted feature extraction. This process allowed to dramatically decrease the solution space by extracting a more meaningful condensed information from the image instead of relying on raw pixels. A typical workflow would comprise the following stages: key-point detection, descriptor generation, and classification. Over the years feature detectors became increasingly more complicated and versatile (SIFT, SURF, HOG, BRIEF, etc.), so did the classifiers



Figure 1.1: 6D pose estimation on a tablet. Our RGB dense pose object detector (DPOD) from Chapter 4 estimates 6D pose of the object in real time running on a tablet.

(kNN, AdaBoost, SVM, Random Forest, etc.). However, it soon became evident that hand-crafted features pose a significant bottleneck. Since they are hand-crafted, there is no guarantee that we extract the most useful information about the object and the information loss results in a potential performance loss. On the other hand, designing ever new feature descriptors for new problems deemed infeasible. Thankfully, everything changed with the emergence of deep learning.

Current state-of-the-art computer vision algorithms are based on deep neural networks. They almost completely replaced traditional pipelines utilizing hand-crafted features. Instead, deep neural networks automatically generate task-specific features through training over large datasets of annotated data. The process of training aims to find the features that are best for the problem at hand and, as a result, making them highly discriminative. Deep neural networks allow to reach performance not even remotely imaginable before and in some cases neural nets already beat human performance. This success allowed computer vision to largely expand its application field and industrial significance.

Pose Estimation and Object Recognition Importance

Some of the most industry relevant computer vision tasks are pose estimation and object recognition since in combination they allow not only understand what kind of object we see, but also its exact location with respect to the camera (see Fig. 1.1 and Fig. 1.2). For example, it is essential in robot grasping where one needs to know the object's shape and pose to decide the best way to grasp it. In autonomous driving applications it is

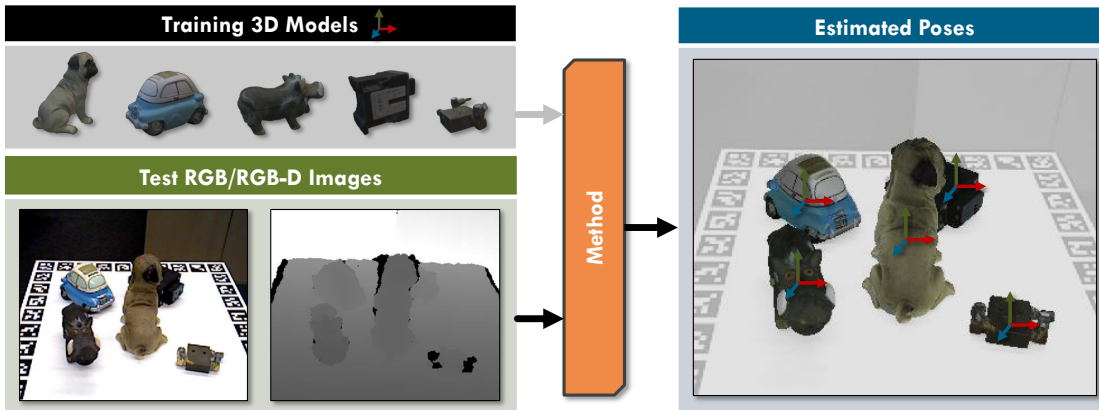


Figure 1.2: 3D object pose estimation. Given a collection of 3D object models, the goal is to estimate the pose for each of the objects of interest in RGB / RGB-D image.

required to know very precise positions of the objects surrounding the car due to strict safety constraints. Various surgical workflows track the positions of medical tools as well as human organs to help surgeons in analyzing and performing treatments. All these and many other applications rely on the quality of object and pose recognition and might fail completely in case of a poor estimate.

While the task of 2D object recognition is rather well-developed and scales well with respect to the number of objects, it is not the case when the object's pose comes into play. Pose estimation assumes the prior knowledge of the object appearance and requires more sophisticated techniques to get reliable estimates. This is especially true in the case of the single shot RGB-based pose estimation, due to perspective ambiguities, as opposed to the methods relying on depth data. However, while RGB methods are much more ill-posed and challenging, they also become more and more popular due to the ubiquitous availability of RGB cameras. Therefore, the success of RGB-based methods would lead to an easy and cheap deployment, which is of great interest not only for researchers and enthusiasts, but also for many big industry players.

Domain Gap Problem

The main power and also the main weakness of the neural network-based algorithms are the data. In practice, and especially in the industry, access to real physical objects is very limited (e.g., one cannot capture new image datasets for every new client, product, part, environment, etc.), but 3D CAD models are widely available. It thus became common to leverage such data to train recognition methods e.g., by rendering large datasets of relevant synthetic images and their annotations. However, such approaches are subject to a severe problem of domain gap, in which synthetic images are very different from those one gets from commodity RGB cameras (see Fig. 1.3). As a result, models trained on synthetic data perform poorly on real images or do not work at all.

Bridging the gap between real and synthetic data is the task of the domain adaptation field, which has become crucial for deep learning methods and is another major topic



Figure 1.3: Domain gap. Synthetic data are very different from real images coming from commodity sensors not only due to the sensor-specific properties, but also due to external illumination and object appearance changes. The problem becomes even more difficult when objects have no texture.

of this thesis. The difficulty of the domain adaptation task largely depends on the discrepancy between the source and target domains. While in the case of the depth data it mainly arises from the imperfections of the depth sensor itself, the RGB domain holds multiple other challenges. The formation of the RGB image is not only affected by the internal sensor noise and parameters, but also by the perceptive ambiguities and various external changes. For example, the same very object can look extremely different depending on the time of the day, the light sources surrounding it, and the camera the shot is taken from. Moreover, objects can also change their texture with time through wearing and other numerous reasons. The change in appearance results in an image containing completely different pixel values for the same object. Making it even more difficult, it is often the case that CAD models are not textured leaving no color prior to even start from. This scenario requires methods that either use depth data or are invariant to color.

1.2 Contributions

In this work, we concentrate on the above-mentioned problems of object pose estimation as well as domain adaptation and propose solutions that achieve state-of-the-art results on different tasks. The main contributions of this thesis can be summarized as follows:

- **Scalable object recognition and 3D pose estimation based on manifold learning** Scalability with respect to the number of objects is a critical issue for methods that not only classify objects, but also estimate their pose. We propose a solution based on manifold learning, where images are mapped to a highly-discriminative lower-dimensional manifold space. Once mapped, this allows to efficiently retrieve both object’s class and pose. Moreover, we extend this solution by combining it with the direct pose regression making it end-to-end and show how this multi-task pipeline improves the discriminative power of the method.

- Real-time RGB 6D pose estimation pipeline based on dense correspondences** While object detection and 6D pose estimation is much easier to solve in depth images, RGB solutions are much more desirable due to the widespread availability of RGB cameras. However, perspective ambiguities and significant appearance changes of the object when seen from different viewpoints make RGB-based methods rather rare and unreliable. We propose a solution based on dense correspondences as opposed to former state of the art either regressing the pose directly or relying on a coarse number of correspondences. While being very precise compared to the former state-of-the-art solutions, the presented approach is also real-time capable and can be trained entirely on synthetic CAD models. Additionally, we present a deep learning-based pose refinement pipeline that further improves the performance of the method.
- 9D pose and shape estimation pipeline using SDF shape priors and differentiable SDF rendering** RGB-based 6D pose estimation pipelines assume the prior knowledge (i.e., dimensionality and scale) about the objects of interest. However, when we deal with entire classes of objects this assumption becomes infeasible. To tackle this issue, we propose a solution based on SDF (signed distance field) shape priors covering possible shape variations within classes. Furthermore, we develop an optimization pipeline utilizing a novel differentiable SDF renderer. The presented solution reliably estimates 9D object poses and their shapes allowing for automatic labeling of the data. In turn, this allows to train downstream methods achieving state-of-the-art results without any ground truth labels available for training.
- RGB-D 6D Pose Estimation Dataset** We present a dataset for 6D pose estimation targeting training from 3D models (both textured and textureless), scalability, occlusions, and changes in light conditions and object appearance. The dataset features 33 objects (17 toy, 8 household, and 8 industry-relevant objects) over 13 scenes of various difficulty. We also present a set of benchmarks to test various desired detector properties, particularly focusing on scalability with respect to the number of objects, and resistance to changing light conditions, occlusions and clutter.
- Reverse domain adaptation method to map unseen real data into synthetic domains** Generally, domain adaptation methods aim to make synthetically generated data look more realistic. Instead, we propose a novel approach tackling this problem from the opposite angle. Purely trained on synthetic data, playing against an extensive augmentation pipeline in an unsupervised manner, the proposed approach effectively segments images and recovers clean synthetic-looking information even from partial occlusions. As a result, this significantly simplifies training and consistently enhances the performance of the downstream task.
- Network-driven domain randomization** As an alternative to vanilla domain randomization methods, we present a novel adversarial procedure to tackle domain

adaptation between synthetic and real data. In particular, the presented solution uses the downstream task network as an adversarial guide toward useful augmentations that maximize the uncertainty of the output. Unlike GAN-based approaches that require unlabeled data from the target domain, this method achieves robust mappings that scale well to multiple target distributions from source data alone. The method was tested on multiple benchmarks and compared to a number of domain adaptation approaches, thereby demonstrating similar results with superior generalization capabilities.

1.3 Outline

This section provides an overview of the subsequent thesis chapters. All the described methods have been published at major peer-reviewed conferences. We therefore provide a reference for each related work and encourage the reader to consult online materials for video demonstrations.

Chapter 2 – Background First, we cover the fundamentals important to grasp the main concepts that further chapters are based on. In particular, we briefly discuss various topics from the fields of deep learning and projective geometry, and also provide an overview of relevant pose estimation and domain adaptation methods.

Part I: Pose Estimation

Chapter 3 – 3D Pose Estimation Based on Manifold Learning Here, we present a 3D pose estimation and object recognition method based on mapping images to a highly discriminative lower-dimensional space. A novel dynamic margin loss function allows to improve the discriminative power of the underlying manifold space, which results in significantly better estimates and improved scalability properties. To further improve performance, we propose to combine manifold learning with direct pose regression. The resulting multi-task learning pipeline not only shows a mutual improvement on both tasks, but also demonstrates how to make an end-to-end pipeline benefiting from the underlying manifold structure.

- **S. Zakharov**, W. Kehl, B. Planche, A. Hutter, S. Ilic. *3D Object Instance Recognition and Pose Estimation Using Triplet Loss with Dynamic Margin*. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2017
- M. Bui, **S. Zakharov**, S. Albarqouni, S. Ilic, N. Navab. *When Regression meets Manifold Learning for Object Recognition and Pose Estimation*. IEEE International Conference on Robotics and Automation (ICRA), 2018

Chapter 4 – 6D Pose Estimation Based on Dense Correspondences This chapter aims to solve a more difficult problem of object detection and 6D pose estimation

from RGB images. The proposed solution uses dense correspondences, as opposed to the former state-of-the-art methods. As a result, we get a robust real-time pipeline that can be trained entirely on synthetic data. Moreover, this chapter also presents a refinement approach improving the predicted estimates even further.

- **S. Zakharov***, I. Shugurov*, S. Ilic. *DPOD: 6D Pose Object Detector and Refiner*. IEEE International Conference on Computer Vision (ICCV), 2019 (*equal contribution)

Chapter 5 – 9D Pose Estimation for Autolabeling The lack of information about the detected object adds additional degrees of freedom to the problem of pose estimation. Additionally to rotation and translation, one has to determine the shape and scale of the object. The pipeline presented in this chapter does exactly this. To make this possible we use SDF shape priors and introduce a novel differentiable SDF renderer. The resulting solution can be used to automatically label datasets.

- **S. Zakharov***, W. Kehl*, A. Bhargava, A. Gaidon. *Autolabeling 3D Objects with Differentiable Rendering of SDF Shape Priors*. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2020 (*equal contribution)

Chapter 6 – RGB-D 6D Pose Estimation Dataset In this chapter, we present a dataset for 6D pose estimation targeting training from synthetic CAD models and such challenges as scalability, robustness to occlusions, illumination and appearance changes. We present benchmarks for each of the challenges to promote the field of 6D pose estimation and raise the importance of the domain adaptation problem.

- R. Kaskman, **S. Zakharov**, I. Shugurov, S. Ilic. *HomebrewedDB: RGB-D Dataset for 6D Pose Estimation of 3D Objects*. IEEE International Conference on Computer Vision (ICCV) Workshops, 2019

Part II: Domain Adaptation

Chapter 7 – Reverse Domain Adaptation In this chapter, we present a novel reverse domain adaptation idea that denoises, declutters, and maps previously unseen images to synthetic domains. This solution not only improves the performance of the downstream tasks, but also makes the training process much more practical by domain adaptation uncoupling.

- **S. Zakharov***, B. Planche*, Z. Wu, A. Hutter, H. Kosch, S. Ilic. *Keep it Unreal: Bridging the Realism Gap for 2.5D Recognition with Geometry Priors Only*. International Conference on 3DVision (3DV), 2018 (*equal contribution)
- B. Planche*, **S. Zakharov***, Z. Wu, A. Hutter, H. Kosch, S. Ilic. *Seeing Beyond Appearance - Mapping Real Images into Geometrical Domains for Unsupervised CAD-based Recognition*. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2019 (*equal contribution)

Chapter 8 – Network-Driven Domain Randomization This chapter presents a novel domain randomization technique. Instead of using a regular randomization pipeline i.e., randomly varying backgrounds, changing brightness and contrast, etc., we design differentiable augmentation modules and utilize the task network to serve as an augmentation guide. Experiments on various datasets demonstrate competitive performance while having superior generalization capabilities.

- **S. Zakharov**, W. Kehl, S. Ilic. *DeceptionNet: Network-Driven Domain Randomization*. IEEE International Conference on Computer Vision (ICCV), 2019

Chapter 9 – Conclusion and Outlook This chapter concludes the thesis, summarizes our findings, and discusses the directions for future work.

2 Background

This chapter gives an overview of the theoretical background required for the further chapters. We first give a basic introduction to neural networks and the field of deep learning. Then we cover relevant topics of projective geometry, and provide an overview of related works on pose estimation and domain adaptation.

2.1 Neural Networks

Humans and higher animals identify, make decisions, and learn at every step of their lives. The neural network approach arose from a desire to understand how the brain solves such complex problems and to implement these principles in automatic devices.

Artificial neural networks or ANNs are very simplified analogues of the natural neural networks. The nervous system of animals and humans is much more complex than what people can currently achieve with the help of modern technology. However, it appears that for the successful solution of many practical problems it is enough to know the general principles of how the nervous system functions. Most of the ANNs represent mathematical models that have only a distant resemblance with neurophysiology. This however does not preclude their practical application in various fields, e.g., image classification, segmentation and natural language processing.

The purpose of this section is to give a quick introduction to neural networks, including the main concepts and terminology, which are essential for understanding the further chapters. It is mainly based on the material from [10, 11, 12, 13, 14, 15].

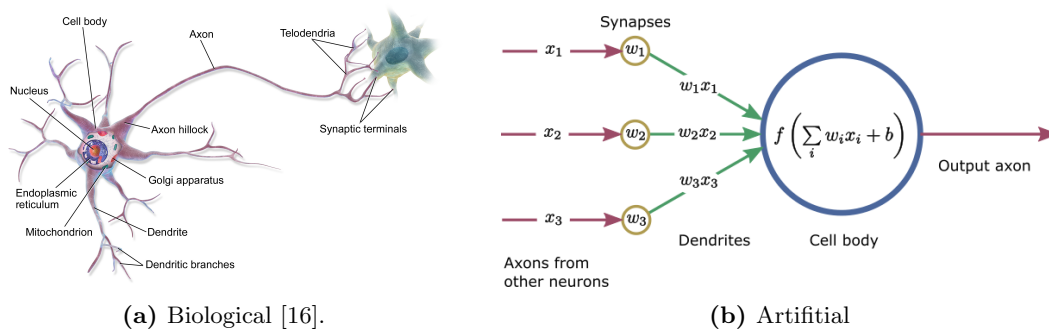


Figure 2.1: Neuron representations: Artificial neuron model is a very rough approximation of the biological model.

Artificial Neuron

First, let us consider the general working principles of a nerve cell or neuron (Fig. 2.1a). A neuron consists of a cell body, dendrites and an axon. Dendrites are tree-like thin structures that extend away from the cell body. An axon is a long tube-like structure that sends signals to other neurons through a synaptic terminal, which is usually connected to a dendrite of another cell. Being in the excited state a cell generates an electrical impulse of about $100mV$ and a duration of $1ms$, which runs along the axon to the synapse. On the arrival of the signal, the synapse triggers the migration of vesicles containing neurotransmitters through which the signal is transferred. If the total charge received by a cell exceeds a certain threshold, the cell becomes excited and fires an impulse that propagates along the axon and reaches the synapses, thereby contributing to excitation of other cells.

A mathematical representation of a biological neuron is usually presented by a simple model of the perceptron, see Fig. 2.1b. It was developed by the scientist Frank Rosenblatt in the late 1950s and is based on the McCulloch-Pitts Threshold Logic Unit (TLU) model. As shown in Fig. 2.1b, each input value coming from the left is multiplied by a specific weight, which is a simulation of the synaptic strength. Then all the weighted inputs are added together to get a total signal strength. Finally, an activation function is applied on the sum; in case of a perceptron it is a simple step function (see Fig. 2.3a). If the sum is greater than a threshold value, also called bias, then the output is set to 1, otherwise it is set to 0. Although the perceptron is a very rough model of a biological neuron, it can perform a substantial level of computation.

2.1.1 Fully Connected Neural Networks

The true computing power of an artificial neural network comes from connecting multiple neurons. Neurons are usually connected into a network by layers. Therefore, the simplest possible network structure is a single-layer network shown in Fig. 2.2a, where every node represents a neuron and lines can be thought of as axon-dendrite connections.

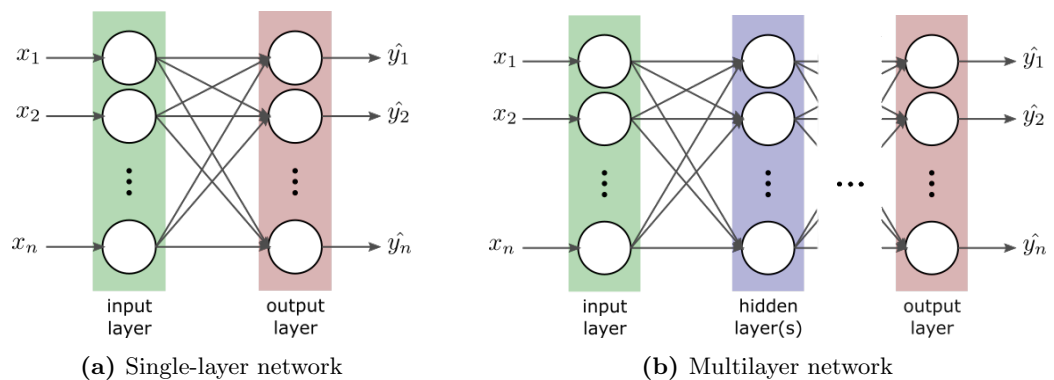


Figure 2.2: Network structures. Going from a single to multiple layers.

As with every neural network, it contains the input and output layers of neurons. The input layer on the left is simply the representation of our input data and it is not counted when specifying the network depth. The data from the input layer is connected to each artificial neuron in the output layer. In neural network terminology a layer in which neurons are fully pairwise connected to the neurons of the previous layer is called a fully-connected layer. Since each output neuron receives the same set of data, the results are varied based on the values of weights and biases. The output results usually represent the class scores or real values depending on the nature of the problem, i.e., classification or regression. A single-layer network can easily be extended to a multilayer or a deep network (giving rise to the term deep learning) by adding more layers (see Fig. 2.2b). This more complex structure helps to achieve a higher level of computational capability. Any layer between the input and output layers is called a hidden layer.

The network structure on itself is not very exciting if we can not train it to adapt to specific tasks. The training of a network consisting of a group of neurons is realized by slightly changing the weights and biases to get the best results or, in other words, such that the output error is minimized.

Activation Functions

The perceptron neuron model uses a step activation function (Fig. 2.3a). However, in modern artificial neural networks other types of neurons are used. The main reason behind not using the step activation function is that it is not smooth. However, the training of a network requires the calculation of the activation function's derivatives and therefore it has to be differentiable or smooth.

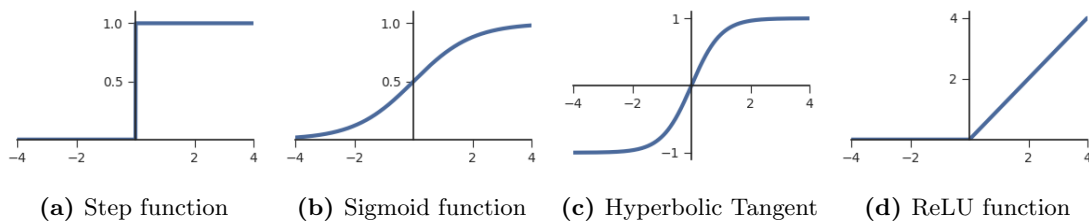


Figure 2.3: Activation functions. Step function and its smooth alternatives.

Non-linear activation functions allow neural networks to model complex patterns that simpler models may miss. The most common types of the activation functions used in practice are: sigmoid (Fig. 2.3b), hyperbolic tangent (Fig. 2.3c), and rectified linear unit (Fig. 2.3d). Their mathematical forms are defined as follows.

$$f_{sigmoid}(x) = \frac{1}{1 + e^{-x}}, \quad f_{tanh}(x) = \frac{2}{1 + e^{-2x}} - 1, \quad f_{ReLU}(x) = \begin{cases} 0 & : x < 0 \\ x & : x \geq 0 \end{cases} \quad (2.1)$$

Forward Propagation

The process of passing the input data through the network is called the forward propagation. In order to demonstrate it, let us introduce a simple 2-layer network shown in Fig. 2.4. The bias terms are going to be ignored for simplicity.

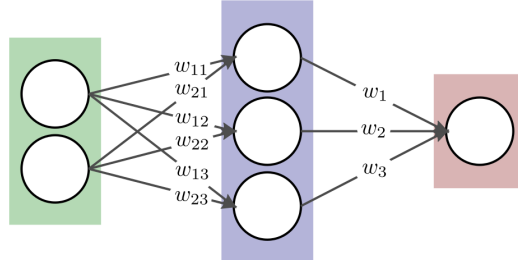


Figure 2.4: Sample neural network to illustrate the forward and backward propagation processes.

The first step is to calculate the output for each neuron of hidden layer just as it was shown for a perceptron in Fig. 2.1b: all the input values are multiplied by the respective weights and summed together, then an activation function is applied on the sum. This can be effectively computed using the matrix representation. Using input data matrix X and weight matrices $W^{(1)}$ we first compute activations $Z^{(1)}$ of the hidden layer:

$$\begin{bmatrix} i_{11} & i_{12} \\ i_{21} & i_{22} \end{bmatrix} \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} = \begin{bmatrix} i_{11}w_{11} + i_{12}w_{21} & i_{11}w_{12} + i_{12}w_{22} & i_{11}w_{13} + i_{12}w_{23} \\ i_{21}w_{11} + i_{22}w_{21} & i_{21}w_{12} + i_{22}w_{22} & i_{21}w_{13} + i_{22}w_{23} \end{bmatrix} \quad (2.2)$$

$X \qquad W^{(1)} \qquad Z^{(1)}$

To calculate the final neurons' outputs the activation function is applied to matrix $Z^{(1)}$ element-wise:

$$A^{(1)} = f(Z^{(1)}) \quad (2.3)$$

In the second step, we compute the activations of the output layer similarly to first step with the only difference being the input values:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} a_{11}w_1 + a_{12}w_2 + a_{13}w_3 \\ a_{21}w_1 + a_{22}w_2 + a_{23}w_3 \end{bmatrix} \quad (2.4)$$

$A^{(1)} \qquad W^{(2)} \qquad Z^{(2)}$

The final result matrix \hat{Y} is achieved by applying the activation function to the elements of $Z^{(2)}$:

$$\hat{Y} = f(Z^{(2)}) \quad (2.5)$$

As we can see, forward propagation is a very trivial process that can be efficiently computed using matrix representation. Once it is completed, the output votes are available and can be used to improve the network.

Gradient Descent Solvers

The network training is performed via the gradient descent algorithm. The idea of the gradient descent algorithm is to use the gradient of the cost function L with respect to the parameters in order to update them. Given the gradient at a certain point we can say which direction we have to go to minimize or maximize the cost function L by the value of the gradient. To get to the minimum from a randomly initialized point, we have to move in the direction of a negative gradient with a predefined step size. The update rule can therefore be formulated as follows:

$$W_{t+1} = W_t - \alpha \nabla L(W_t), \quad (2.6)$$

where $\nabla L(W_t)$ is the derivative of the cost function, W_t are the weights at the current iteration t , and α is the learning rate.

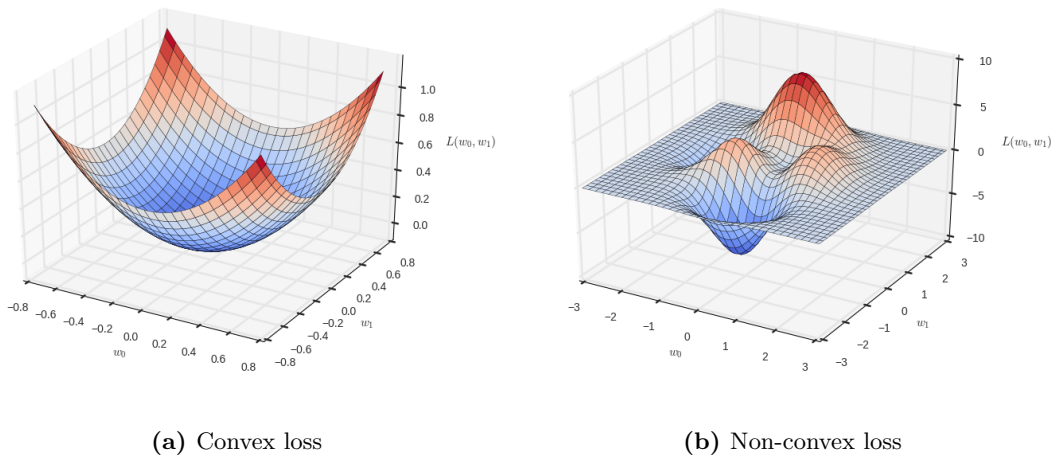


Figure 2.5: Possible loss function landscapes. In many real-world applications we have to deal with multidimensional non-convex loss functions.

Gradient descent provides no guarantees of converging to a good solution, converging to a solution in a certain amount of time, or converging to a solution at all. The Euclidean loss function has a parabolic shape (Fig. 2.5a), which has only one minimum so it can be found by the gradient descent very easily. However, it could be the case that our cost function is non-convex (Fig. 2.5b), meaning that it does not always go in the same direction. Given that, the gradient descent method may get stuck in a local minimum instead of a global one. We can move too slowly along the gradient and never reach the minimum or move too quickly and skip the minimum. Both are possible and there are various techniques that try to optimize the gradient descent method.

The most common variations of the gradient descent method are the stochastic gradient descent (SGD) with momentum and adaptive moment estimation (Adam).

SGD is a simplification of gradient descent, in which at each iteration the gradient is computed with respect to a few training examples comprising a minibatch. Despite the

2 Background

noise introduced by this simplification, SGD can lead to an improved convergence, due to the reduced variance and, moreover, it takes much less time to make an update. The notion of momentum introduces the additional velocity term influenced by the gradient and can further improve the convergence:

$$\begin{aligned}V_{t+1} &= \mu V_t - \alpha \nabla L(W_t), \\W_{t+1} &= W_t + V_{t+1},\end{aligned}\tag{2.7}$$

One of the biggest problems of SGD solver is the setting of the learning rate. Usually people reduce the learning rate every n epochs (one run through the training set), which can be set by a step decay parameter in the available neural network frameworks. Nevertheless, there is no golden rule to set the decay parameter and it heavily depends on the problem specification. This is the reason much work has gone into developing the methods that adaptively set the learning rate.

Adam or Adaptive moment estimation is one of the most popular methods that frees us from the need of setting a learning rate parameter. It defines the following update rule:

$$\begin{aligned}M_{t+1} &= \beta_1 M_t + (1 - \beta_1) \nabla L(W_t), \\V_{t+1} &= \beta_2 V_t + (1 - \beta_2) \nabla L(W_t)^2, \\W_{t+1} &= W_t - \mu \frac{\sqrt{1 - \beta_2^{t+1}}}{1 - \beta_1^{t+1}} \frac{M_{t+1}}{\sqrt{V_{t+1} + \epsilon}}.\end{aligned}\tag{2.8}$$

It uses the second order moment V , which is computed as an exponentially decaying average of historical gradients. This eliminates the problem of the learning rate inevitably converging to zero by considering only the historical gradients within a recent time window, and not within the entire history as it is defined for AdaGrad. The term M is called the first order momentum. It is used to avoid the direct usage of the gradient $\nabla L(W)$ in the update rule and is very similar to the moment rule we discussed earlier. Finally, the $\frac{\sqrt{1 - \beta_2^{t+1}}}{1 - \beta_1^{t+1}}$ term is used for bias-correction since both V and M are initialized to zero and are biased towards it.

Backward Propagation

The whole process of computing the derivatives with respect to the weights is called the backward propagation or backpropagation for short. The name comes from the fact that the process of computing the gradient of the loss function can be seen as propagating the error backwards to the weights using the chain rule.

We will continue working with the network we have defined in the forward propagation section (Fig. 2.4) and the Euclidean cost function. Our weights are distributed across two matrices, namely $W^{(1)}$ and $W^{(2)}$. It means that to compute the derivative of the cost function with respect to all the weights, we just need to do so for these two.

Let us first take the derivative with respect to the matrix $W^{(2)}$:

$$\frac{\partial L}{\partial W^{(2)}} = \frac{\partial \sum \frac{1}{2}(y - \hat{y})^2}{\partial W^{(2)}} \quad (2.9)$$

$$= \sum \frac{\partial \frac{1}{2}(y - \hat{y})^2}{\partial W^{(2)}} \quad (2.10)$$

$$= \sum -(y - \hat{y}) \frac{\partial \hat{y}}{\partial W^{(2)}} \quad (2.11)$$

$$= \sum -(y - \hat{y}) \frac{\partial \hat{y}}{\partial Z^{(2)}} \frac{\partial Z^{(2)}}{\partial W^{(2)}} \quad (2.12)$$

$$= -(A^{(1)})^T (Y - \hat{Y}) f'(Z^{(2)}). \quad (2.13)$$

Using the sum rule, we can rewrite Eq. 2.9 as Eq. 2.10. The derivative of Eq. 2.10 is computed as Eq. 2.10 using the chain rule. Then, again applying the chain rule on $\frac{\partial \hat{y}}{\partial W^{(2)}}$ in Eq. 2.11, we get $\frac{\partial \hat{y}}{\partial Z^{(2)}} \frac{\partial Z^{(2)}}{\partial W^{(2)}}$ in Eq. 2.12. The rate of change of \hat{y} with respect to $Z^{(2)}$ or $\frac{\partial \hat{y}}{\partial Z^{(2)}}$ is the same as the derivative of the activation function with respect to $Z^{(2)}$ or $f'(Z^{(2)})$. The last term $\frac{\partial Z^{(2)}}{\partial W^{(2)}}$ is simply derived as matrix $A^{(1)}$, due to the fact that $Z^{(2)} = A^{(1)}W^{(2)}$. Finally, if we replace single output examples y and \hat{y} with matrices Y and \hat{Y} , containing all the output examples, and do certain manipulations as shown in Eq. 2.12 the sum term can be avoided.

The same operation is performed for $\frac{\partial L}{\partial W^{(1)}}$:

$$\frac{\partial L}{\partial W^{(1)}} = \sum -(y - \hat{y}) \frac{\partial \hat{y}}{\partial W^{(1)}} \quad (2.14)$$

$$= \sum -(y - \hat{y}) \frac{\partial \hat{y}}{\partial Z^{(2)}} \frac{\partial Z^{(2)}}{\partial W^{(1)}} \quad (2.15)$$

$$= \sum -(y - \hat{y}) f'(Z^{(2)}) \frac{\partial Z^{(2)}}{\partial A^{(1)}} \frac{\partial A^{(1)}}{\partial W^{(1)}} \quad (2.16)$$

$$= \sum -(y - \hat{y}) f'(Z^{(2)}) (W^{(2)})^T \frac{\partial A^{(1)}}{\partial Z^{(1)}} \frac{\partial Z^{(1)}}{\partial W^{(1)}} \quad (2.17)$$

$$= -X^T (Y - \hat{Y}) f'(Z^{(2)}) (W^{(2)})^T f'(Z^{(1)}). \quad (2.18)$$

The derivation for $\frac{\partial L}{\partial W^{(1)}}$ goes in a similar manner to $\frac{\partial L}{\partial W^{(2)}}$. We again follow the chain rule until the point where all the components are known. As you can see, the components computed for $\frac{\partial L}{\partial W^{(2)}}$ are present in the $\frac{\partial L}{\partial W^{(1)}}$ derivation as well and, therefore, can be efficiently reused.

As a result, we get two matrices of derivatives $\nabla L(W^{(1)})$ and $\nabla L(W^{(2)})$ of the same sizes as $W^{(1)}$ and $W^{(2)}$. Knowing the gradients, the weights can then be easily updated using one of the update rules we discussed in Section 2.1.1. Concluding this section, the network can successfully be trained by iteratively doing forward and backward propagations and updating the parameters using one of the GD-based solvers.

2.1.2 Convolutional Neural Networks

Fully-connected networks can be used and show state-of-the-art performance for many different applications. Nevertheless, they are not very well suited for working with images. The logical approach to adapt fully-connected networks to work with images would be to treat every pixel as a separate input for a neuron in the next layer. However, in this case, the spatial structure of the image will not be taken into account. It turns out, there are better ways to deal with images.

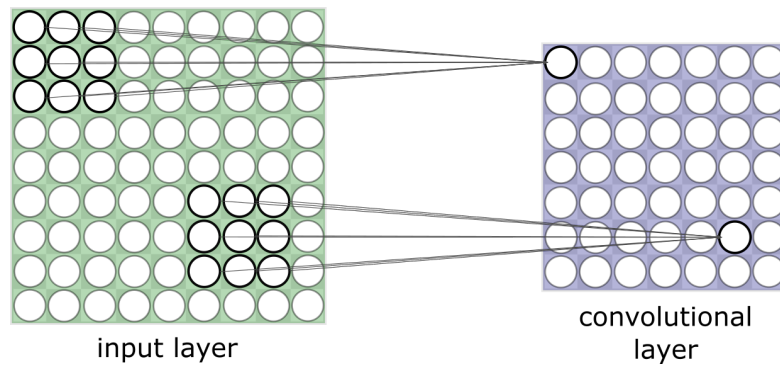


Figure 2.6: Receptive fields. Groups of pixels are considered for the next layer, as opposed to treating each pixel individually.

Convolutional neural networks, or CNNs, are the networks specifically designed to work with images and are inspired by the approaches used in image processing. They have three features that distinguish them from regular neural networks: local receptive fields, shared parameters, and pooling layers.

Local Receptive Fields The first distinctive feature of CNNs are the local receptive fields. Instead of considering each pixel as a separate input value for the next hidden convolutional layer we take groups of pixels. This means that each neuron in the convolutional layer has a group of input pixels assigned to it. The size of this group is called the local receptive field of the respective neuron.

The local receptive field is moved along the input image and each new position corresponds to a mapping to a different neuron in the hidden layer as shown in Fig. 2.6. The shift between the closest receptive field positions is defined by the stride parameter.

Parameter Sharing The second core feature of CNNs is parameter sharing. As previously stated, each neuron of a convolutional layer is connected to its own local receptive field in the previous layer. Parameter sharing says that all these neurons share the same weights and bias, and so this operation can be thought of as convolution.

Convolution is one of the most common operations used in image processing. Loosely saying, it is performed by sliding a small kernel matrix over the image and applying it at each new position, which is done by using the summation over the element-wise multiplication between the kernel and the part of the image covered by it. Therefore, at

each new position we produce a single output value, resulting in an image of the same or smaller size depending on the border treatment type. This is exactly what happens in the convolutional layer: we slide a kernel, or feature, across the image multiplying each pixel by defined weights and ultimately forming a feature map.

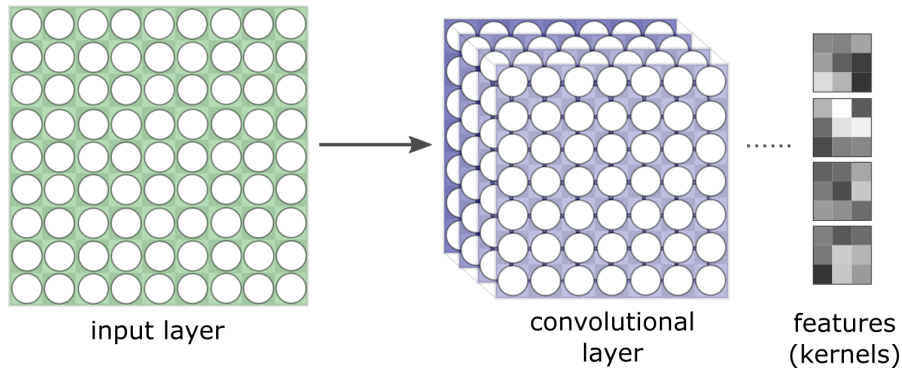


Figure 2.7: Feature maps and features. Convolutional layers consist of feature maps each defined by a set of shared weights (features) and a single shared bias.

Parameter sharing makes CNNs highly translationally invariant. A certain feature, e.g. an edge, may be found in any part of the image and detecting it is equally useful no matter what position of the image it is. Complete convolutional layers consist of several feature maps (Fig. 2.7), each defined by a set of shared weights and a single shared bias, allowing the network to detect several kinds of features across the entire image. Apart from that, this technique is particularly beneficial to the network’s performance since many fewer parameters need to be tuned compared to fully-connected networks showing a similar performance on images.

Pooling Layers Another common type of layer that is often used in CNNs is the pooling layer. It simply reduces the spatial size of the image and, therefore, decreases the number of parameters still keeping the relative spatial structure. The downsampling is performed by again sliding a kernel over the image, but with a stride the length of the kernel. For each position we output a single value based on a type of pooling. For max-pooling, which is the most popular pooling type, we always take the maximum value within the treated region. However, other variations of pooling are often used. L2 pooling for example outputs an L2 norm for each treated region.

2.1.3 Optimization

There are various methods and techniques to optimize the way neural networks work and especially prevent them from overfitting. Overfitting is one of the most common problems we have to deal with in training. The problem arises when a model adapts to the training data too well to the point where it starts learning peculiarities and the noise belonging to it. As a result, while scoring the perfect accuracy with the training data, it will not be the case for the test data.

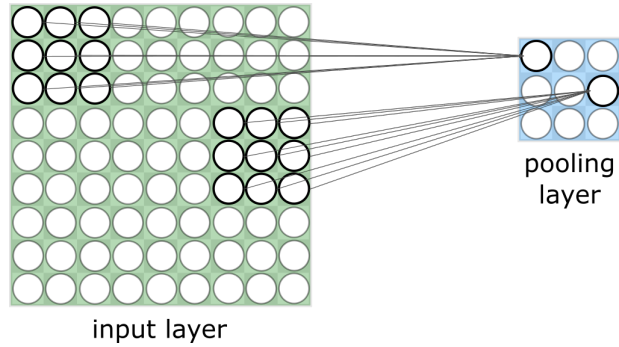


Figure 2.8: Pooling layer. Used to decrease the number of parameters while keeping the relative spatial structure.

The first logical solution to this problem is to extend the training data. While perfectly legitimate, it is not always possible to get more data. The other solution would be to decrease the number of parameters. With fewer parameters there is a lower probability of overfitting, but models with more parameters can be much more powerful in terms of their learning capabilities. However, there are methods that do not require additional data and still help a lot to deal with overfitting.

Regularization The most popular technique to deal with overfitting is regularization. It modifies the cost function by adding an additional regularization term:

$$C = C_0 + R \quad (2.19)$$

There are two common types of regularization: $L2$ and $L1$. In the case of $L2$, the regularization term R is equal to $\frac{1}{2}\lambda w^2$, where λ is the regularization parameter. By minimizing the modified cost function, we are basically finding a compromise between minimizing the original function and finding small weights. This limitation makes it more difficult for the network to learn the effects of noise in the training set since it is much harder to change its behavior by slightly changing the input data, whereas this is possible with greater weights.

For the $L1$ regularization, the regularization term R takes a form of $\lambda|w|$. This has a similar effect of penalizing the weights with a large weight factor. However, the way the weights are penalized is different, coming from the derivatives of the two terms. The $L2$ regularization shrinks the weights proportionally to their value ($\frac{\partial \frac{1}{2}\lambda w^2}{\partial w} = \lambda w$), whereas $L1$ shrinks the weights by a constant ($\frac{\partial \lambda|w|}{\partial w} = \lambda \text{sgn}(w)$). Thereby, the weight vectors become sparse, i.e. many weights are close to zero, prompting the network to use the most important inputs and ignore the noise, which allows for explicit feature selection. However, if the mentioned properties of the $L1$ regularization are not needed, people tend to use the $L2$ regularization since it usually gives better results than $L1$.

Dropout Another popular technique is called the dropout. It is very simple yet effective technique that can be used together with the $L1$ and $L2$ regularizations. Instead of modifying the cost function, the network structure is changed. The idea is to use a layer where the probability for each neuron to be omitted is set to a certain value. In this way, always different combinations of neurons are selected at each iteration (forward + backward propagation) while training. Using a dropout layer(s) can be thought of as training several different networks and averaging them. Whereas a single network may easily overfit, combining the results from several single networks can reduce the effect of overfitting.

Batch Normalization Also known as batchnorm [17], it is a very popular normalization technique used in almost every neural network. It was initially designed to tackle the vanishing gradient problem making it possible to train very deep neural networks. However, batch normalization also helps to dramatically reduce the training time and has a regularization effect which reduces the generalization error. In essence, batchnorm normalizes activations to zero mean and unit standard deviation for each neuron allowing them to learn on a more stable input distribution. Not to limit the expressive power of the network, batchnorm also allows the network to estimate these moments during training across multiple batches.

2.2 Projective Geometry

To be able to understand the contents of the camera image, one first needs to know how the image is formed. The important prerequisites for that is the knowledge of the field of projective geometry. This section presents a quick overview of the Euclidean and projective geometry topics used in this thesis. For more details please refer to Multiple View Geometry in Computer Vision, Hartley and Zisserman [1].

Homogeneous Coordinates Generally, 2D points are denoted as $\mathbf{x} = (x, y)^\top \in \mathbb{R}^2$ and 3D points as $\mathbf{X} = (X, Y, Z)^\top \in \mathbb{R}^3$. To conveniently represent transformations, the notion of homogeneous coordinates is used, where points are embedded into projective spaces having an extra dimension w for 2D and W for 3D spaces forming $\tilde{\mathbf{x}} = (x, y, z, w)^\top \in \mathbb{P}^2$ and $\tilde{\mathbf{X}} = (X, Y, Z, W)^\top \in \mathbb{P}^3$ respectively. When $W = 1$ allows to write rigid-body motions as linear transformations using simple matrix-vector multiplications as:

$$\begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix} \cdot \tilde{\mathbf{X}} = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix} \cdot \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \begin{pmatrix} \mathbf{R} \cdot \mathbf{X} + \mathbf{t} \\ 1 \end{pmatrix} \quad (2.20)$$

2.2.1 Rigid Body Transformations

Rigid body motion changes the position and orientation while preserving relative distance and angles between any pair of points. All such transformations in 3D Euclidean space

2 Background

consist of rotation and translation and form the *special Euclidean group* $SE(3)$. A single transformation can be represented in the matrix form:

$$\begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix}, \quad (2.21)$$

where $\mathbf{R} \in SO(3)$ is the orthogonal matrix representing the rotation and belonging to the *special orthogonal group* $SO(3)$ and $\mathbf{t} \in \mathbb{R}^3$ is a translation vector.

The *special orthogonal group* $SO(3)$ is the space of all 3D rotations represented by 3×3 matrices that satisfy the following properties:

$$\mathbf{R}^\top \cdot \mathbf{R} = \mathbf{R}^\top \cdot \mathbf{R} = \mathbf{I} \quad \text{and} \quad \det(\mathbf{R}) = 1 \quad (2.22)$$

These properties ensure the rigid-bodiedness of motions by preserving the local orientation and length. While this parametrization is very practical because it allows to transform points in linear form by matrix-vector multiplication, it suffers from over-parametrization. Despite representing only 3 degrees of freedom (3DoF), such matrix consists of 9 values, which makes a possible optimization problematic. This is the reason of appearance of various other rotational representations, e.g., Euler angles, axis angle, unit quaternions, etc.

Euler Angles One of the most simple representations are Euler angles α, β, γ , which represent the rotations around the canonical axes which can be combined to represent any possible rotation. While being the most compact, this representation has significant drawbacks. First of all it is not unique, i.e., different angles can result in the same final rotation. Second, it suffers from the infamous gimbal lock problem, which leads to the loss of degrees for certain rotation application orders.

Axis-Angle According to Euler's rotation theorem, all 3D rotations have an axis-angle representation. In other words, every orientation of a rigid body in space can be described by a single rotation θ about some axis \hat{r} through the origin. This observation allows to get away from the ambiguous Euler angle representation. However, while being very intuitive, one cannot directly combine two rotations to give an equivalent total rotation when using axis-angle representation. The easy solution is to either convert it back to a matrix representation or to use quaternions.

Unit Quaternions Another well-established and often used alternative are unit quaternions. Unit quaternions are generalization of complex numbers represented as vectors of 4 elements $\mathbf{q} = (q_w, q_x, q_y, q_z)^\top$, with

$$\|\mathbf{q}\| = \sqrt{q_w^2, q_x^2, q_y^2, q_z^2} = 1, \quad \text{and} \quad \mathbf{q}^{-1} = \frac{\mathbf{q}}{\|\mathbf{q}\|} \quad (2.23)$$

This representation can be seen as a modification of the axis-angle representation with $q_w = \cos(\theta/2)$ corresponding to the angle of rotation θ and remaining elements representing the normalized rotation axis $(q_x, q_y, q_z)^\top = \hat{r} \sin(\theta/2)$. Quaternions are

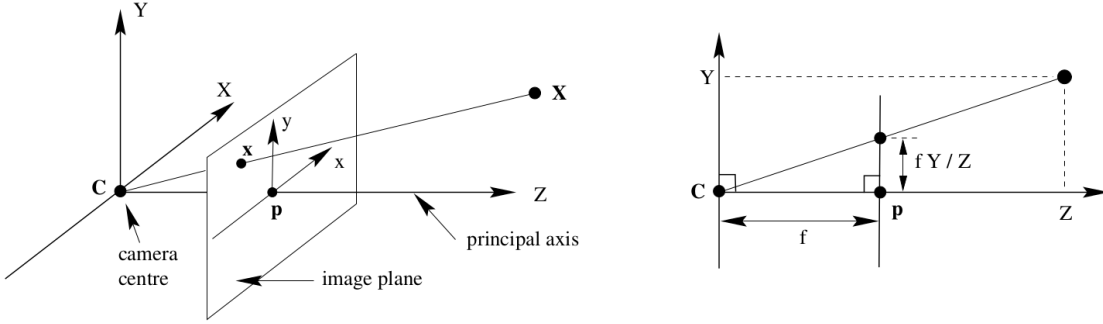


Figure 2.9: Pinhole camera model. Image is taken from [1].

popular due to their compactness and are the representation that is mainly used in this work. To rotate a 3D point \mathbf{X} using quaternions, it has to be first embedded into the imaginary part of the quaternion $\mathbf{X}_q = (0, X, Y, Z)$, and then transformed using $\mathbf{q} \cdot \mathbf{X}_q \cdot \mathbf{q}^{-1} = (0, X', Y', Z')$. Alternatively, quaternion can also be easily converted back to \mathbf{R} as follows:

$$\mathbf{R}(\mathbf{q}) = \begin{pmatrix} 1 - 2q_y^2 - 2q_z^2 & 2q_xq_y - 2q_zq_w & 2q_xq_z + 2q_yq_w \\ 2q_xq_y + 2q_zq_w & 1 - 2q_x^2 - 2q_z^2 & 2q_yq_z + 2q_xq_w \\ 2q_xq_z + 2q_yq_w & 2q_yq_z + 2q_xq_w & 1 - 2q_x^2 - 2q_y^2 \end{pmatrix} \quad (2.24)$$

Another useful property of quaternions is their ability for convenient interpolation. In particular, given two unit quaternions q_1, q_2 and interpolation factor μ , we can use e.g., linear interpolation (*lerp*) or spherical linear interpolation (*slerp*):

$$\text{lerp}(\mathbf{q}_1, \mathbf{q}_2, \mu) = (1 - \mu)\mathbf{q}_1 + \mu\mathbf{q}_2 \quad (2.25)$$

$$\text{slerp}(\mathbf{q}_1, \mathbf{q}_2, \mu) = \frac{\sin((1 - \mu)\alpha)}{\sin \alpha} \mathbf{q}_1 + \frac{\mu\alpha}{\sin \alpha} \mathbf{q}_2, \quad \text{with } \alpha = \arccos(\mathbf{q}_1 \cdot \mathbf{q}_2) \quad (2.26)$$

2.2.2 Pinhole Camera Model

In computer vision, a camera model describes the projection of the 3D world onto the 2D image plane. The most common and widespread camera model is the so-called pinhole camera model. It considers the camera as a lens-free infinitesimally small hole. Light rays passing through this hole form a projection on the image, following the procedure depicted in Fig. 2.9. Consequently, this model does not account for distortions and is merely an idealistic approximation. However, this approximation is sufficiently precise for most application and grants the model popularity due to its simplicity and straightforward calibration techniques.

The pinhole camera model is most commonly parametrized by a focal length f , which is the distance between the camera center and the image plane, and the principle point o_x, o_y , describing the intersection between the image plane and the principle axis. This

2 Background

forms the intrinsic camera matrix \mathbf{K} :

$$\mathbf{K} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \quad (2.27)$$

The parameters of the intrinsic camera matrix are estimated using various calibration techniques [18].

To project a 3D point $\mathbf{X} = (X, Y, Z)^\top$ onto the image plane at the pixel $\mathbf{x} = (x, y)^\top$, the projection operator $\pi : \mathbb{R}^3 \rightarrow \mathbb{R}^2$ is used:

$$\pi(\mathbf{X}) := \left(\frac{f_x X}{Z} + c_x, \frac{f_y Y}{Z} + c_y \right)^\top \quad (2.28)$$

The inverse operation is called back-projection and it maps given pixel coordinates back to 3D points as follows:

$$\left(\frac{x - c_x}{f_x} Z, \frac{y - c_y}{f_y} Z, Z \right)^\top \quad (2.29)$$

If Z value is not available, back-projection cannot be uniquely defined. However, it allows to compute a projective ray from the given pixel back into the world, which includes the original 3D point.

2.3 Pose Estimation

The topic of 3D object pose estimation is vast spanning a large number of different methods. In this section, we cover the most relevant of them. The methods are coarsely grouped into three categories: template-based, correspondence-based, and methods directly regressing the pose from the input.

2.3.1 Template-Based Methods

Template methods are based on a rather simple idea. Each template represents a global descriptor capturing the object's appearance under a certain view angle. A set of such templates is extracted to cover all possible object's views. At the test time, the entire template set is compared to the input image at all possible locations using a predefined matching score. The best candidates are often pruned using e.g., non-maximum suppression. Then, since templates incorporate the information about the object view and scale, using the best candidate allows us to recover the full 6D pose of the object (cf. Fig. 2.10).

While rather straight-forward, template based methods have a number of drawbacks. First of all, in real world we cannot extract templates for all possible views and use a discrete set of views instead. As a result, the found estimate is often very imprecise and additional post-processing refinement methods are required, e.g., Iterative Closest Point (ICP). Using a large number of templates can improve the accuracy, but in return



Figure 2.10: Template-based methods. A database of templates covers possible object views. Each template is compared to the input image and the best fitting template is used as a pose estimate.

leads to a slow matching during the test phase. Moreover, since template-based methods usually rely on global descriptors, they are often very sensitive to occlusions and clutter. However, despite the drawbacks, optimized template methods proved to be very efficient during the test stage.

The early template-based methods were relying on matching object gradients, e.g., using HOG (histogram of oriented gradients) descriptors [19]. Initially built for pedestrian detection, HOG can also be used for pose estimation. However, it is built to be to some degree invariant to the object appearance, which allows to handle the class-specific variability. While perfectly suited for human detection, rigid object pose estimation expects a higher degree of discriminability. This motivated researchers to seek for other solutions. Hinterstoisser et al. introduced the instance specific template methods [20, 21] relying either on RGB gradients (LINE2D), surface normals computed from depth images (LINE3D), or on their combination (LINEMOD). Due to a very efficient implementation based on quantizing and binarizing gradients, the method was capable to cover the existing angle and scale combinations and still estimate the pose in real time. A follow-up work from the same authors [3] demonstrated how to extract templates from synthetic 3D models and then apply them to real data. This was of a great advantage to industrial applications since all possible views and scales for a given object can be generated at no cost. However, the pose space used in LINEMOD is discrete and ICP is used to get reliable estimates. Moreover, despite being real-time, the number of templates in LINEMOD scales linearly with the number of objects the pose has to be estimated for. The scalability issue was subsequently covered by the follow-up works. For example, the work of Konishi et al. [22] proposed to use hierarchical pose trees (HPS) for arranging templates resulting in faster matching. Alternatively, Rios-Cabrera et al. [23] built template cascades using discriminative learning again resulting in a significant matching speed up. Subsequently, Kehl et al. [24] and Hodan et al. [25] proposed to use hashing functions making matching scale sub-linearly with respect to the number of objects.

2 Background

Most recent template approaches rely on CNNs to learn descriptors from data using the process called manifold learning as opposed to utilizing hand-crafted templates. Manifold learning is an approach to non-linear dimensionality reduction, motivated by the idea that high-dimensional data, e.g., images and video, can be efficiently represented in a lower-dimensional space. This concept using CNNs was well studied by Hadsell et al. [26], where image data are mapped directly to the similarity-preserving descriptor space. In order to learn the mapping, the authors used the so-called Siamese network, which takes two inputs instead of one. The cost function was defined such that the squared Euclidean distance between similar objects was minimized and for dissimilar objects the hinge loss was applied forcing them to be pulled apart using a margin term.

The later manifold learning approaches, such as the work of Guo et al. [27], started to use triplet networks, which outperform Siamese networks in generating well-separated manifolds (cf. Hoffer et al. [28]). A triplet network, as the name suggests, takes three images as an input, where two images belong to the same class and the third one to another class. This allows for faster and more robust mapping since both positive and negative samples are taken into account within a single term. Building on these advances, the method by Wohlhart et al. [29] introduced the pose into the equation using a triplet CNN with a specifically designed loss function. The loss function imposes two constraints: the Euclidean distance between the views of dissimilar objects is large, whereas the distance between the views of objects of the same class is relative to their poses. Therefore, the method learns the embedding of the object views into a low-dimensional descriptor space. The matching is then done by applying efficient and scalable nearest neighbor search methods on the descriptor space to retrieve the closest neighbors. Moreover, apart from only finding the object’s pose, it also finds its identity, solving two separate problems at the same time and further increasing the value of this method.

The approach presented in Chapter 3 of this thesis is a template-based method largely inspired by the work of Wohlhart et al. [29]. We extend the method by introducing an additional rotational degree of freedom (making it 3DoF), improve the descriptor space by introducing a modified loss function, study the effect of different input modalities, and introduce and analyze new training augmentations. Building on the improvements, we also introduce a multi-task extension combining manifold learning with direct pose regression leveraging the advantages of both.

2.3.2 Correspondence-Based Methods

A popular alternative to template-based methods are methods utilizing correspondences (see Fig. 2.11). They usually comprise two stages: establishing correspondences between the input image and 3D model, and the actual pose estimation that tries to align correspondences by minimizing a predefined error metric. In real-world applications one often gets false correspondences that can severely harm the resulting pose. However, methods like RANSAC (Random Sample Consensus) allow to efficiently handle such cases. The power of correspondence-based methods lies in their increased robustness to occlusions and clutter since only a few correspondences are needed to actually estimate a pose.

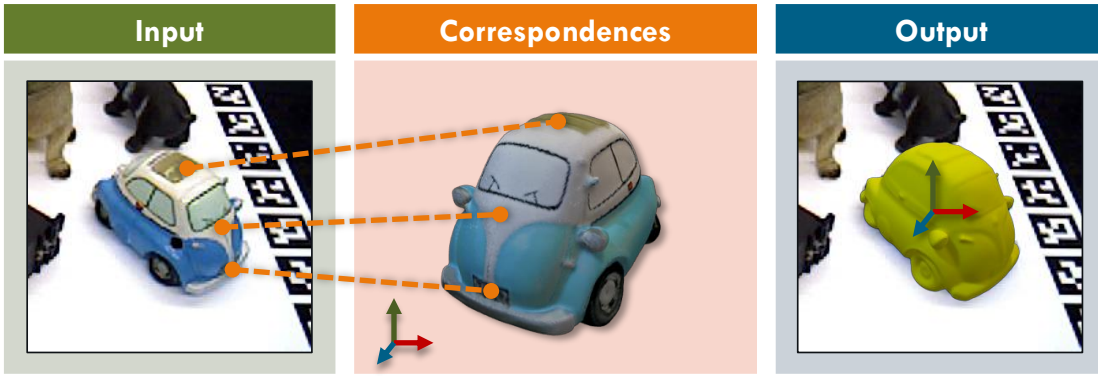


Figure 2.11: Correspondence-based methods. Given a set of 1-to-1 correspondences between the input image and 3D object model, we can recover the pose by aligning correspondences minimizing a predefined error metric.

Early correspondence-based methods relied on sparse feature detectors, extracting and describing interest points with local descriptors. For example, Lowe [30] used SIFT (scale invariant feature transform) detecting blobs in scale-space to cluster images from different but similar viewpoints to a single model. Although proven to be efficient for object recognition, SIFT required a significant computation complexity being a serious drawback for real-time applications. Therefore, the follow-up works focused on speeding it up. Bay et al. [31] introduced SURF (speed up robust feature) that approximates SIFT using Haar filters making it faster. An even faster alternative named ORB (Oriented FAST and Rotated BRIEF) was proposed by Rublee et al. It combined the strengths of the FAST key point detector and modified BRIEF descriptor. Later, sparse feature pipelines were combined with machine learning to improve the interest points [32, 33] and the matching quality [34, 35]. Most notably Yi et al. [36] introduced LIFT (learned invariant feature transform) that encodes SIFT into a single end-to-end differentiable network. Therefore, the three components of standard pipelines (feature extraction, orientation estimation, and descriptor extraction) could be learned directly from data and consequently outperform the former hand-crafted methods.

The common drawback of traditional sparse correspondence methods is the texture requirement. If an object has little to no texture, a sparse feature detector will most likely fail to detect stable feature points. Similarly to template-based methods, deep learning helps to significantly improve in this regard. Crivellaro et al. [37] train a part-based approach representing each part by a set of 3D control points. These control points are virtual and do not necessarily correspond to specific image features, making this representation invariant to the part’s location and relying only on its appearance. Alternatively, Rad et al. presented a holistic three-stage approach called BB8 [38]. In the first two stages the coarse-to-fine segmentation is performed, the result of which is then fed to the third network trained to output projections of the object’s bounding box points. Knowing 2D-3D correspondences, a 6D pose can be estimated using a PnP (Perspective-n-Point) solver. The main disadvantage of this pipeline is its multi-stage

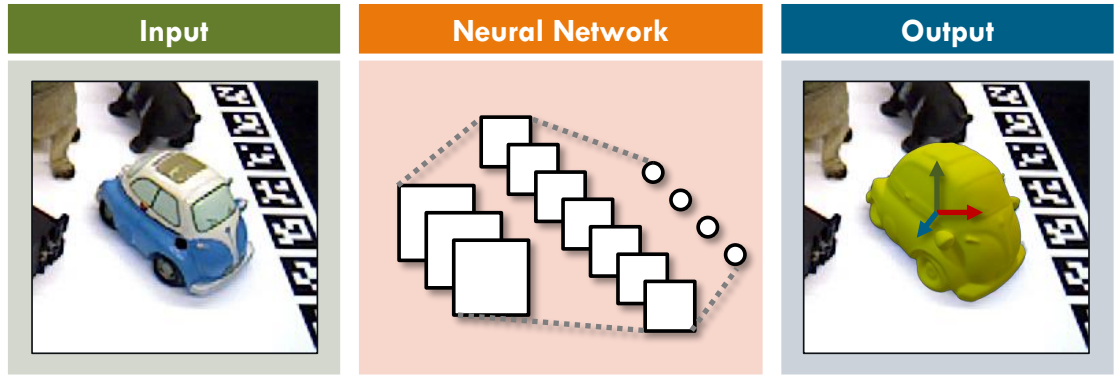


Figure 2.12: Direct pose regression. A neural network maps an input image containing the object directly to its pose estimate.

nature, resulting in slow run times. Building on the popular 2D detector YOLO [39] and BB8 ideas, YOLO6D [40] proposed a novel deep learning architecture capable of efficient and precise object detection and pose estimation without refinement. As is the case with BB8, the key feature here is to perform the regression of reprojected bounding box corners in the image. Once correspondences are estimated, the pose can be estimated using a PnP solver, similarly to BB8.

Another alternative is to use dense correspondences. While establishing few correspondences is likely to result in an inaccurate pose, dense correspondences can still lead to a good solution. For example, this was shown in the works of Shotton et al. [41], Guzman-Rivera et al. [42], and Valentin et al. [43], where the authors used pixel to 3D scene point correspondence prediction based on random forest for camera localization. The depth image is then used to establish 3D-3D correspondences and fit a rigid body transform by means of the iterative pose refinement using RANSAC. Along camera regression works, Taylor et al. [44] introduced a dense correspondence solution for human pose estimation.

The commonality of the aforementioned dense-correspondence approaches is their reliance on depth data. Depth information disambiguates the object’s scale that is the most critical in RGB images due to perspective projection. Therefore, using only RGB images for detection and 6D pose estimation is a quite challenging problem. One of the first representatives of RGB-only camera localization was the work of Brachman et al. [45] building on the Random Forest solution of Shotton et al. [41]. The more recent work for human pose estimation by Guler et al. [46] introduces a framework to densely map canonical 2D coordinates to human bodies.

Building on the ideas of the dense correspondence works, we developed one of the first RGB-only CNN-based rigid pose estimation methods relying on dense correspondences in Chapter 4. In Chapter 5 we use the developed method as a component in the introduced autolabeling pipeline for recovering 9D cuboids and object shapes.

2.3.3 Direct Pose Regression Methods

While the previously described methods utilize multi-stage pipelines, deep neural nets allow to learn an end-to-end mapping from the input image to the pose (see Fig. 2.12). Arguably, the most recognized such work is PoseNet [47] by Kendall et al., where a CNN is employed to regress the position and orientation of a camera given an RGB image. Walch et al. [48] improve the localization results by combining CNN with LSTM (long short-term memory) units. Subsequently, Kendall and Cipolla [49] studied the effect of different loss functions on PoseNet performance further improving the performance. Alternatively, Kehl et al. in their method SSD6D [50] introduced a 6D pose estimation pipeline based on a discrete viewpoint classification rather than direct regression of rotations. All possible poses are divided into a large number of discrete ones, and each of them is further divided into a smaller number of discrete in-plane rotations. This significantly reduces the solution space, which helps during training and allows to achieve more reliable pose estimates. While these methods are able to infer the camera’s 6 degrees of freedom (DoF) in an end-to-end fashion using only an RGB image as input, the reported accuracies are still significantly lower than the reported results based on point correspondence and template-based methods.

In Chapter 3, we introduce a multi-task pipeline combining direct pose regression with manifold learning. By using this combination, we are able to significantly improve the regression performance as well as to create more robust feature descriptors compared to the single task methods.

2.3.4 Pose Refinement

Often pose estimation methods result in imprecise pose estimates. A common solution to this is to use pose refiners as a post-processing step. The most popular refinement method is the iterative closest point or ICP. However, other types of refiners emerged such as deep learning-based refiners and refiners based on differentiable rendering.

Iterative Closest Point (ICP)

Iterative Closest Point, or ICP for short, is arguably the most popular approach for aligning point clouds with dozens of different implementations. The first classical implementations of it were described in the works of Besl and McKay [51] and Chen and Medioni [52]. Both variants as well as their later modifications and improvements consist of the following common steps. In the first step, we select and match the points in both point clouds. Each point pair is then weighted and unreliable points are rejected according to a predefined criterion. Then, provided with 1-to-1 correspondences, we aim to minimize a chosen error metric depending on the ICP implementation.

There are two main approaches to choosing an error metric for pairs of points. Besl and McKay [51] introduced the point-to-point metric, which minimizes the pair-wise distances between the two point clouds. Alternatively, Chen and Medioni [52] proposed the point-to-plane metric minimizing the distance between the points of the first point cloud and the tangent planes to the corresponding points of the second point cloud.

2 Background

The point-to-point variant has an analytical solution and is guaranteed to converge to a minimum, although it might be a local one. On the other hand, the point-to-plane variant does not have a respective analytical solution and is usually solved using the Levenberg-Marquardt solver. Despite this limitation the point-to-plane formulation usually results in a better fitting alignment.

Since RGB-only methods gain more and more popularity due to omnipresent availability of RGB sensors, ICP methods working in such environment are also of great importance. One of the most popular edge-based solutions was presented by Harris [53], and further adapted by Drummond and Cipolla [54], and Kehl et al. [55]. The available 3D model of the object of interest is rendered onto the image and a sparse set of 3D contour points is extracted. Then, each such 3D point casts a ray perpendicular to its orientation to find the closest extracted image edge. The error metric is therefore reformulated to seek the best alignment by minimizing the corresponding distances in the image plane.

Deep Learning Pose Refiners

The other prominent direction in pose refinement is based on deep learning. The recent works of Manhardt et al. [56] and Li et al. [9] show promising results. Both refiners are conceptually very similar and are designed to output relative transformation between the real input image patch and the patch containing the object rendered with the predicted pose. The main differences between them are the used backbone architectures and loss functions. Both refinement algorithms rely on external object detection and pose estimation algorithms: for DeepIM [9] it is PoseCNN, and for Manhardt et al. [56] it is SSD6D [50]. The former relies on real data, whereas the latter focuses on training on synthetic images. Initial poses are used to render the object on top of a black background. Afterwards, the rendered patch and a corresponding crop of the real image are fed into a neural network which predicts a relative transformation. In case of [56] the network accepts those two images as two separate inputs, then uses a pretrained Inception network [57] as a fixed feature extractor to produce features that are later stacked together. Those stacked features are used to predict relative transformation from the rendered pose to the pose to be estimated. The authors proposed to utilize a visual loss which aligns contours of observed and rendered objects. On the other hand, [9] proposed to stack extracted crops right away and use the resulting tensor as input to the network instead of the standard three-channel image input. They directly optimize the ADD score [58] as their main loss function, which emphasizes more the actual 3D pose rather than reprojection similarity. Moreover, they discuss in detail a proper parametrization of the pose regression so that it is more robust to scale changes.

In Chapter 4 we propose our own refinement solution combining the best features of the above-mentioned works. Moreover, in contrast to [56, 9], our refinement network can be trained indifferently of whether real or synthetic data is used.

Differentiable Rendering-Based Refiners

The other variety or pose refiners that has recently emerged is based on differentiable rendering. Differentiable rendering approaches produce 2D images from 3D models simulating the process of image formation by relating each pixel to the 3D parameters. The differentiability allows to obtain a dense pixel supervision for a variety of 3D reasoning tasks including pose estimation.

The first advances in differentiable rendering were the works of Loper et al. [59] and Kato et al. [60]. Instead of making the full complex rendering pipeline differentiable, they opted to only approximate the backward gradient using hand-crafted functions. While showing promising results in the task of 3D reconstruction from images, the discrepancy between forward and backward propagation may result in uncontrolled behavior limiting the applicability to other possible tasks. The more recent work of Liu et al. [61] introduced a fully differentiable pipeline by reformulating conventional discrete rendering operations showing significant improvements in the tasks of mesh reconstruction and image-based pose refinement. In this case, the loss term for the pose refinement is defined as a difference between an input image and a rendering. A gradient-based solver, e.g. SGD or Adam, is then used to update the parameters such as the pose or color of the model. In contrast to the above-mentioned rasterizer-based solutions, Li et al. [62] propose a path tracing approach towards differentiable rendering of triangle meshes. It allows to easily simulate more advanced rendering effects such as mirrors and shadows. However, the ray-tracing-based solutions proved to be much more resource- and time-consuming than their rasterizer-based counterparts.

Parallel to the work of Liu et al. [61], we introduce a fully differentiable surfel-based renderer that can also be extended to render signed distance fields (SDFs) as shown in Chapter 5. This allows to not only optimize over the object’s pose and color, but also its shape.

2.4 Domain Adaptation

The domain gap, or more particularly realism gap, is a very well known problem for computer vision methods that rely on synthetic data, as the knowledge acquired from these modalities usually poorly translates to the more complex real domain, resulting in a dramatic accuracy drop. Several ways to tackle this issue have been investigated so far and in this section we will cover the most relevant of them (see Fig. 2.13).

Simulation Tools A first obvious solution is to improve the quality and realism of the synthetic models. Several works try to push forward simulation tools for sensing devices and environmental phenomena. For instance, state-of-the-art depth sensor simulators, e.g., Landau et al. [63] and Planche et al. [64] work fairly well, as the mechanisms impairing depth scans have been well studied and can be rather well reproduced. In the case of color data however, the problem lies not in the sensor simulation, but in the actual complexity and variability of the color domain (e.g., sensibility to lighting conditions, texture changes with wear-and-tear, etc.). This makes it extremely arduous

2 Background

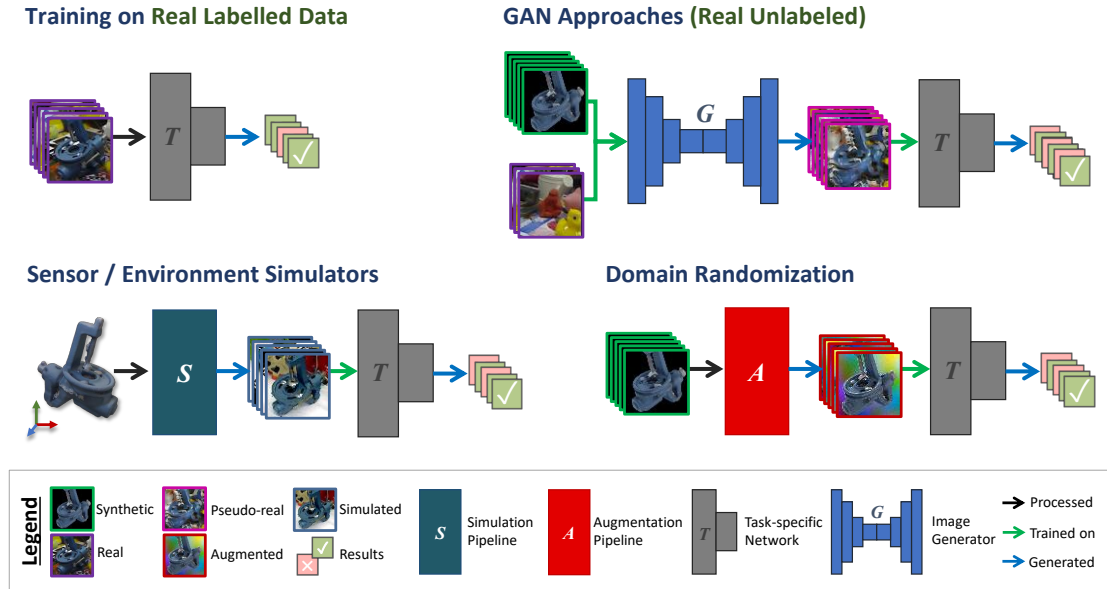


Figure 2.13: Domain adaptation methods. An overview of different method classes tackling the domain gap problem.

to come up with a satisfactory mapping, unless precise, exhaustive synthetic models are provided (e.g., by capturing realistic textures). Proper modeling of target classes is however often not enough, as recognition methods would also need information on their environment (background, occlusions, etc.) to be applied in real-life scenarios.

GAN Approaches For this reason, and in complement simulation tools, recent CNN-based methods are trying to further bridge the realism gap by learning a mapping from rendered to real data, directly in the image domain. These solutions, e.g., Taigman et al. [65], Shrivastava et al. [6], Bousmalis et al. [66], and Isola et al. [67], are mostly based on unsupervised conditional generative adversarial networks (GANs).

- **Generative Adversarial Networks (GANs)** First introduced by Goodfellow et al. [68], and quickly improved and derived through numerous works, e.g., [69, 70, 71, 67, 72], the GAN framework has proven itself a great choice for image generation [68, 69, 73], edition [67, 7, 6, 66], or segmentation [74, 75, 76]. The generator network in these solutions benefits from competing against a discriminator one (with adversarial losses) to properly sample realistic images from the learned distribution. Methods conditioned on noise vectors [77, 78], labels [77, 79, 73] and/or images [67, 7, 79, 6, 8, 66] soon appeared to add control over the generated data. Given these additions, recognition pipelines started integrating conditional GANs. Some works are for instance using a classifier network along their discriminator, to help the generator grasp the conditioned image distribution by back-propagating the classification results on generated data [80, 66]; while others are using GANs

to estimate the target domain distribution, to sample training images for their classifier [81, 82, 65, 6, 8, 66].

Despite the advantages of the GAN-based methods, they still require a set of real relevant image samples to learn their mapping. Moreover, if the real set is not large enough, these methods can overfit to the chosen target domain and exhibit a decline in performance for unfamiliar out-of-distribution samples.

Learning Domain-Invariant Features Other approaches are focusing on adapting the recognition methods themselves to make them more robust to domain changes. Therefore, instead of working in the image space, they aim to learn domain-invariant features. For instance, DANN by Ganin et al. [82] adds a domain classifier head connected to the feature extractor of the task network. The extended task network is then trained using a presented gradient reversal layer that maximizes the domain classifier error ensuring that the feature distributions over the two domains are made similar. Tzeng et al. [83] utilize a domain discriminator network with adversarial loss to force a network to learn domain-invariant features. Rad et al. [84] aim to predict synthetic features from real ones and use them for inference, but it requires target annotations. In their follow-up work [85], Rad et al. propose a method to map color images to depth domain, which also supports training from synthetic depth data. However, unlabeled real images from the target domain are still required to learn such a mapping.

Domain Randomization Considering real-world and industrial use cases when only texture-less CAD models are provided, Sadeghi and Levine [86] and Tobin et al. [87] are compensating the lack of target domain information by training their recognition algorithms using heavy image augmentations or a randomized rendering engine. The claim is that with enough variability in the simulator, real data may appear just as another variation to the model. While being completely independent of real data, domain adaptation methods still heavily underperform when compared to solutions using data from the target domain.

Considering similar applications (when no real samples or even texture information are available), in Chapter 7 we present a pipeline that tackles the problem of synthetic-to-real domain adaptation from a different angle, i.e. by mapping real data to synthetic domains. We demonstrate how this different approach not only improves the end accuracy, but also makes the overall solution more modular. Subsequently, in Chapter 8 we introduce a novel adversarial domain randomization pipeline driven by the task network.

Part I

Pose Estimation

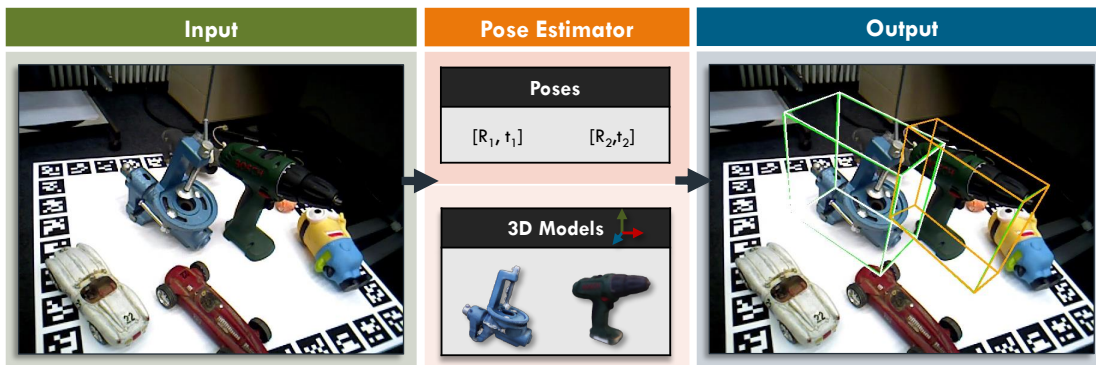


Figure 2.14: 3D Object Pose Estimation. Given an input image, the aim is to recognize object instances and estimate their poses. This allows to project them back onto the image.

Object pose estimation is a widely researched topic in the field of computer vision with many application possibilities in augmented reality, medicine, surveillance, and robotics. Despite its popularity, there is still a large room for improvement. The current methods often struggle from clutter and occlusions and are sensitive to background and illumination changes. The most common pose estimation methods use a single classifier per object, making their complexity grow linearly with the number of objects for which the pose has to be estimated. Moreover, the majority of them is trained using real annotated data, which is expensive to acquire and annotate. In the following chapters we propose deep network-based solutions to tackle the mentioned challenges.

The first chapter aims to tackle the problems of 3D pose estimation and object instance recognition using manifold learning. The descriptor neural network maps images to a lower-dimensional highly-discriminative space, which makes it simple to distinguish between the different classes and poses of the objects. The output descriptor is then compared to the template descriptors and the closest neighbor is used to evaluate the recognition results. First we aim to improve the intermediate manifold representation for better scalability and interpretability. Moreover, we analyze the performance of different input modalities using the predefined metrics. Then, we extend this approach further with a goal to make it end-to-end. For this we combine the popular regression approach with manifold learning, which results in a mutual improvement of both.

The second chapter deals with an extended problem of 6D pose estimation and object detection. Based on dense correspondences it maps each RGB pixel of the input image to the corresponding model surface points. These correspondences are then fed to a fast PnP solver along with the camera parameters to estimate a precise 6D pose. We also introduce a deep network-based refiner to further improve the pose estimates. The entire solution can be trained either on real or on fully synthetic data, while achieving state-of-the-art results on the real test data.

The third chapter provides a solution to the more ambitious 9D pose estimation problem. It consists of a more advanced correspondence network and a differentiable SDF

shape database. Here, we not only estimate the rotation and translation of the object, but also its shape and scale. Moreover, we further refine the estimates by utilizing a newly introduced differentiable SDF renderer. While being trained entirely on synthetic data, the method is used to estimate reliable labels from real unlabeled images, which can be further used to train other downstream tasks.

The last chapter presents a new dataset for 6D pose estimation. It features 33 objects and 13 scenes of various difficulty. The main goal of the dataset is to test the current state-of-the-art detectors with respect to the properties desired in the industry, i.e., scalability, robustness to illumination changes, occlusions and clutter. Moreover, we propose a simple pipeline to extend or create new datasets.

3 3D Pose Estimation Based on Manifold Learning

In this chapter, we address the problem of 3D pose estimation and object instance recognition of localized objects in cluttered environments using manifold learning and direct pose regression. Inspired by the approach of Wohlhart et al. [29], we propose a method that introduces the dynamic margin in the manifold learning triplet loss function. Such a loss function is designed to map images of different objects under different poses to a lower-dimensional similarity-preserving descriptor space on which efficient nearest neighbor (NN) search algorithms can be applied. Introducing the dynamic margin allows for faster training times and better accuracy of the resulting low-dimensional manifolds.

We further improve this solution by introducing an efficient multi-task learning framework combining the strengths of manifold descriptor learning and pose regression. By doing so, we can either estimate the pose directly reducing the complexity compared to NN search, or use learned descriptors for the NN descriptor matching. By in-depth experimental evaluation of the multi-task loss function we observe that the view descriptors learned by the network are much more discriminative resulting in almost 30% increase regarding relative pose accuracy. On the other hand, regarding directly regressed poses we also obtain a significant improvement compared to a simple pose regression.

Furthermore, we contribute by the following: adding in-plane rotations (ignored by the baseline method) to the training, treating symmetric objects, proposing new background noise types that help to better mimic realistic scenarios and improve accuracy with respect to clutter, and evaluating new combinations of input image modalities including surface normals. We perform an exhaustive evaluation to demonstrate the effects of our contributions.

3.1 Introduction

We propose an efficient view-based solution inspired by the work of Paul Wohlhart and Vincent Lepetit [29]. The authors of [29] tackle both pose estimation and object instance recognition of already-detected objects simultaneously by learning a discriminative feature space using CNNs. Particularly, given a single RGB-D image patch containing an already-detected object in the center surrounded with the cluttered background, the descriptor CNN is used to map this patch to a lower-dimensional manifold of the computed descriptors. This manifold preserves two important properties: the large Euclidean distance between the descriptors of dissimilar objects, and the distance between the descriptors of the objects from the same class is relative to their poses. Once the mapping is learned, efficient and scalable nearest neighbor search methods can be applied on the

descriptor space to retrieve the closest neighbors for which the poses and identities are known. This allows us to efficiently handle a large number of objects together with their view poses, resolving the scalability issue.

The manifold learning in [29] is performed using the triplet loss function, where the triplet is a group of samples $(\mathbf{s}_i, \mathbf{s}_j, \mathbf{s}_k)$ selected such that \mathbf{s}_i and \mathbf{s}_j represent similar views of the same object and \mathbf{s}_k comes either from the same object with a slightly different pose or from a completely different object. The fixed margin term in the triplet loss sets the minimum ratio for the Euclidean distance between descriptors of similar and dissimilar sample pairs. Using a fixed margin throughout the training results in a slow separation of the manifolds for different objects and similar objects with different poses, causing long training times and limited accuracy in case of short-sized descriptors. To overcome this problem, we introduce a dynamic margin in the loss function by explicitly setting the margin term as a function of an angular difference between the poses for the same object and to a constant value that is larger than the maximal possible angular difference in case of different objects. This allows faster training and better quality of the resulting lower-dimensional manifolds, which, in turn, enables the use of smaller-sized descriptors with no loss of accuracy.

Building up on the proposed solution, we also introduce an end-to-end multi-task learning pipeline, which combines the strengths of manifold learning and regression, to learn robust features from which the object’s pose can be inferred. Thus, we are able to combine the generalization capabilities shown in manifold learning tasks and the variability of regression into a deep learning framework for object pose estimation. For this purpose, we introduce a multi-task loss function, which leverages both manifold learning and regression terms. We analyze how the two tasks influence each other and show that each task can be beneficial to one another in the context of estimating object poses. To summarize, our contributions described in this chapter include the following:

- **The triplet loss with dynamic margin** resulting in faster training times and better accuracy;
- **The multi-task loss function using a combination of regression and manifold learning** to create an end-to-end framework for object recognition and 3D pose estimation;
- **Significant improvement in accuracy and feature robustness** compared to the baseline method;
- **Adding in-plane rotations** existing in real-world scenarios and ignored by the initial method;
- **Introducing new background noise types** for synthetic patches that help to better mimic realistic scenarios and allow for better performance when no real data is used in training.

In the next sections, a detailed description of the proposed pipeline is given. Then, we validate the method to demonstrate the importance of the newly introduced improvements. Most importantly, we compare the standard triplet loss with our dynamic margin

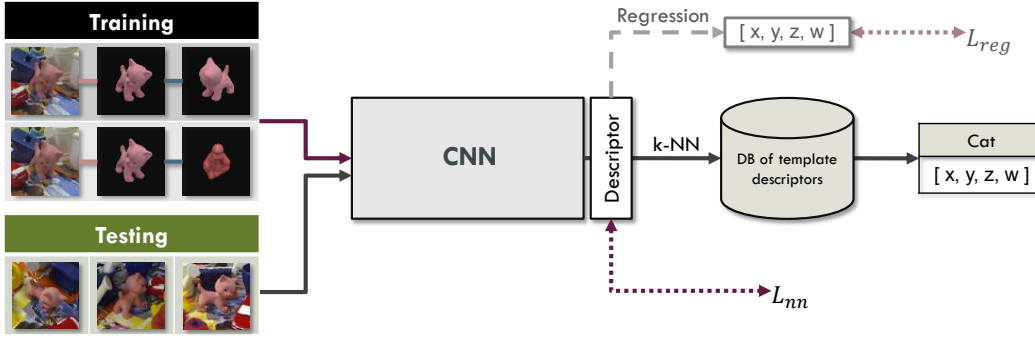


Figure 3.1: Pipeline description. Given an input image patch \mathbf{x}_i , we create corresponding triplets $(\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k)$ and pairs $(\mathbf{x}_i, \mathbf{x}_j)$ to train our model on manifold embedding creating robust feature descriptors. Additionally, the pose can be regressed directly by introducing the regression head as demonstrated in our multi-task pipeline extension. The pose \mathbf{q} can then be obtained either by a direct pose regression or using the resulting feature descriptor for nearest neighbor search in the descriptor database.

implementation demonstrating superior performance for both training times and final accuracy. Then, we validate our extended multi-task pipeline. By using a combination of manifold learning and regression, we are able to significantly improve the regression performance as well as to create more robust feature descriptors compared to the baseline method. Thus, we improve both aspects: nearest neighbor pose retrieval and direct pose regression with our framework and obtain a large accuracy boost.

3.2 Methodology

Our methodology (shown in Fig. 3.1) starts with training a CNN model for a given training set $S_{train} = \{\mathbf{s}_1, \dots, \mathbf{s}_N\} = \{(\mathbf{x}_1, c_1, \mathbf{q}_1), \dots, (\mathbf{x}_N, c_N, \mathbf{q}_N)\}$ consisting of N samples. Each sample \mathbf{s} comprises an image patch $\mathbf{x} \in \mathbb{R}^{n \times n}$ of an object, identified by its class $c \in \mathbb{N}$, together with the corresponding pose vector, $\mathbf{q} \in \mathbb{R}^4$, which gives the orientation represented by quaternions. Given input \mathbf{x} , our objective is to learn a mapping to a highly discriminative descriptor space, from which object class \hat{c} and 3D pose $\hat{\mathbf{q}}$ can be easily extracted. The extraction is executed using the k-nearest neighbor (k-NN) search in the precomputed template database S_{db} defined similarly to S_{train} . This intermediate representation brings practical benefits: scalability and the ability to add and remove objects and poses. To achieve a robust mapping, we train a CNN subject to a triplet-based objective enforcing the Euclidean distance between descriptors from similar image views to be close and from different objects to be far away.

3.2.1 Loss Function

Similarly to [29], the basic objective for training a discriminative descriptor mapping is the loss function is defined as a sum of two separate loss terms $L_{triplets}$ and L_{pairs} :

$$L_{nn} = L_{triplets} + L_{pairs}. \quad (3.1)$$

The first addend $L_{triplets}$ is a loss defined over a set T of triplets, where a triplet is a group of samples $(\mathbf{s}_i, \mathbf{s}_j, \mathbf{s}_k)$ selected such that \mathbf{s}_i and \mathbf{s}_j always come from the same object under a similar pose, and \mathbf{s}_k comes from either a different object or the same object under a less similar pose (Fig. 3.2a). In other words, a single triplet consists of a pair of similar samples, \mathbf{s}_i and \mathbf{s}_j , and a pair of dissimilar ones, \mathbf{s}_i and \mathbf{s}_k . In our terminology, we call \mathbf{s}_i an anchor, \mathbf{s}_j a positive sample or a puller, and \mathbf{s}_k a negative sample or a pusher. The triplet loss component has the following form:

$$L_{triplets} = \sum_{(\mathbf{s}_i, \mathbf{s}_j, \mathbf{s}_k) \in T} \max \left(0, 1 - \frac{\|f(\mathbf{x}_i) - f(\mathbf{x}_k)\|_2^2}{\|f(\mathbf{x}_i) - f(\mathbf{x}_j)\|_2^2 + m} \right), \quad (3.2)$$

where \mathbf{x} is the input image of a certain sample, $f(\mathbf{x})$ is the output of the neural network given the input image, and m is the margin, which introduces the margin for classification and sets the minimum ratio for the Euclidean distance of the similar and dissimilar pairs of samples.

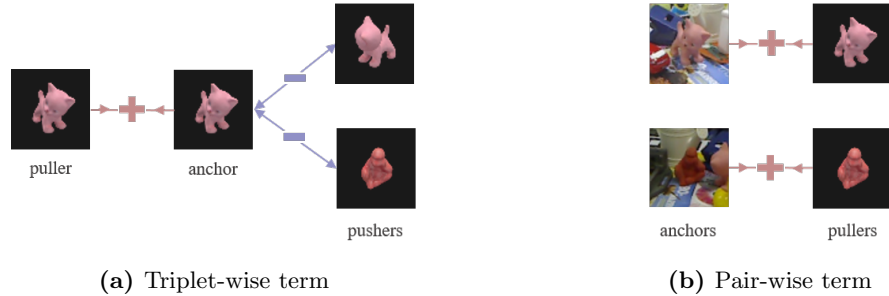


Figure 3.2: CNN input format. Triplets are used to learn a well-separated manifold, whereas pairs make the mapping invariant to various imaging conditions.

By minimizing $L_{triplets}$, one enforces two important properties that we are trying to achieve, namely: maximizing the Euclidean distance between descriptors from two different objects and setting the Euclidean distance between descriptors from the same object so that it is representative of the similarity between their poses.

The second addend L_{pairs} is a pair-wise term. It is defined over a set P of sample pairs $(\mathbf{s}_i, \mathbf{s}_j)$. Samples within a single pair come from the same object under either a very similar pose or the same pose but with different imaging conditions. Different imaging conditions may include illumination changes, different backgrounds, or clutter. It is also possible that one sample comes from the real data and the other from synthetic data. The goal of this term is to map two samples as close as possible to each other:

$$L_{pairs} = \sum_{(s_i, s_j) \in P} \|f(\mathbf{x}_i) - f(\mathbf{x}_j)\|_2^2. \quad (3.3)$$

By minimizing the L_{pairs} , or the Euclidean distance between the descriptors, the network learns to treat the same object under different imaging conditions in the same way by mapping them onto the same point. Moreover, it ensures that samples with similar poses are set close together in the descriptor space, which is an important requirement for the triplet term.

3.2.1.1 Triplet Loss with Dynamic Margin

The triplet loss function, in the way it is used in [29], has one significant drawback. The margin term is a constant and is the same for all the different types of negative samples. This means that we are trying to push apart the objects of same and different classes with exactly the same margin term, whereas the desired goal is to map the objects of different classes farther away from each other. This slows down the training in terms of classification and results in a worse separation of the manifold. The logical solution to this is to set the margin term to be a variable and change it depending on the type of the negative sample.

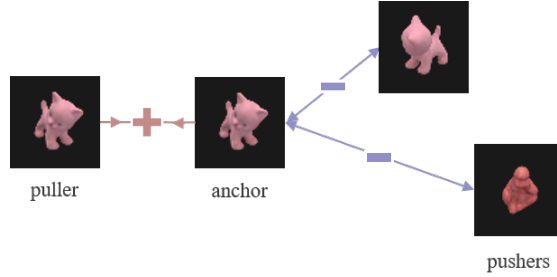


Figure 3.3: Triplet loss with dynamic margin. A better separation achieved by setting different inter- and intra-class margins.

We propose the following solution. If the negative sample belongs to the same class as the anchor, the margin term is set to be the angular distance between the samples. If, however, the negative sample belongs to a different class, the distance is set to a constant value γ that is larger than the maximal possible angular difference. The effect of the dynamic margin is illustrated in Fig 3.3. The updated loss function is defined as follows:

$$L_{triplets_{dm}} = \sum_{(s_i, s_j, s_k) \in T} \max \left(0, 1 - \frac{\|f(\mathbf{x}_i) - f(\mathbf{x}_k)\|_2^2}{\|f(\mathbf{x}_i) - f(\mathbf{x}_j)\|_2^2 + m} \right),$$

$$\text{where } m = \begin{cases} 2 \arccos(|\mathbf{q}_i \cdot \mathbf{q}_j|) & \text{if } c_i = c_j, \\ \gamma & \text{else.} \end{cases} \quad (3.4)$$

3.2.1.2 Multitask Loss

While k-nearest neighbor search has efficient KD-tree-based implementations having a sublinear complexity, it cannot beat regression having a constant complexity. On the other hand, direct pose regression does not provide the same level of pose quality template-based methods are capable of. To combine the strengths of two approaches, we model the problem as a multi-task learning by combining manifold learning with direct pose regression. Thus the overall objective function can be written as

$$L_{mtl} = (1 - \lambda)L_{reg} + \lambda L_{nn}, \quad (3.5)$$

where λ is a regularization parameter. L_{reg} and L_{nn} are the objective functions for the pose regression task and the manifold learning task respectively.

Regression To make the direct regression possible, we extend our CNN by adding a regression head (see Fig. 3.1) that takes the lower dimensional feature vector $f(\mathbf{x}) \in \mathbb{R}^d$ and outputs the pose prediction $\hat{\mathbf{q}}$. To train the regression head, the following loss function is used:

$$L_{reg} = \left\| \mathbf{q} - \frac{\hat{\mathbf{q}}}{\|\hat{\mathbf{q}}\|} \right\|_2^2, \quad (3.6)$$

where $\|\cdot\|_2$ is the l_2 -norm and \mathbf{q} is the corresponding ground truth pose.

3.2.2 Dataset Generation

The datasets we use contain the following data: 3D mesh models of the objects and RGB-D images of the objects in real environments with their camera poses. Using these data, we generate three sets: the training set S_{train} , the template set S_{db} and the test set S_{test} . The training set is used exclusively for the purpose of training the network. The test set S_{test} , as its name suggests, is used only in the test phase for evaluation. The template set S_{db} is used in both training and test phases. Each set consists of samples,

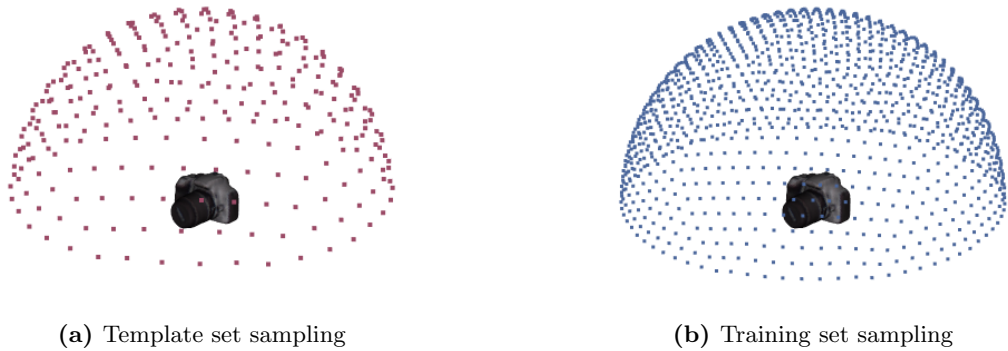


Figure 3.4: Different sampling types. each vertex represents a camera position from which the object is rendered.

where each sample $\mathbf{s} = (\mathbf{x}, c, \mathbf{q})$ is made of an image \mathbf{x} , the identity of the object c , and the pose \mathbf{q} .

The first step in preparing the data is the generation of samples for the sets. Our sets are constructed from two types of imaging data: real and synthetic. The real images represent the objects in real-world environments and are generated using a commodity RGB-D sensor, e.g., Kinect or Primesense. They have to be provided together with the dataset. The synthetic images, however, are not initially available and must be generated by rendering provided textured 3D mesh models.

Given 3D models of the objects, we render them from different view points covering the upper part of the object in order to generate synthetic images. In order to define the rendering views, an imaginary icosahedron is placed on top of the object, where each vertex defines a camera position. To make the sampling finer, each triangle is recursively subdivided into four triangles. The method defines two different sampling types: a coarse one (Fig. 3.4a), achieved by two subdivisions of the icosahedron, and a fine one (Fig. 3.4b), achieved by three consecutive subdivisions. The coarse sampling is used to generate the template set S_{db} , whereas the fine sampling is used for the training set S_{train} . For each camera pose (vertex) an object is rendered on an empty (black) background and both RGB and depth channels are stored.

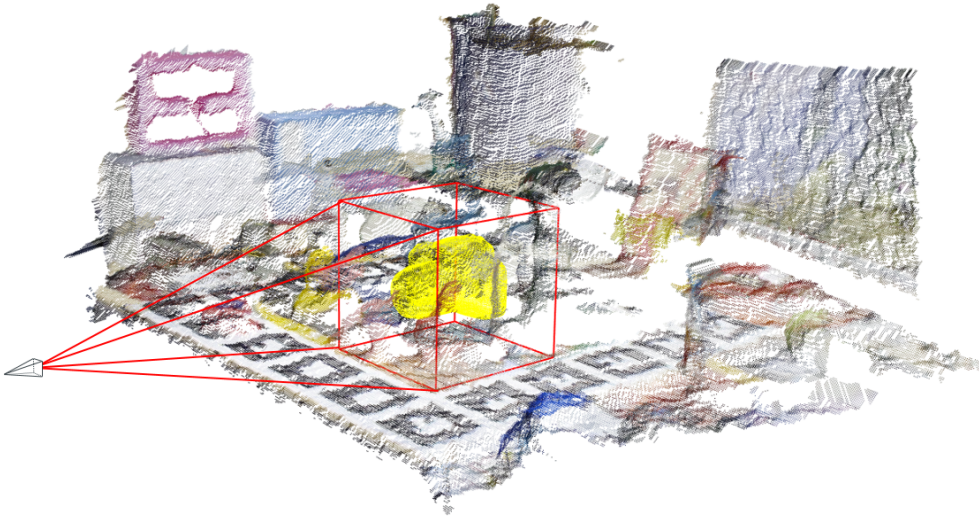


Figure 3.5: Patch extraction. the object of interest (shown in yellow) is covered by the cube of 40 cm^3 in dimension; only RGB and depth data covered by the cube is taken to generate a single patch.

When all the synthetic images are generated and we have both real and synthetic data at hand, samples can be generated. For each of the images, we extract small patches covering and centered on the object. This is done by virtually setting a cube, of 40 cm^3 in dimension, centered at the object’s center of mass as shown in Fig. 3.5. When all the patches are extracted, we normalize them. RGB channels are normalized to the zero mean and unit variance. The depth values within the defined bounding cube are

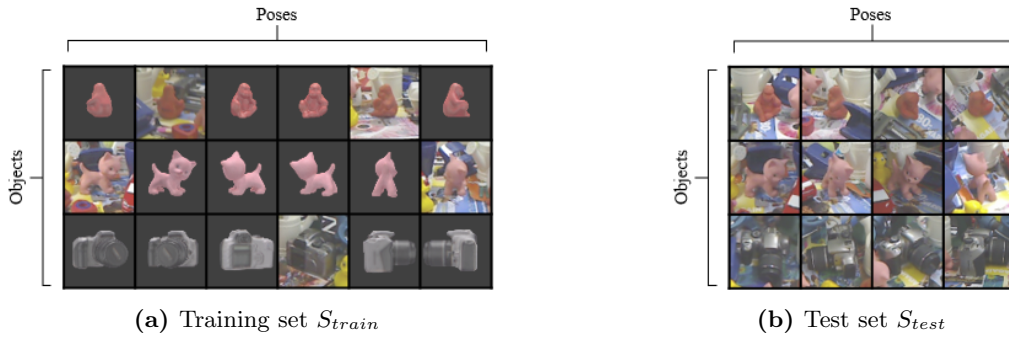


Figure 3.6: Datasets. The training set S_{train} consists of both real and synthetic (fine sampling); the test set S_{test} consists of the real data not used for the training set S_{train} .

normalized and mapped to the range $[0, 1]$ and the rest of the values are clipped. Finally, each patch \mathbf{x} is stored within a sample in addition to the object’s identity c and its pose \mathbf{q} . The next step is to divide the samples between the sample sets S_{train} , S_{db} and S_{test} , accordingly.

The template set S_{db} contains only synthetic samples with the renderings coming from the coarse sampling (Fig. 3.4a). It is used in both training (to form triplets) and test (as a database for the nearest neighbor search) phases. The samples of S_{db} define a search database on which the nearest neighbor search is later performed. This is the main reason for the coarse sampling: We want to minimize the size of the search database for faster retrieval. However, the sampling defined for the template set also directly limits the accuracy of the pose estimation.

The training set S_{train} (Fig. 3.6a) consists of a mix of synthetic and real data. The synthetic data represent samples coming from the renderings defined by the fine sampling (Fig. 3.4b). Approximately 50% of the real data are added to the training set. This 50% is selected by taking the real images that are close to the template samples in terms of the pose. The rest of the real samples are stored in the test set S_{test} (Fig. 3.6b), which is used to estimate the performance of the algorithm.

3.2.2.1 In-plane Rotations

The initial method proposed in [29] has a major limitation of not considering in-plane rotations or, in other words, omitting one additional degree of freedom. However, in real-world scenarios, it is hardly possible to avoid in-plane rotations. In order to introduce them to the algorithm, one needs to generate additional samples with in-plane rotations and define a metric to compare the similarity between the samples to be able to build triplets.

Generating synthetic in-plane rotated samples is relatively simple. What we need is to rotate the view camera at each sampling point (vertex) around its shooting axis and record a sample with a certain frequency as shown in Fig. 3.7. Currently, for the

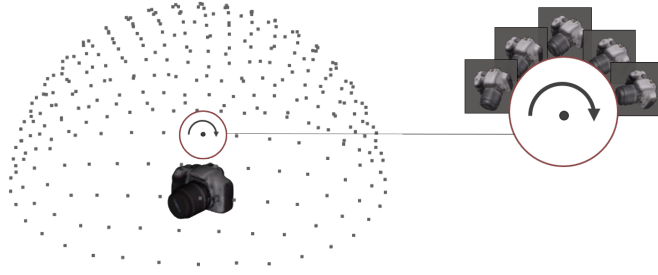


Figure 3.7: In-plane rotations. At each vertex extra views are rendered by rotating the camera around the axis pointing at the object center.

LineMOD dataset, we generate seven samples per vertex, going from -45 to 45 degrees with a stride of 15 degrees.

As for the similarity metric, we cannot use the dot product of the sampling point vectors anymore, as was proposed in the initial method, since we cannot incorporate an additional degree of freedom this way. Instead it was decided to use the quaternions to represent rotations of the models and the angle between the samples' quaternions as a pose comparison metric $\theta(\mathbf{q}_i, \mathbf{q}_j) = 2 \arccos(|\mathbf{q}_i \cdot \mathbf{q}_j|)$.

3.2.2.2 Treating Rotation-Invariant Objects

Four out of fifteen objects of the LineMOD dataset have a property of rotation-invariance and introduce an ambiguity to the generation of triplets needed for the triplet loss. For instance, the *bowl* object is symmetric around an axis, whereas the *cup*, *eggbox* and *glue* object are symmetric around a plane. These four objects need to be treated differently from the rest. This comes from the fact that, in the case of rotation-invariant objects, samples representing different poses might look exactly the same, which can result in faulty triplets required for the triplet loss function.

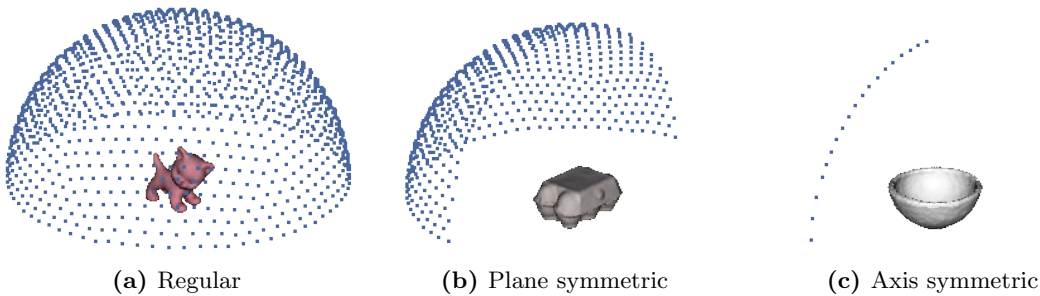


Figure 3.8: Sampling points for different objects types. Vertices represent camera positions from which the object is rendered.

To solve this problem, we render only a limited amount of poses for those objects, such that every image patch is unique. Sample vertices for different object types are demonstrated in Fig. 3.8b and 3.8c. Since both training set S_{train} , and test set S_{test}

also include real samples we also omit ambiguous poses in them and only consider those that are close to the ones coming from the renderer.

3.2.2.3 Surface Normals

Surface normals are considered an extra modality representing an object image, in addition to existing RGB and depth channels, to improve the algorithm accuracy. By definition, a surface normal defined at point $\mathbf{p} \in \mathbb{R}^3$ is a 3D vector that is perpendicular to the tangent plane to the model surface at point \mathbf{p} . Applied to many points on the model, surface normals result in a powerful modality describing its curvature.

In our pipeline, surface normals are calculated based on the depth map images (no additional sensor data required) using the method for the fast and robust estimation in dense range images proposed in work [21] and resulting in a 3-channel modality. This approach smooths the surface noise and, therefore, allows for better surface normal estimates around depth discontinuities.

3.2.2.4 Background Noise Generator

One of the most difficult problems for computer vision methods is the treatment of clutter and different backgrounds in images. Since our samples do not have any background by default, it is difficult or sometimes impossible for the network to adapt to the real data full of noise and clutter in the background and foreground.

One of the easiest approaches for solving this problem is to use real images for training. Then, the network might adapt to the realistic data, but the major problem comes when no or very limited real data are available. In these cases, we have to teach the network to ignore the background or simulate backgrounds to cover possible variations.

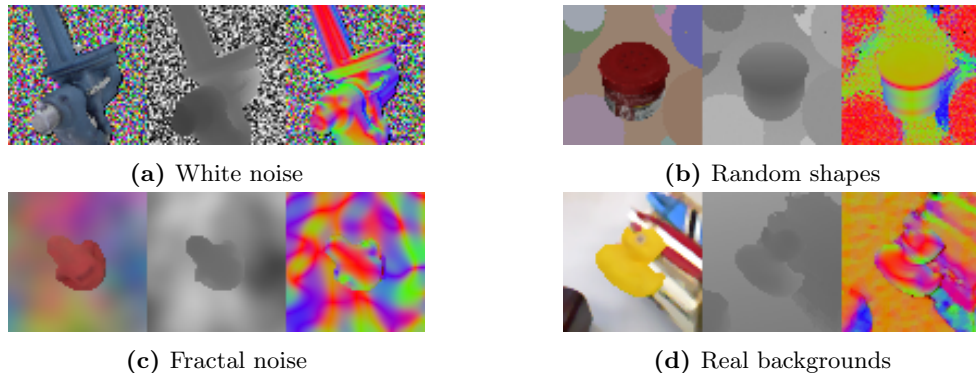


Figure 3.9: Background noise types for synthetic data shown for different channels, i.e., RGB, depth, and normals.

In our implementation, we have a separate class generating different kinds of noise: white noise, random shapes, gradient noise, and real backgrounds.

Table 3.1: Test setups. Each underlined entry represents the tested parameter for a given test.

	Dataset	Training data	Testing data	In-plane rotations	Background augmentation	Data channels	Descriptor dimension	Triplet margin type
Test A	LineMOD, 15 objects	synthetic+real	real	<u>with and without</u>	fractal noise	RGB-D	32	static
Test B	LineMOD, 6 objects	synthetic+real	real	with	fractal noise	RGB-D	<u>3, 32</u>	<u>static, dynamic</u>
Test C	LineMOD, 15 objects	<u>synthetic</u>	real	with	<u>white noise, fractal noise, random shapes, real bg.</u>	RGB-D	32	dynamic
Test D	LineMOD, 15 objects	synthetic+real	real	with	fractal noise	<u>depth, normals, normals+depth</u>	32	dynamic
Test E	<u>BigBIRD, 50 objects</u>	synthetic+real	real	without	fractal noise	RGB-D	32	<u>static, dynamic</u>

The first and the simplest type of noise is white noise (Fig. 3.9a). To generate it, we simply sample a float value from 0 to 1 from a uniform distribution for each pixel. In the case of RGB, we do that three times for each pixel in order to fill all the channels.

The second type of noise is the random shape noise (Fig. 3.9b). The idea is to represent the background objects such that they have similar depth and color values. The color of the objects is again sampled from the uniform distribution, from 0 to 1, and the position is sampled from the uniform distribution, from 0 to the width of the sample image. This approach can also be used to represent foreground clutter by placing random shapes on top of the actual model.

The third type of noise we used is fractal noise (Fig. 3.9c), which is often used in computer graphics for texture or landscape generation and is the most advanced synthetic noise presented here. The fractal noise implementation we use is based on summing together multiple octaves of simplex noise first introduced by Ken Perlin in [88]. It results in a smooth sequence of pseudo-random numbers avoiding rapid intensity changes, as in the case of white noise, which is much closer in spirit to the real-world scenarios.

The fourth and last type of noise is real backgrounds (Fig. 3.9d). Instead of generating the noise, we use RGB-D images of real backgrounds in a similar way to [89]. Given a real image, we randomly sample a patch of a needed size and use it as a background for a synthetically generated model. This modality makes it simpler to bridge the domain gap and is especially useful when we know beforehand in what kinds of environments the objects are going to be located.

One of the drawbacks of the baseline method [29] is that the batches are generated and stored prior to execution. This means that at each epoch we use the same filled backgrounds over and over again, limiting the variability. To overcome this problem, in our implementation we generate batches online. At each iteration we fill the background of the chosen positive sample with one of the available noise modalities.

3.3 Evaluation

This section is devoted to the validation and evaluation of the implemented pipeline. We first perform a series of tests to evaluate the effect of the triplet loss with dynamic margin as well as the newly introduced modifications, e.g., in-plane rotations, background noise

Table 3.2: Comparison of the network trained without in-plane rotations (baseline) with the one trained using in-plane rotations (baseline+).

	Angular error			Classification
	10°	20°	40°	
Baseline	34.6%	63.8%	73.7%	81.9%
Baseline+	60%	93.2%	97%	99.3%

types. Next, we perform a series of tests to validate our multi-task extension combining manifold learning and regression.

3.3.1 Tests on In-plane Rotations

As we already know, the authors of the initially proposed method [29] do not take in-plane rotations into account and do not include them in training, which is, however, needed for working in real-world scenarios. This test compares the performances of two networks: the one that is trained with in-plane rotations and the other that is trained without them. The goal is to see how avoiding in-plane rotations in training affects the performance on the test data with in-plane rotations and also to demonstrate the ability of the network to perform well with an additional degree of freedom introduced.

Given the setup, we compare the two above mentioned networks, labeled as baseline (without in-plane rotations) and baseline+ (with in-plane rotations), and obtain the results shown in Table 3.2.

The evaluation is performed only for a single nearest neighbor. As can be seen from Table 3.2, one gets a radical improvement over the results shown by the first modality, which is not trained to account for in-plane rotations. The results also demonstrate a successful adaptation to an additional degree of freedom.

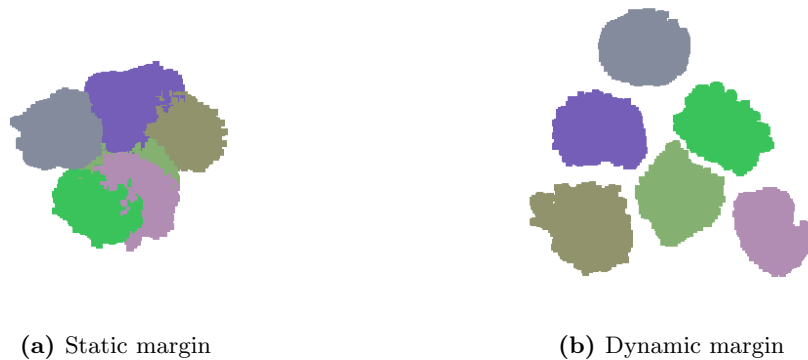


Figure 3.10: Test set samples mapped to a 3D descriptor space. Each color represents a separate object.

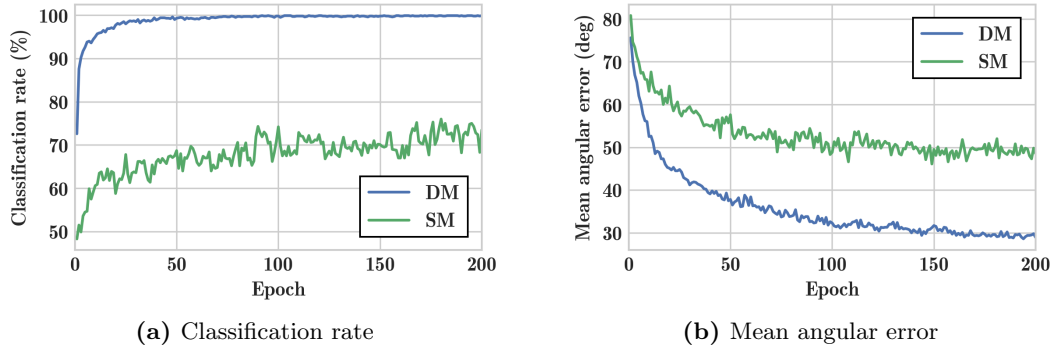


Figure 3.11: Comparison of triplet loss with (DM) and without (SM) dynamic margin for the 3D output descriptor.

3.3.2 Tests on the Dynamic Margin Triplet Loss

To evaluate the new loss function with dynamic margin, a set of tests comparing it with the old loss function was performed. Particularly, two tests were executed on six LineMOD objects (the lower amount is chosen for visualization purposes) using the best-performing training configurations for 3- and 32-dimensional output descriptors.

Figure 3.11 compares the classification rates (Fig. 3.11a) and mean angular errors for correctly classified samples (Fig. 3.11b) over the set of training epochs (one run through the training set) for two modalities, i.e., the networks trained using the loss function with static and dynamic margins. It is clearly seen from the results that the new loss function makes a huge difference to the output result. It enables the network to learn a better classification much faster in comparison to the original. While the dynamic margin modality reaches 100% classification accuracy very quickly, the old baseline fluctuates around 70%. Moreover, Fig. 3.11b shows that we get a lower angular error for around 30% more correctly classified samples.

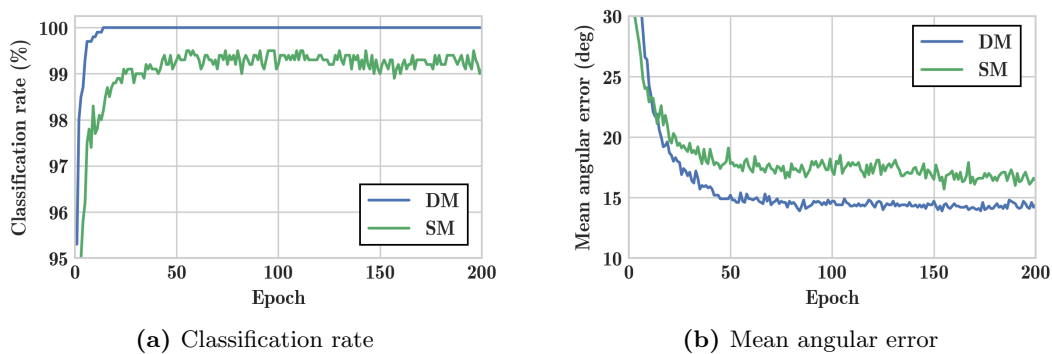


Figure 3.12: Comparison of triplet loss with (DM) and without (SM) dynamic margin for the 32D output descriptor.

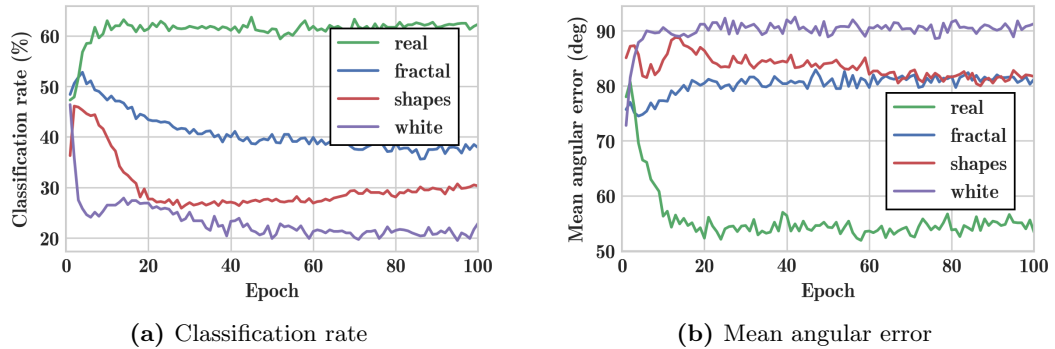


Figure 3.13: Comparison of four different background noise modalities without any real data used for training.

Figure 3.10 shows the test samples mapped to the 3D descriptor space using the descriptor network trained with the old (Fig. 3.10a) and new (Fig. 3.10b) loss functions. The difference in the degree the objects are separated is explicit: in the right figure, the objects are well-separated preserving the minimal margin distance, resulting in a perfect classification score; the left figure still shows well-distinguishable object structures, but they are placed very close together and overlap, causing the classification confusion that is quantitatively estimated in Fig. 3.11a.

In practice, however, we use dimensionally higher descriptor spaces, which improves both classification and angular accuracies. Fig. 3.12 shows the same charts as Fig. 3.11 but for a descriptor of a higher dimension, i.e. 32D. This results in a significant quality jump for both modalities, but the tendency stays the same: the new modality learns the classification much faster and provides a better angular accuracy for a larger number of correctly classified test samples.

3.3.3 Tests on Background Noise Types

Since we often do not have real RGB-D sequences on hand in real-world applications, but only 3D models provided, it would be beneficial to avoid using real data in training. The purpose of the following test is to show how well the network can adapt to the real data by only using the synthetic samples with artificially filled backgrounds in training. Specifically, we compare four different background noise modalities introduced in Section 3.2.2.4: white noise, random shapes, fractal noise, and real backgrounds.

Figure 3.13 shows the classification and pose accuracies for the four mentioned background noise modalities. The white noise modality shows the overall worst results, achieving around 21% of classification accuracy (Fig. 3.13a), a marginal improvement over randomly sampling objects from a uniform distribution.

By switching to the random shapes modality, we get better results and fluctuate around 30% of classification accuracy. The fractal noise modality shows the best results among the synthetic noise types and reaches up to 40% of recognition rate. However, the real backgrounds modality outperforms fractal noise in classification terms and,

moreover, shows better pose accuracy for a larger quantity of correctly classified samples (Fig. 3.13b). As a result, if we can collect images from environments similar to the test set, the best option is to fill the backgrounds with real images.

3.3.4 Tests on Input Image Channels

In this test, we demonstrate the influence of different input image modalities, i.e., depth, normals, and their combination, on the output accuracy. To do that, we train the network using the patches exclusively represented by the aforementioned channels.

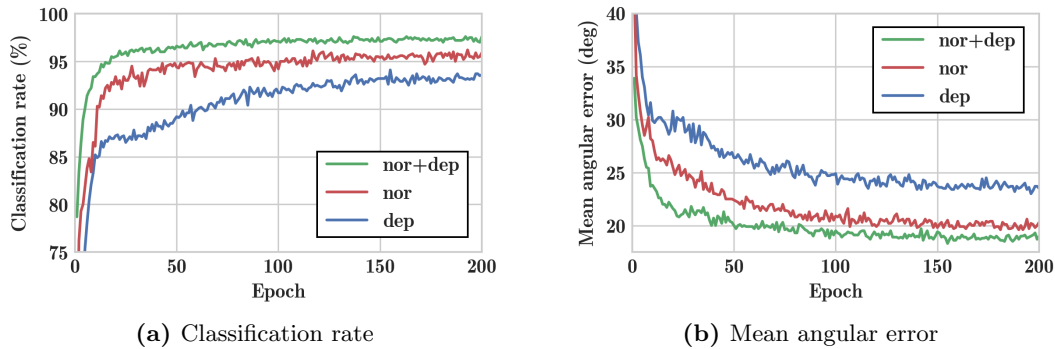


Figure 3.14: Comparison of three modalities representing different input image channels used in training.

Figure 3.14 demonstrates the classification rate and pose error charts for three different networks trained on three different combinations of input patch channels: depth, normals, and normals+depth. It can be observed that the network trained on surface normals performs better than the one trained on the depth maps only. This is beneficial since surface normals are generated entirely based on the depth maps and no additional sensor data is needed. Additionally, by combining the surface normals and depth channels into a single modality, we get even better results compared to using them separately. More importantly, the same effect holds true when RGB channels are added to the presented modalities.

3.3.5 Tests on Larger Datasets

The goal of this experiment is to see how well the algorithm generalizes to a larger number of models. In particular, we want to evaluate how the increased amount of models in training affects the overall performance. Since the LineMOD dataset has only 15 models available, the adapted BigBIRD dataset, which offers many more models, is used for this test.

Given one of the most powerful pipeline configurations, we have trained the network on 50 models of the BigBIRD dataset. After finishing the training, we achieved the results shown in Table 3.3. Table 3.3 shows the histogram of classified test samples for several tolerated angle errors. The results are encouraging: for 50 models each

3 3D Pose Estimation Based on Manifold Learning

represented by approximately 300 test samples, we get a classification of 98.7% and a very good angular accuracy, the significant improvement over the old loss function. As a result, this approach proves to scale well with respect to the number of models, making it suitable for industrial applications.

Table 3.3: Angular error histogram computed using the samples of the BigBIRD test set for a single nearest neighbor.

	Angular error			Classification
	10°	20°	40°	
SM	67.4%	79.6%	83.5%	85.4%
DM	67.7%	91.2%	95.6%	98.7%

3.3.6 Combining Manifold Learning and Regression

In this section, we evaluate the proposed multi-task extension that combines the strengths of the manifold learning and regression, and further improves the results of the proposed solution. For that purpose, we use a constrained setup. For all the following experiments we only consider the depth data as input modality and set the feature descriptor size to 64. As it is mentioned in [29], at some point increasing the feature descriptor size does not improve the methods performance anymore. As for regression, we found a similar effect during our experiments, in which we experienced $d = 64$ to be a good trade-off between the nearest neighbor and regression performance.

3.3.6.1 Multi-Task Learning vs Single-Task Learning

We first evaluate our multi-task extension on models trained on a different number of objects, for which the mean angular error is reported in Table 3.4. Overall, we

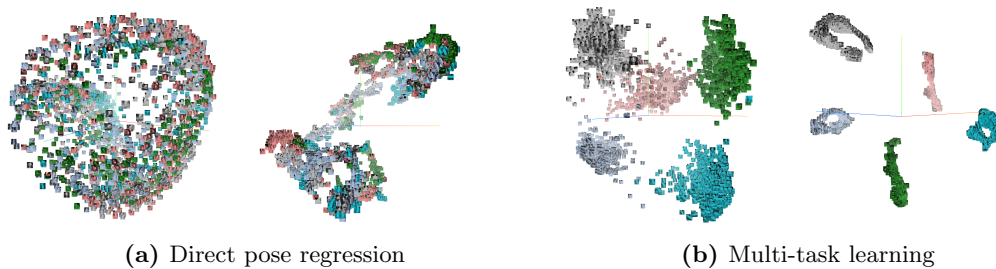


Figure 3.15: Feature space comparison. By using a multi-task learning framework, we are able to improve feature descriptors learned for object pose estimation. Depicted here is the feature visualization using left: PCA and right: t-SNE [2] for five objects of the LineMOD [3] dataset.

Table 3.4: Angular error of the baseline method (NN), regression (R) and our approach (Rmt , $NNmt$).

	15 Objects		10 Objects		5 Objects	
	Mean (Median) \pm Std	Class	Mean (Median) \pm Std	Class	Mean (Median) \pm Std	Class
NN	25.29° (11.76°) \pm 40.75°	92.46%	19.98° (10.58°) \pm 34.78°	92.56%	24.19° (10.72°) \pm 43.34°	99.31%
$NNmt$	17.70° (11.59°) \pm 25.78°	97.07%	14.74° (11.53°) \pm 15.04°	97.50%	13.05° (10.29°) \pm 15.19°	99.90%
R	38.23° (26.16°) \pm 34.65°	-	29.17° (20.69°) \pm 28.03°	-	22.07° (15.56°) \pm 24.40°	-
Rmt	27.28° (19.25°) \pm 27.26°	-	23.08° (17.56°) \pm 21.25°	-	19.16° (13.80°) \pm 21.54°	-

experience a significant improvement in performance for both regression and nearest neighbor search accuracy. During training, the usually more difficult regression task, seems to be optimized by additionally focusing on learning a meaningful embedding, improving the mean angular error by 28.8%. Since poses as well as objects are already well-distinguished and the feature descriptors separated by the triplets and pair loss functions, regression can more easily be learned.

As for the performance of nearest neighbor search, we observe an improvement in robustness and accuracy of our multi-task learning framework compared to the single-task baseline NN . The standard deviation as well as the mean angular error of the multi-task model $NNmt$ decreases significantly, making the method more robust. Here we can report a relative improvement of 30% for the mean angular error while training on the full LineMOD dataset, meaning fifteen objects.

Both regression and nearest neighbor method benefit from jointly learning robust features and poses. Which model to choose now becomes a trade-off between time complexity and accuracy, which we will address further in Section 3.3.6.4.

3.3.6.2 Influence of Network Architecture

Additionally, to explore the multi-task pipeline performance using network architectures with varying depths, we run our model using the network architecture described in [90]. This architecture is two layers deeper and removes max pooling layers by including convolutional layers with stride two. Stated by the authors of [29], a deeper network architecture did not seem to improve the accuracy of the method further, which we experienced in our test as well, however by using our multi-task learning framework and testing on a deeper network architecture we find that we can improve the pose estimation accuracy even further. Here we are able to achieve the results seen in Table 3.5, abbreviated as $NNmt_{deeper}$. We report a relative improvement of 7.2% using nearest neighbor search and 9.0% in the mean angular error of our regression results by using a deeper network architecture, while training on the full LineMOD dataset. We believe that by optimizing the network further, we can achieve even better regression accuracy.

3.3.6.3 Feature Visualization

As we have shown in Table 3.5, apart from the improvement in accuracy for pose regression, we also experienced an increase in the performance of the nearest neighbor pose

3 3D Pose Estimation Based on Manifold Learning

Table 3.5: Comparison between the classification and angular accuracy of the baseline method, *NN*, and our results on 15 objects of the LineMOD dataset.

	Angular error			Classification
	10°	20°	40°	
<i>NN</i>	35.98%	71.56%	82.72%	92.46%
<i>NNmt</i>	37.89%	79.61%	92.27%	97.07%
<i>NNmt_{deeper}</i>	41.32%	82.52%	93.51%	97.26%

retrieval. Our results demonstrate that the feature descriptors provided by the model trained on both tasks seem to be more discriminative. To analyze the resulting feature descriptors, we visualize the descriptors in the lower dimensional 3D-space using PCA and t-SNE.

For t-SNE, we use a perplexity of 100, learning rate of 10 until convergence. Using PCA, the variance including the best three components resulted in 53.2%. The resulting clusters for five objects can be seen in Fig. 3.15. We observe that in both cases the object classes are nicely distinguished using feature descriptors obtained by the multi-task approach.

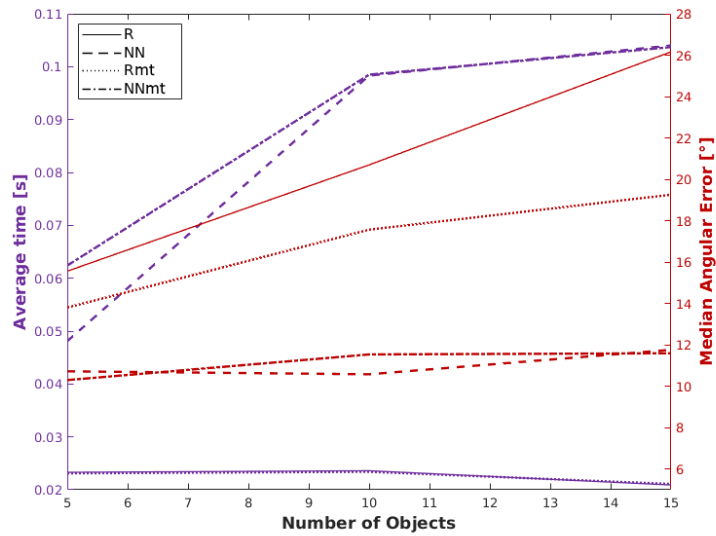


Figure 3.16: Average time and median angular error of nearest neighbor pose retrieval, regression and our approach.

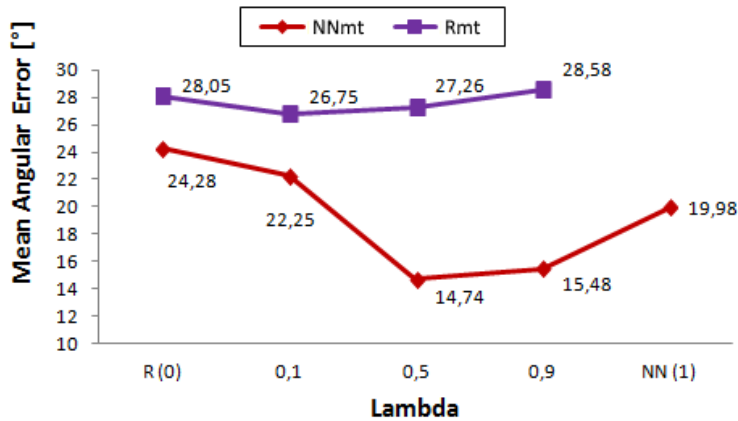


Figure 3.17: Sensitivity of λ in our loss function $L_{mtl} = (1 - \lambda)L_{reg} + \lambda L_{nn}$. Depicted is the influence of different weighting parameters on the mean angular error for regression as well as nearest neighbor pose retrieval.

3.3.6.4 Scalability

Here, we analyze the time complexity and accuracy of our models at different number of objects. Figure 3.16 shows the mean time of our models and the corresponding angular error. The mean time for regression is calculated as one forward pass of the neural network. For nearest neighbor methods only the matching time is tracked. To obtain the total time needed, the time of one forward pass should be added to the shown results for matching. One can see nicely that regression has a constant time, regardless of how many objects are used, whereas for the nearest neighbor search the time increases with additional objects. Depending on the application, this and the drop in accuracy for additional objects should be taken into account.

3.3.6.5 Sensitivity to Regularization Parameter λ

Since our multi-task loss function includes a regularization parameter λ balancing the two components of regression and manifold learning, we conducted experiments on the sensitivity of this parameter using the LineMOD dataset. By choosing different values for λ and thus weighting either the L_{nn} loss or the pose loss L_{reg} more, we find that the results improve for nearest neighbor pose retrieval, if the two terms are equally weighted, and decreases when focusing more on the regression loss. Regarding regression, we observe similar results: improvement when additionally focusing on the L_{nn} loss, enhancing the feature representation and decrease, if the model is only trained on regression. Nevertheless, it can be seen that regression has a much stronger influence on the nearest neighbor pose retrieval in terms of performance than the other way around.

The results, depicted in Fig. 3.17, emphasize our assumption that the two terms are beneficial to one another, i.e., both features and pose regression are mutually optimized.

Note that we omit the regression result when the model was only trained on the feature representation since in this case the regression layer was excluded from training.

3.4 Conclusion

In this chapter, the method first introduced in [29] was improved in terms of its learning speed, robustness to clutter, and usability in real-world scenarios. We have implemented a new loss function with dynamic margin that allows for a faster training and better accuracy. Moreover, we introduced in-plane rotations (present in real-world scenarios) and new background noise types (to better mimic the real environments), and evaluated the performance of different image modalities including surface normals.

Next, we have presented a multi-task learning extension that combines manifold learning with direct pose regression. As a result, we were able to improve both the nearest neighbor pose retrieval as well as the direct pose regression by a large margin when compared to the single-task methods. Subsequently, we conducted a detailed analysis of the feature descriptor learning, regression and the effect that both tasks have on each other in the context of object pose estimation.

4 6D Pose Estimation Based on Dense Correspondences

In this chapter, we present a deep learning method for 3D object detection and 6D pose estimation from RGB images. Our method, named DPOD (Dense Pose Object Detector), estimates dense multi-class 2D-3D correspondence maps between an input image and available 3D models. Given the correspondences, a 6DoF pose is computed using PnP and RANSAC. An additional RGB pose refinement of the initial pose estimates is performed using a custom deep learning-based refinement scheme. Our results and comparison to a vast number of related works demonstrate that a large number of correspondences is beneficial for obtaining high-quality 6D poses both before and after refinement. Unlike other methods that mainly use real data for training and do not train on synthetic renderings, we perform evaluation on both synthetic and real training data demonstrating superior results before and after refinement when compared to all recent 6D detectors. While being precise, the presented approach is also real-time capable.

4.1 Introduction

Inspired by the DensePose method of Gueler et al. [91], which estimates dense correspondences between a human body model and humans in the image, we propose a novel 3D object detector and pose estimator based on dense 2D-3D correspondences. Unlike DensePose for humans, which requires a sophisticated annotation tool and enormous annotation efforts, our method is annotation-free and only requires creation of arbitrary UV texture maps of the objects, that we do automatically – mainly by spherical projections. The two key elements of our approach are: the pixel-wise prediction of the multi-class object ID masks and classification of correspondence maps that directly provide a relation between image pixels and 3D model vertices. In this way, we end up with a large number of pixel-wise correspondences, which allow for a much better pose estimation than, for example, 9 regressed virtual points of the object’s bounding box as in YOLO6D [40].

In addition to this, we introduce a deep learning-based pose refinement network that takes initial poses estimated with our DPOD detector and enhances them. The proposed refinement approach builds on the successes of [9, 56], but is shown to be faster, simpler to train, able to be trained both on synthetic and real data, and it outperforms the former solutions in terms of pose quality. We demonstrate that even our poses, which are already of high quality, can be further improved with our refiner.

We experimented by training our detector with only synthetic and only real images. In both cases, our unified method, named DPOD, composed of the dense pose detector

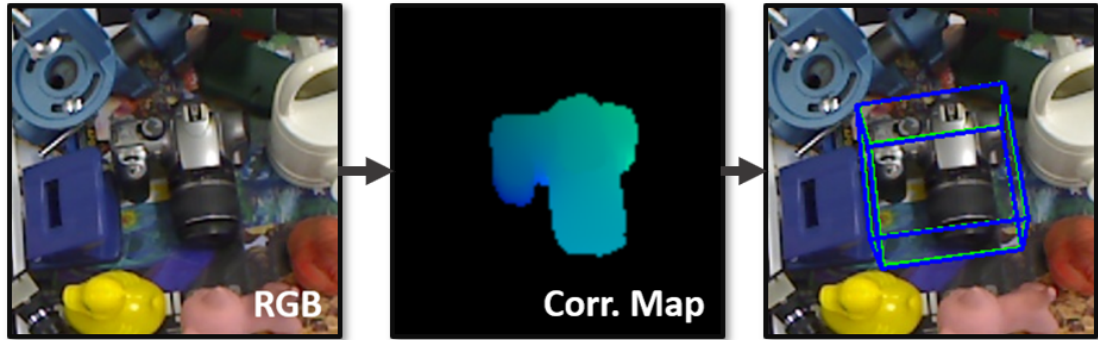


Figure 4.1: Example output of the DPOD method. Given a single RGB image, we regress its ID mask and its 2D-3D correspondences. PnP+RANSAC is then applied to estimate the final pose. The green bounding box shows the ground truth pose, while the blue one corresponds to the estimated pose. The almost perfect overlap of the bounding boxes indicates that estimations are very accurate.

and the refiner outperforms other related works. Dense correspondences not only allow for standard PnP and RANSAC to estimate accurate poses, but also pave the way for a successful pose refinement. For the models trained on real data, one iteration of refinement is enough to even outperform the related works using the depth-based ICP refinement.

In the remainder of this chapter, we first introduce our approach explaining data preparation, training, architectures and pose refinement, and then present an exhaustive experimental validation and comparison with related works, where we demonstrate the superiority of our approach.

4.2 Methodology

In this section, we first discuss the training data preparation steps, followed by the neural network architecture and loss functions used, as well as the pose estimation step from dense correspondences. Finally, we describe our deep learning model-based pose refiner.

4.2.1 Data Preparation

Most recent RGB-based detectors can be divided in two groups based on the type of data they use for training: synthetic-based and real-based. The first group of methods, e.g., SSD6D [50] and AAE [92], makes use of textured 3D models, usually provided with the public 6D pose detection datasets. The objects are rendered from different viewpoints, producing a synthetic training set. The methods of the second group on the other hand, e.g., BB8 [38], YOLO6D [40], PVNet [93], use the training split of the real dataset. They utilize ground truth poses provided with the dataset and compute object masks to crop the objects from real images producing a training set.

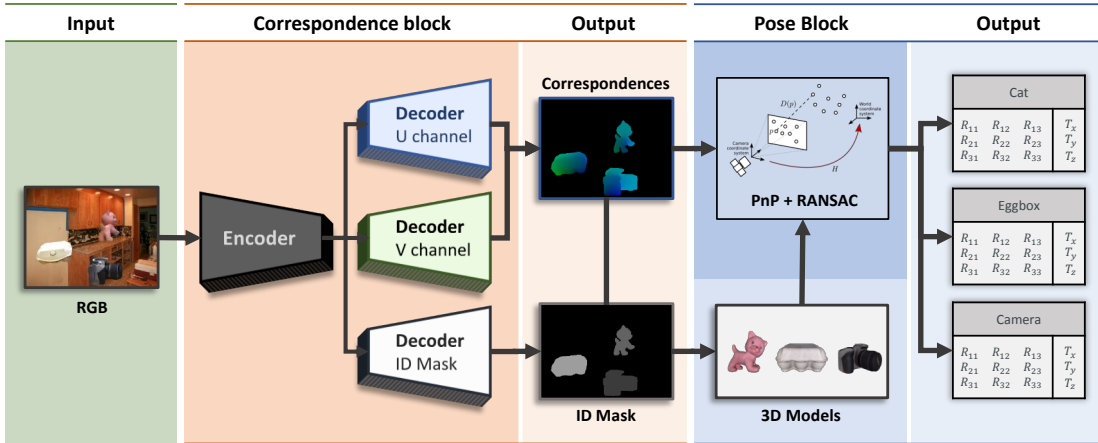


Figure 4.2: Pipeline description. Given an input RGB image, the correspondence block, featuring an encoder-decoder neural network, regresses the object ID mask and the correspondence map. The latter one provides us with explicit 2D-3D correspondences, whereas the ID mask estimates which correspondences should be taken for each detected object. The respective 6D poses are then efficiently computed by the pose block based on PnP+RANSAC.

Both types of data generation have their pros and cons. When real images sufficiently covering the object are available, it is more advantageous to use them for training. The reason is that their close resemblance to the actual objects allows for faster convergence and better results. However, training on real images biases the detector to light conditions, poses, scales and occlusions present in the training set, which might lead to problems with generalization in new environments. When, however, no pose annotations are available, which can often be the case since acquiring pose annotations is an expensive process, we are left with 3D models of the objects. With synthetic renderings, one can produce a virtually infinite number of images from different viewpoints. Despite being advantageous in terms of the pose coverage, one has to deal with the domain gap problem severely hindering the performance if no additional data augmentation is applied. Potentially, one can benefit from the advantages of both data types by mixing real and synthetic data in the training set. Therefore, approaches which can be trained on both types of data are desirable. Since our pipeline is not data-specific, we show how to generate the training data for both scenarios.

Synthetic Training Data Generation. Given 3D models of the objects of interest, the first step is to render them from different poses sufficiently covering the object. The poses are sampled from the half-sphere above the object. Additionally, in-plane rotations of the camera around its viewing direction from -30 to 30 degrees are added. Then, for each of the camera poses, an object is rendered on a black background and both RGB and depth channels are stored.

Having the renderings at hand, we use a generated depth map as a mask to define a tight bounding box for each generated rendering. Cropping the image with this bounding box position, we store RGB patches, masks separating them from the background, and the camera poses. At this point, we have everything ready for the online augmentation stage, which is described in the later subsection. This step of data preparation is identical for the detector and for the refinement pipelines.

Real Training Data Generation. In this case, an available dataset with pose annotations is divided into non-overlapping train and test subsets. Here, we follow the protocol defined by BB8 [38] and YOLO6D [40] and use 15% of data for training and the rest 85% for evaluation. Poses are selected such that the relative orientation between them is larger than a certain threshold. This approach guarantees that selected poses cover the object from all sides. For training the detector, objects are cut out from the original image using the provided mask and then stored as patches for the online augmentation stage. Additional in-plane rotations are added to artificially simulate new poses. For training the refinement, objects are left as they are.

4.2.1.1 Correspondence Mapping

To be able to learn dense 2D-3D correspondences, each model of the dataset is textured with a correspondence map (see Fig. 4.3). A correspondence map is a 2-channel image with values ranging from 0 to 255. Objects are textured using either simple spherical or cylindrical projections. Once textured, we get a bijective mapping between the model’s vertices and pixels on the correspondence map. This provides us with easy-to-read 2D-3D correspondences since given the pixel color, we can instantaneously estimate its position on the model surface by selecting the vertex with the same color value. For convenience, we call the copies of the original models textured with the correspondence map *correspondence models*. Given the predicted correspondence map, we estimate the object pose with respect to the camera using the pose estimation block, which is described later. Similar to the synthetic or real data generation steps, we render correspondence

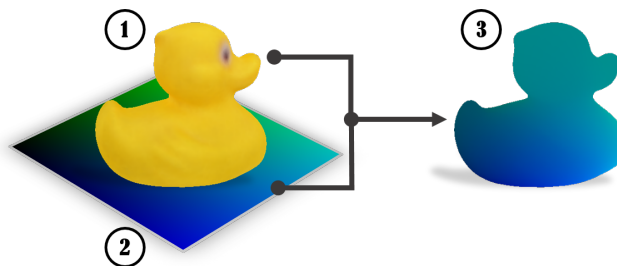


Figure 4.3: Correspondence model. Given a 3D model of interest (1), we apply a 2 channel correspondence texture (2) to it. The resulting correspondence model (3) is then used to generate GT maps and estimate poses.

models under the same poses as for training data and store correspondence patches for each RGB patch.

4.2.1.2 Online Data Generation and Augmentation

Detection and Pose Estimation. The final stage of data preparation is the online data generation pipeline, which is responsible for providing full-sized RGB images ready for training. Generated patches (real or synthetic) are rendered on top of images from MS COCO dataset [94] producing training images containing multiple objects. It is an important step, which ensures that the detector generalizes to different backgrounds and prevents it from overfitting to backgrounds seen during training. Moreover, it forces the network to learn the model’s features needed for pose estimation rather than to learn contextual features which might not be present in images when the scene changes. This step is performed no matter whether the training is being done with synthetic or real patches. We additionally augment the RGB image by random changes in brightness, saturation, and contrast, and by adding Gaussian noise. Moreover, object ID masks and correspondence patches are also rendered on top of the black background in order to generate ground truth correspondence maps. An object ID mask is constructed by assigning a class ID number to each pixel that belongs to the object.

Pose Refinement. In the case of pose refinement, pairs of images containing the object in the current (searched) pose and in the predicted pose are provided to the network. The final stage of data preparation differs considerably depending on the type of data used. In case of synthetic data, images are generated by in-painting objects on random backgrounds in a current pose. A crucial part of the augmentation is to add random light sources for every image. If real images are used for training, no in-painting is performed. In any case, produced images are further augmented as discussed above. Then a random pose is sampled around the current pose simulating the predicted pose from the detector, which will be used as an original guess of the poses to be refined. It is crucial to choose the proper prior distribution from which distorted poses are sampled.

4.3 Dense Object Detection Pipeline

Our inference pipeline is divided into two blocks: the correspondence block and the pose block (see Fig. 4.2). In this section, we provide their detailed description.

Correspondence Block. The correspondence block consists of an encoder-decoder convolutional neural network with three decoder heads which regress the ID mask and dense 2D-3D correspondence map from an RGB image of size $320 \times 240 \times 3$. The encoder part is based on a 12-layer ResNet-like [95] architecture featuring residual layers that allow for faster convergence. The decoders upsample the feature up to its original size using a stack of bilinear interpolations followed by convolutional layers. However, in principle the proposed method is agnostic to a particular choice of encoder-decoder architecture. Any other backbone architectures can be used without any need to change

the conceptual principles of the method. For the ID mask head the output is a $H \times W \times O$ tensor, where H and W are the height and width of the original input image and O is the number of objects in the dataset plus one additional class for background. Similar to the ID mask head, the two correspondence heads regress tensors with the following dimensions $H \times W \times C$, where C stands for the number of unique colors of the correspondence map, i.e., 256. Each channel of the output tensors stores the probability values for the class corresponding to the channel number. Once tensors are regressed, we store them as single channel images where each pixel stores the class with the maximal estimated probability, forming the ID mask, U and V channels of the correspondence image.

Formulating color regression problem as discrete color class classification problem proved to be useful for much faster convergence and for the superior quality of 2D-3D matches. Initial experiments on direct coordinate regression showed very poor results in terms of correspondence quality. The main reason for the problem was the infinite continuous solution space, i.e., $[-1; 1]^3$, where 3 is the number of dimensions and $[-1, 1]$ is the normalized coordinate range of a 3D model. Classification of the discretized 2D correspondences allowed for a huge boost of the output quality by dramatically decreasing the output space (now 256^2 , where 256 is the size of a single UV map dimension). Moreover, this parametrization also ensures that 3D points of the predicted correspondences always lie on the object surface.

The network parameters are optimized subject to the composite loss function:

$$\mathcal{L} = \alpha\mathcal{L}_m + \beta\mathcal{L}_u + \gamma\mathcal{L}_v, \quad (4.1)$$

where \mathcal{L}_m is the mask loss, and \mathcal{L}_u and \mathcal{L}_v are the losses responsible for the quality of the U and V channels of the correspondence image. α, β , and γ are weight factors set to 1 in our case. Both \mathcal{L}_u and \mathcal{L}_v losses are defined as multi-class cross-entropy functions, whereas \mathcal{L}_m uses the weighted version of it.

Pose Block. The pose block is responsible for the pose prediction. Given the estimated ID mask, we can observe which objects were detected in the image and their 2D locations, whereas the correspondence map maps each 2D point to a coordinate on an actual 3D model. The 6D pose is then estimated using the Perspective-n-Point (PnP) [96] pose estimation method that estimates the camera pose given correspondences and intrinsic parameters of the camera. Since we get a large set of correspondences for each model, RANSAC is used in conjunction with PnP to make camera pose prediction more robust to possible outliers. For the results presented in the evaluation section, for each pose we run 150 RANSAC iterations with the reprojection error threshold set to 1.

4.4 Deep Model-Based Pose Refinement

The proposed pose refiner is a natural extension of refiners presented in [56, 9] and relies on the strengths of both approaches. Similar to [56, 50, 97] we exploit an idea of using a network already pre-trained on ImageNet as a backbone architecture. Analogous to the

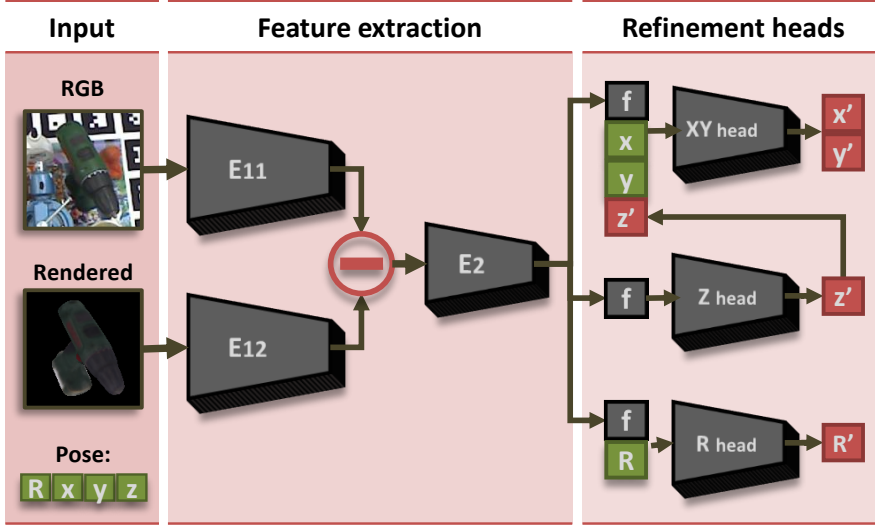


Figure 4.4: Refinement architecture. The network predicts a refined pose given an initial pose proposal. Crops of the real image and the rendering are fed into two parallel branches. The difference of the computed feature tensors is used to estimate the refined pose.

detector, we used a ResNet-based architecture. Similar to [9], our loss function for pose estimation is the ADD measure with a more robust L_1 norm:

$$m = \text{avg}_{\mathbf{x} \in \mathcal{M}_s} \left\| (\mathbf{R}\mathbf{x} + \mathbf{t}) - (\hat{\mathbf{R}}\mathbf{x} + \hat{\mathbf{t}}) \right\|_1, \quad (4.2)$$

representing the vertex to vertex distance between the object in a ground truth pose and predicted pose. \mathbf{R} , \mathbf{t} denote the ground truth pose rotation and translation, whereas $\hat{\mathbf{R}}$ and $\hat{\mathbf{t}}$ denote the predicted transformation; \mathcal{M}_s is a set of points sampled from the CAD model. Points are resampled at every iteration. The number of sampled points was limited to ten thousand in order to ensure the efficiency of training iterations and reasonable memory consumption.

In Fig. 4.4 we show a schematic representation of the refiner. In order to be able to benefit from the network weights pretrained on ImageNet, the network has two parallel input branches, each composed of the first five ResNet layers. These layers are initialized from the pre-trained network. One branch receives an input image patch (E_{11}), while the other (E_{12}) one extracts features from the rendering of the object in the predicted pose. Then features \mathbf{f}_r and \mathbf{f}_s from these two networks are subtracted and fed into the next ResNet block (E_2) producing the feature vector \mathbf{f} . If the refinement is trained on synthetic data, it is essential to keep the first five layers unchanged and use them as the feature extractor as was shown in [98, 97, 56]. Freezing the branch that extracts features from object renderings is unnecessary as it always operates on synthetic data. The network ends with three separate output heads: one for regressing the rotation, one for regressing the translation in X and Y directions, and one for regressing the translation in

Z direction. We opted for three separate heads as the scale of their outputs is different. Each head is implemented as two fully connected layers.

Rotation is always represented in the object coordinate system, which ensures that identically looking objects have the same rotation and that the network does not have to learn a more complicated transformation which arises if the world coordinate system is used. The first layer of the rotation regression head takes the feature vector \mathbf{f} produced by ResNet and adds four values, which are the quaternion representing an initial rotation. The second layer takes the output of the previous one, stacks with the initial quaternion and outputs the final rotation.

The head responsible for the regression of X and Y translations operates in the coordinate system of the image rather than in the full 3D space, which significantly restricts the space of possible solutions. Similar to the rotation head, the XY regression head takes the initial 2D location of the object as input and refines it. Additionally, it takes the refined prediction of Z translation.

Weights of the fully connected layers are initialized in such a way that for the 0th iteration the network just outputs the input pose, and then during training learns how to refine those values. That significantly increases stability and speed of the training procedure as the network produces meaningful results from the very start.

4.5 Training Details

Our pipeline is implemented using the Pytorch deep learning framework. All the experiments were conducted on an Intel Core i7-6900K CPU 3.20GHz with NVIDIA TITAN X (Pascal) GPU. To train our method, we used the ADAM solver with a constant learning rate of 3×10^{-4} and weight decay of 3×10^{-5} .

When training on synthetic data, the problem of domain adaptation becomes one of the main challenges. Training the network without any prior parameter initialization makes it impossible to generalize to the real data. The easy solution to this problem was proposed in several works, including [97, 56], where they freeze the first layers of the network trained on a large dataset of real images, e.g., ImageNet [99] or MS COCO [94], for the object classification task. The common observation that the authors conclude is that these layers, learning low-level features, very quickly overfit to the perfect object renderings. We follow this setup, and freeze the first five layers of our encoder initialized with the weights of the same network pretrained on ImageNet. Last but not least, we found it crucial for the performance of the detector to use various light sources during the rendering of synthetic views to account for changing light conditions and shadows in the real data.

4.6 Evaluation

In this section we evaluate our algorithm in terms of its pose and detection performance, as well as its runtime, and compare it with the state of the art RGB detector solutions.

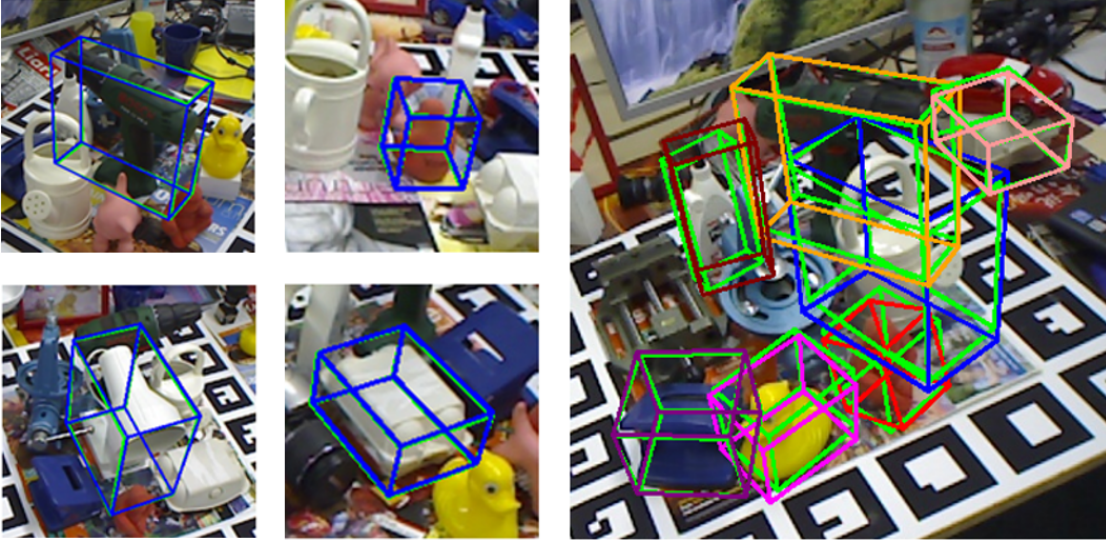


Figure 4.5: Qualitative results. Poses predicted with the proposed approach on (a) the LineMOD dataset and (b) the OCCLUSION dataset. Green bounding boxes correspond to ground truth poses, bounding boxes of other colors to predicted poses. For both datasets predicted poses are very close to correct poses.

4.6.1 Datasets

All experiments were conducted on LineMOD [58] and OCCLUSION [100] datasets, as they are the standard datasets for evaluation of object detection and pose estimation methods. The LineMOD dataset consists of 13 sequences, each containing ground truth poses for a single object of interest in a cluttered environment. CAD models for all the objects are provided as well. The OCCLUSION dataset is an extension of LineMOD, suitable for testing how well detectors can deal with occlusions. Although it comprises only one sequence, all visible objects from the LineMOD dataset are supplied with their poses.

4.6.2 Evaluation Metrics

We evaluate the quality of 6DoF pose estimation following the procedure suggested at SSD6D [50] also used in other papers. Analogously to other related papers [40, 50, 93, 101], we measure the accuracy of pose estimation using the *ADD score* [58]. ADD is defined as an average Euclidean distance between model vertices transformed with the predicted and the ground truth pose. More formally it is defined as follows:

$$m = \text{avg}_{\mathbf{x} \in \mathcal{M}} \left\| (\mathbf{R}\mathbf{x} + \mathbf{t}) - (\hat{\mathbf{R}}\mathbf{x} + \hat{\mathbf{t}}) \right\|_2, \quad (4.3)$$

where \mathcal{M} is a set of vertices of a particular model, \mathbf{R} and \mathbf{t} are the rotation and translation of a ground truth transformation whereas $\hat{\mathbf{R}}$ and $\hat{\mathbf{t}}$ correspond to those of an

Table 4.1: Pose estimation performance. Comparison of our approach to the other RGB detectors on the LineMOD dataset. The table reports the percentages of correctly estimated poses w.r.t. the ADD score. Among the methods trained on synthetic data, our method shows the best results significantly surpassing the former state-of-the-art. The variant of our method trained on real data again demonstrates outstanding performance outperforming most of the competitors. Moreover, our new refinement pipeline improves the estimated poses even further and shows the best overall results.

Train data	Synthetic			+ Refinement		Real				+ Refinement	
Object	SSD6D [50]	AAE [92]	Ours	SSD6D [56]	Ours	YOLO6D [40]	PoseCNN [101]	PVNet [93]	Ours	DeepIM [9]	Ours
Ape	2.6	3.96	37.22	-	55.23	21.62	-	43.62	53.28	77.0	87.73
Bvise	15.1	20.92	66.76	-	72.69	81.80	-	99.90	95.34	97.5	98.45
Cam	6.1	30.47	24.22	-	34.76	36.57	-	86.86	90.36	93.5	96.07
Can	27.3	35.87	52.57	-	83.59	68.80	-	95.47	94.10	96.5	99.71
Cat	9.3	17.90	32.36	-	65.10	41.82	-	79.34	60.38	82.1	94.71
Driller	12.0	23.99	66.60	-	73.32	63.51	-	96.43	97.72	95.0	98.80
Duck	1.3	4.86	26.12	-	50.04	27.23	-	52.58	66.01	77.7	86.29
Eggbox	2.8	81.01	73.35	-	89.05	69.58	-	99.15	99.72	97.1	99.91
Glue	3.4	45.49	74.96	-	84.37	80.02	-	95.66	93.83	99.4	96.82
Holep.	3.1	17.60	24.50	-	35.35	42.63	-	81.92	65.83	52.8	86.87
Iron	14.6	32.03	85.02	-	98.78	74.97	-	98.88	99.80	98.3	100
Lamp	11.4	60.47	57.26	-	74.27	71.11	-	99.33	88.11	97.5	96.84
Phone	9.7	33.79	29.08	-	46.98	47.74	-	92.41	74.24	87.7	94.69
Mean	9.1	28.65	50	34.1	66.43	55.95	62.7	86.27	82.98	88.6	95.15

estimated transformation. The ADD metric can be extended in order to handle symmetric objects as in [58]:

$$m = \text{avg} \min_{\mathbf{x}_2 \in \mathcal{M}} \min_{\mathbf{x}_1 \in \mathcal{M}} \left\| (\mathbf{R}\mathbf{x}_1 + \mathbf{t}) - (\hat{\mathbf{R}}\mathbf{x}_2 + \hat{\mathbf{t}}) \right\|_2 \quad (4.4)$$

Instead of measuring distance from a predicted location of each particular model’s vertex to its ground truth location, it suggests to take the closest vertex of the model transformed with the ground truth transformation.

Conventionally, a pose is considered correct if ADD is smaller than the 10% of the model’s diameter. The accuracy of pose estimation is reported as the percentage of correctly estimated poses.

4.6.3 Single Object Pose Estimation

Results of the pose estimation experiments on the LineMOD dataset are reported in Table 4.1. We separately compared our method trained either on real data or on synthetic data. The table provides the comparison of deep learning-based refinement pipelines as well. The left-hand side of the table reports the accuracy of pose estimation as percentages of poses which are correct according to the ADD measure for the training done on synthetic data. If no refinement is used, our approach outperforms all other approaches by a significant margin on the majority of the objects. Moreover, the average percentage of correctly estimated poses (50%) is significantly higher than 28.65% of the second best approach. The accuracy gap is more prominent on small objects such as the

Table 4.2: Pose estimation for multiple objects. Comparison of our approach on real data to the other RGB detectors on the OCCLUSION dataset. The table reports percentages of correctly estimated poses w.r.t. the ADD score.

Method	YOLO6D [40]	PoseCNN [101]	SSD6D+Ref [56]	HMap [102]	PVNet [93]	Ours	Ours+Ref
Mean	6.42	24.9	27.5	30.4	40.77	32.79	47.25

ape and duck. The availability of a large number of 2D-3D correspondences ensures that the performance of our method is 5 times better than SSD6D’s and almost 2 times better than AAE’s. If deep learning-based refinement is used, we significantly outperform [56] with 66.43% of correct poses against 34.1%.

If trained on real data, our method is the second best after [93]. The right-hand side of Table 4.1 compares the proposed approach to the previous deep learning-based ones. If no refinement is used, the proposed approach outperforms PoseCNN and YOLO6D by a significant margin, while performing on par with PVNet on most of the objects. On average, we are better than PoseCNN by 31%, YOLO6D by 23.57%. Again, our approach uses RGB data exclusively and does not rely on depth data. Fig. 4.5 provides a visual comparison of ground truth poses versus predicted poses. Poses are visualized as projections of 3D bounding boxes of models in given poses on top of a test image. In comparison to deep learning-based refinement of [9], we perform on average better by 6.55% reaching 95.15% of correct poses. When DeepIM was applied to the poses predicted by the proposed approach, ADD improved to 91.8% which is better than the original 88.6% reported in their paper, but still worse than the result of our refiner.

In conclusion, the proposed detector achieves state-of-the-art results surpassing other detectors by a large margin on synthetic data and performs either much better or comparable to the other detectors on real data. The proposed refinement clearly outperforms all the competitors both on real and synthetic data. Pose quality varies from object to object, but in general poses are significantly better for larger objects since there are more 2D-3D correspondences available. On the other hand, simplicity of the proposed approach also makes it quick. On average our detector performs at 33 FPS. The runtime can be adjusted by changing the number of RANSAC iterations, as it is the bottleneck of the pipeline. One iteration of the refinement takes 5ms, excluding the rendering time, which heavily depends on the renderer used. Two refinement iterations suffice for synthetic data, one iteration – for real data.

Table 4.3: Detection performance for multiple objects. Comparison of the state-of-the-art mean average precision (mAP) scores on the OCCLUSION dataset.

Method	SSD6D [50]	YOLO6D [40]	Brachmann [103]	Ours
mAP	0.38	0.48	0.51	0.48

Table 4.4: RANSAC iterations test. The effect of the number of RANSAC iterations on the overall ADD score.

RANSAC #	5	25	50	100	150	200	250	350	500
ADD w/o ref	59.15	76.95	80.15	82.12	82.98	83.44	83.79	84.33	84.66
ADD w/ ref	80.45	92.59	93.88	94.79	95.15	95.31	95.39	95.38	95.39
RANSAC ms	2	6	10	17	23	28	33	42	54

4.6.4 Multiple Object Pose Estimation

Performance evaluation of the proposed detector in cases when the number of objects to detect increases and when severe occlusions are present was conducted on the OCCLUSION dataset [100]. Accuracy of object detection on the OCCLUSION dataset is conventionally reported in terms of mean average precision (mAP). The confidence score is computed based on the RANSAC inlier proportion as confidence, rendering the final score of 0.48, which is comparable the best result on this dataset (see Table 4.3). Table 4.2 demonstrates ADD scores for various detectors on the OCCLUSION dataset. Before the refinement, the proposed detector shows very competitive results in comparison to other detectors. After the refinement, the proposed approach performs substantially better and achieves the best results.

4.6.5 Ablation Study

In this section, we provide an ablation study of the method’s components. In particular, we report the runtimes, study the effect of RANSAC iterations, evaluate the correspondence quality, and compare our refiner with the state of the art.

4.6.5.1 RANSAC Iterations

The number of RANSAC iterations crucially influences the quality of predicted poses. We ended up using 150 iterations as it yielded the best trade off between quality and runtime. The larger amount of iterations generally did not improve the results significantly, but resulted in longer execution times (see Table 4.4). Additionally, the ADD

Table 4.5: Runtime comparison. Time-efficiency of our approach with respect to the other state-of-the-art approaches.

Method	Frames per second	Refinement
AAE [92]	4	200 ms/object
SSD6D [50]	10	24 ms/object
PVNet [93]	25	-
Ours	33	5 ms/object
YOLO6D [40]	50	-

Table 4.6: Runtime analysis. Runtime of the proposed approach for all models of the LineMOD dataset.

Model	PnP + RANSAC (ms)	Total (ms)	FPS
Ape	7	20	50
Benchvise	40	51	20
Cam	35	49	20
Can	30	44	23
Cat	20	33	30
Driller	26	40	25
Duck	4	16	63
Eggbox	9	23	43
Glue	5	17	59
Holepuncher	20	31	32
Iron	34	48	21
Lamp	40	54	19
Phone	31	45	22
Average	23	36	33

scores after one iteration of the proposed refinement are provided. They show that even 25 iterations of RANSAC are enough to beat the state-of-the-art results if the refinement is used. More iterations of RANSAC do not result in the considerable increase of pose quality.

4.6.5.2 Runtime Analysis

In Table 4.6, we provide the runtimes of the proposed approach for all models of the LineMOD dataset. The total runtime consists of the time needed for PnP and approximately 13 ms for all the auxiliary tasks: the network’s forward pass, post-processing of predicted segmentation, and computation of 2D-3D correspondences. Table 4.5 provides comparison of the runtime of our detector with all the main competitors mentioned in the paper. All the experiments were conducted on an Intel Core i7-6900K CPU 3.20GHz with NVIDIA TITAN X (Pascal) GPU.

4.6.5.3 Correspondence Quality

Here, we demonstrate the quality of the output correspondences. Namely, each classified correspondence point is mapped to 3D and compared to the ground truth 3D point. The ground truth 3D points are obtained in exactly the same way as predicted points, i.e., by matching a UV map rendered in the ground truth pose to model’s vertices.

The results per object are shown in Table 4.7. The table reports the quality of correspondences separately for real and synthetic data. For each model, mean absolute error, median absolute error, and standard deviation of absolute errors are reported in millimeters. Relatively large mean error is explained by outliers, some of which can be quite significant. Therefore, median is a better measure due to its robustness to outliers. The table shows that the median error is consistent across all the models. Additionally,

Table 4.7: Quantative correspondence quality. Correspondence quality for real and synthetic data estimated in terms of mean and median absolute errors, and standard deviation.

Model	Real Data			Synt Data		
	Mean	Median	Std	Mean	Median	Std
Ape	10.05	4.58	14.60	11.46	5.74	15.26
Bvise	10.36	4.70	19.29	15.92	6.99	25.71
Cam	6.57	4.58	10.11	13.31	7.23	20.23
Can	8.19	4.03	13.46	11.97	5.10	18.72
Cat	8.60	4.77	12.22	9.87	5.42	13.99
Driller	8.52	4.78	17.78	18.14	6.80	36.06
Duck	5.93	3.98	8.72	7.63	4.99	10.41
Eggbox	6.00	4.26	10.23	42.39	9.40	48.07
Glue	7.82	4.26	13.73	17.12	8.11	23.19
Holep.	8.25	4.87	13.30	11.28	6.81	16.04
Iron	7.18	4.51	12.31	11.06	6.89	17.34
Lamp	11.64	4.31	24.80	18.60	8.58	30.85
Phone	6.09	2.84	12.94	9.52	4.38	18.31

it demonstrates that the median error for the detector trained on real data is noticeably lower than for the detector trained on synthetic data. This explains the superior performance of training on real data.

Figure 4.6 provides a visual comparison of predicted and ground truth UV maps and heat maps, which demonstrate where imprecisions take place. One can see that most imprecisions are concentrated on the outer boundaries of the object and, for objects with more complex geometry, on the edges of their structural elements, i.e. in places where rapid correspondence value changes occur.

4.6.5.4 Refinement

A per-model evaluation is provided (see Table 4.8) to compare our proposed refiner with the former state of the art DeepIM [9]. It compares the following ADD scores: 1)

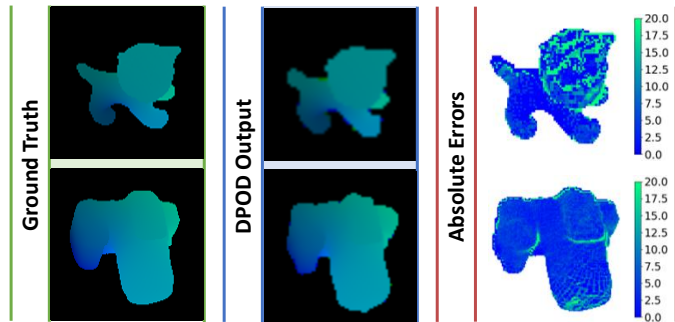
**Figure 4.6: Qualitative correspondence quality.** Comparison of ground truth (left), predicted (center) UV maps and heat maps (right) of absolute errors.

Table 4.8: Comparison of deep learning-based refinement methods: Our refinement approach shows the overall best ADD score with respect to the latest state-of-the-art method DeepIM [9].

Method/Object	Ape	Bench.	Cam	Can	Cat	Dril.	Duck	Eggb.	Gl.	Hol.	Iron	Lamp	Ph.	Avg.
PoseCNN [101] + DeepIM [9]	77.0	97.5	93.5	96.5	82.1	95.0	77.7	97.1	99.4	52.8	98.3	97.5	87.7	88.6
Ours + DeepIM [9]	78.70	98.43	97.75	97.57	85.16	91.55	80.24	99.68	99.48	75.66	99.74	98.20	91.38	91.81
Ours + Our ref.	87.73	98.45	96.07	99.71	94.71	98.8	86.29	99.91	96.82	86.87	100	96.84	94.69	95.15

ADD reported in the original DeepIM paper [9], which used PoseCNN [101] to predict initial poses, 2) ADD if DeepIM is applied to poses predicted by our detector, 3) ADD if poses predicted by the proposed detector are refined with the proposed refinement. It is important to mention that two iterations of DeepIM were made, as was suggested in the original paper. On the other hand, our refiner was run only for one iteration. The table clearly shows that better initial pose hypotheses allow for better results after refinement. It is also clear that our refinement clearly outperforms DeepIM on most of the objects, while performing only insignificantly worse on others.

4.7 Conclusion

In this chapter, we proposed the Dense Pose Object Detector (DPOD) method that regresses multi-class object masks and dense 2D-3D correspondences between image pixels and corresponding 3D models. Unlike the best performing methods that regress projections of the object’s bounding boxes [38, 40] or formulate pose estimation as a discrete pose classification problem [50], dense correspondences computed by our method allow for more robust and accurate 6D pose estimation. We demonstrated that for both, real and synthetic training data, our detector outperforms other related works, such as [40, 101], by a large margin and performs similarly to [93]. The proposed pose refinement approach also performs very well and allows for achieving a pose accuracy that surpasses all other related deep learning-based pose refinement approaches, while having a simpler and more lightweight backbone architecture.

5 9D Pose Estimation for Autolabeling

In this chapter, we present an automatic annotation pipeline to recover 9D cuboids and 3D shapes from pre-trained off-the-shelf 2D detectors and sparse LIDAR data. Our autolabeling method solves an ill-posed inverse problem by considering learned shape priors and optimizing geometric and physical parameters. To address this challenging problem, we apply a novel differentiable shape renderer to signed distance fields (SDF), leveraged together with normalized object coordinate spaces (NOCS). Initially trained on synthetic data to predict shape and dense normalized coordinates, our method uses these predictions for projective and geometric alignment over real samples. Moreover, we also propose a curriculum learning strategy, iteratively retraining on samples of increasing difficulty in subsequent self-improving annotation rounds. Our experiments on the KITTI3D dataset show that we can recover a substantial amount of accurate cuboids, and that these autolabels can be used to train 3D detectors with state-of-the-art results.

5.1 Introduction

Deep learning methods require large labeled datasets to achieve state-of-the-art performance. Concerning object detection for automated driving, 3D cuboids are preferred among other annotation types as they allow appropriately reasoning over all nine degrees of freedom (instance location, orientation, and metric extent). However, obtaining a sufficient amount of labels to train 3D object detectors is laborious and costly, as it mostly relies on involving a large number of human annotators. Existing approaches for scaling up annotation pipelines include the usage of better tooling, active learning, or a combination thereof [104, 105, 106, 107, 108]. Such approaches often rely on heuristics and require human effort to correct the outcomes of semi-automatic labeling, specifically for difficult edge cases.

Alternatively, we propose a novel approach relying on dense correspondence estimation and differentiable rendering of shape priors to recover metric scale, pose, and shape of vehicles in the wild. Our 3D autolabeling pipeline requires only 2D detections (bounding boxes or instance masks) and sparse point clouds (ubiquitous in 3D robotic contexts). Detections themselves are produced using off-the-shelf 2D detectors. We demonstrate that differentiable visual alignment, also referred to as “analysis-by-synthesis” [109] or “render-and-compare” [110], is a powerful approach towards autolabeling for the purpose of autonomous driving.

This chapter introduces three novel contributions. First, we formulate the notion of a Coordinate Shape Space (CSS), which combines Normalized Object Coordinates (NOCS) [111] with the DeepSDF framework [112]. This allows to reliably set object shapes into correspondence to facilitate deformable shape matching. Second, we present

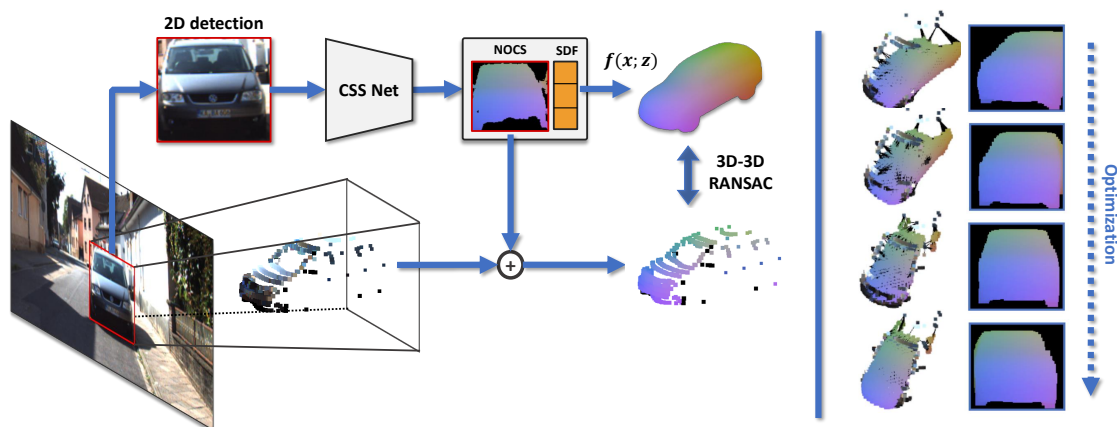


Figure 5.1: Our pipeline for 3D object autolabeling. Left: off-the-shelf 2D detections are fed into our Coordinate Shape Space (CSS) network to predict surface coordinates and a shape vector. We backproject the coordinates to LIDAR in the camera frustum and decode the shape vector into an object model. Then, we establish 3D-3D correspondences between the scene and model to estimate an initial affine transformation. Right: We iteratively refine the estimate via differentiable geometric and visual alignment.

a way to differentiate DeepSDF with respect to its surface, thereby introducing a novel differentiable SDF renderer for comparative scene analysis over a defined shape space. The third contribution is a curriculum learning-based autolabeling pipeline of driving scenes. Fig. 5.1 presents an example optimization on the KITTI3D dataset [113].

Our pipeline starts with a CSS network largely based on the DPOD detector from Chapter 4, which is trained to predict 2D NOCS maps (as opposed to UV maps in DPOD) as well as shape vectors from image patches. To bootstrap an initial version, we train the network on synthetic data, for which we can easily obtain ground truth NOCS and shape vector targets, and apply augmentations to minimize the sim2real domain gap. Our autolabeling loop includes the following steps: 1) leveraging 2D detections to localize instances; 2) running the CSS network on an extracted patch; 3) reprojecting NOCS into the scene using LIDAR, 4) decoding an object model from the shape space; 5) computing an approximate pose using 3D-3D correspondences; and 6) running projective and geometric alignment for refinement. After processing all images, we collect the recovered autolabels and retrain our CSS prediction network to gradually expand it into the new domain. Then, we repeat this process to achieve better CSS predictions and, consequently, better autolabels. To avoid drifting due to noisy autolabels, we employ a curriculum that is focused on easy samples first and increases the difficulty in each loop.

In summary, our main contributions are as follows: (i) a novel, fully-differentiable renderer for signed distance fields that can traverse smooth shape spaces; (ii) a mixed synthetic/real curriculum framework that learns to predict shape and object coordinates on image patches; and (iii) a multi-modal optimization pipeline combining differentiable alignment over vision and geometry. We evaluate our approach on the KITTI3D

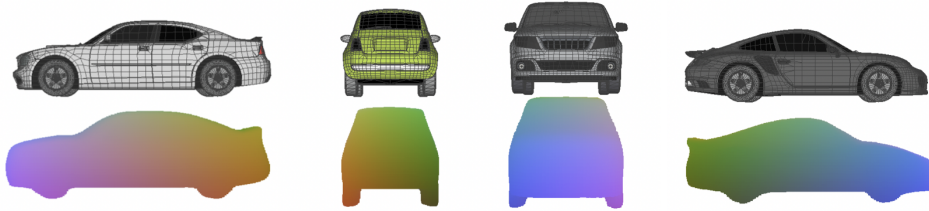


Figure 5.2: CSS representation. Top: Car models from the PD dataset [4]. Bottom: The same cars in the CSS representation: decoded shape vector \mathbf{z} colored with NOCS.

dataset [113] and show that our method can be used to accurately recover metric cuboids with structural, differentiable priors. Furthermore, we demonstrate that such cuboids can be leveraged to train efficient 3D object detectors.

5.2 Methodology

We first discuss our shape space construction and the coupling into the CSS representation. Afterwards, we introduce our differentiable rendering approach tailored towards implicit surface representations. Eventually, our autolabeling pipeline is described in more detail.

5.2.1 Coordinate Shape Space

We employ DeepSDF [112] to embed watertight car models into a joint and compact shape space representation within a single neural network. The idea is to transform input models into signed distance fields in which each value corresponds to a distance to the closest surface, with positive and negative values representing exterior/interior area. Eventually, DeepSDF forms a shape space of implicit surfaces with a decoder f that can be queried at spatially-continuous 3D locations $\mathbf{x} = \{x_1, \dots, x_N\}$ using the provided latent code \mathbf{z} to retrieve SDF values $\mathbf{s} = \{s_1, \dots, s_N\}$ as follows:

$$f(\mathbf{x}; \mathbf{z}) = \mathbf{s}. \quad (5.1)$$

To facilitate approximate deformable shape matching, we combine the shape space with NOCS [111] to form the Coordinate Shape Space (CSS). To this end, we resize our models to unit diameter and interpret the 3D coordinates of the 0-level set as dense surface descriptions.

To train f , we use a synthetic dataset provided by Parallel Domain [4], which comprises car CAD models, as well as rendered traffic scenes with ground truth labels (see Fig. 5.6). Other synthetic datasets (e.g., CARLA [114] & VKITTI [115]) could be used here as well. We trained on a subset of 11 models and with a latent code dimensionality of 3. We follow the original DeepSDF training, but project the latent vector onto the unit sphere after each iteration. In Fig. 5.2, we depict example models with their CSS representations.

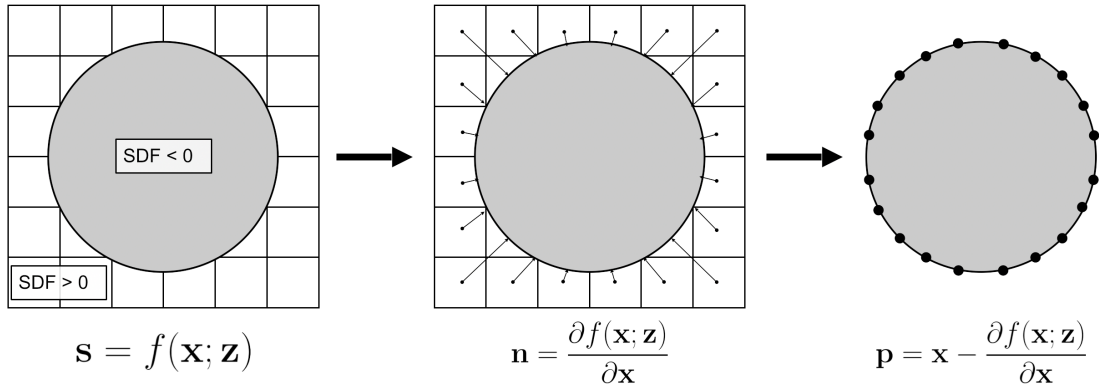


Figure 5.3: Surface projection. DeepSDF outputs the signed values \mathbf{s} for query locations \mathbf{x} . Normals \mathbf{n} can be computed analytically by a single backward pass. Given the signed values and normals, we project the query locations onto the object surface points \mathbf{p} . Only exterior points are visualized.

5.2.2 Differentiable SDF Rendering

An essential component of our autolabeling pipeline is the possibility to optimize objects with respect to the pose, scale, and shape. To this end, we propose, to the best of our knowledge, the first differentiable renderer for signed distance fields. Our renderer avoids mesh-related problems such as connectivity or intersection, but necessitates implementing a different approach for sampling the representation. Rendering implicit surfaces is done either with raytracing [116] or variants of Marching Cubes [117]. Here, we present an alternative that lends itself to backpropagation.

Projection of 0-Isosurface Given query points x_i and associated signed distance values s_i , we need a differentiable way to access the implicit surface encoded by \mathbf{z} . Simply selecting query points based on their distance values do not form a derivative with respect to the latent vector \mathbf{z} . Moreover, the regularly-sampled locations can be estimated only approximately on the surface. However, we utilize that deriving SDFs with respect to their location yields a normal at this point, practically computed in a backward pass:

$$n_i = \frac{\partial f(x_i; \mathbf{z})}{\partial x_i}. \quad (5.2)$$

As normals outline the direction to the closest surface and signed distance values provide the exact distance, we project the query location onto a 3D surface position p_i :

$$p_i = x_i - \frac{\partial f(x_i; \mathbf{z})}{\partial x_i} f(x_i; \mathbf{z}). \quad (5.3)$$

To obtain clean surface projections we disregard all points x_i outside a narrow band ($|s_i| > 0.03$) of the surface. A schematic explanation can be found in Fig. 5.3. With

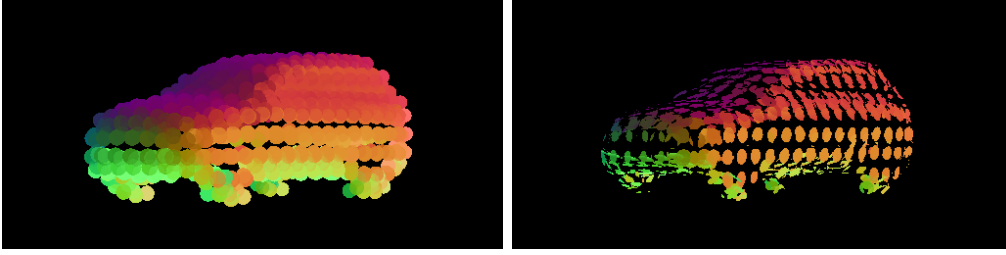


Figure 5.4: Oriented tangent discs (right) represent the surface geometry more accurately than billboard ones. We reduced spatial sampling and diameters for better emphasis.

this formulation, we can define derivatives at p_i with respect to the scale, pose, or latent code.

Surface Tangent Discs In the computer graphics domain, the concept of surface elements (surfels) [118] is a well-established alternative to connected triangular primitives. Our differentiable SDF representation yields oriented points and can be immediately used to render surface discs. To obtain a water-tight surface, we determine disk diameters large enough to close holes. In Fig. 5.4 we outline the difference between (oriented) surface tangent discs and billboard ones pointing straight at the camera.

We construct the surface discs with the following steps:

1. Given the normal of a projected point $n_i = \frac{\partial f(p_i; \mathbf{z})}{\partial p_i}$, we estimate the 3D coordinates of the resulting tangent plane visible in the screen. The distance d of the plane to each 2D pixel (u, v) can be computed by solving a system of linear equations for the plane and camera projection:

$$\begin{cases} u' = (u - o_u) \frac{d}{f_u} \\ v' = (v - o_v) \frac{d}{f_v} \\ Au' + Bv' + Cd - Au'_i - Bv'_i - Cd_i = 0 \end{cases} \quad (5.4)$$

The first two equations are the perspective projection equations and the third one is a plane equation. If we solve the above system by a simple substitution, we get the following:

$$\begin{aligned} d \left(\frac{A(u - o_u)}{f_u} + \frac{B(v - o_v)}{f_v} + C \right) - Au'_0 - Bv'_0 - Cd_0 &= 0 \longrightarrow \\ d &= \frac{Au'_i + Bv'_i + Cd_i}{\left(\frac{A(u' - o_u)}{f_u} + \frac{B(v' - o_v)}{f_v} + C \right)} \\ &= \frac{n_i \cdot p_i}{n_i \cdot \mathbf{K}^{-1}(u, v, 1)^T} \end{aligned} \quad (5.5)$$

where \mathbf{K}^{-1} is the inverse camera matrix, followed by backprojection to get the final 3D plane coordinate:

$$P = \mathbf{K}^{-1} \cdot (u \cdot d, v \cdot d, d)^T. \quad (5.6)$$

2. Estimate the distance between the plane vertex and surface point and clamp if it is larger than a disc diameter:

$$M = \max(\text{diam} - \|p_i - P\|_2, 0) \quad (5.7)$$

To ensure water-tightness we compute the diameter from the query location density: $\text{diam} = \min_{i \neq j} \|x_i - x_j\|_2 \sqrt{3}$. Executing the above steps for each pixel yields a depth map \mathbf{D}_i and a tangential distance mask \mathbf{M}_i at point p_i .

Rendering Function To generate a final rendering we need a function to compose layers of 2D-projected discs onto the image plane. Similarly to [61], we combine colors from different point primitives based on their depth values. The closer the primitive is to the camera, the stronger its contribution. We use softmax to ensure that all primitive contributions sum up to 1 at each pixel. More specifically, the rendering function is:

$$\mathcal{I} = \sum_i \text{NOCS}(p_i) * w_i, \quad (5.8)$$

where \mathcal{I} is the resulting image, NOCS returns coordinate coloring, and w_i are the weighting masks that define the contribution of each disc:

$$w_i = \frac{\exp(-\tilde{\mathbf{D}}_i \sigma) \mathbf{M}_i}{\sum_j \exp(-\tilde{\mathbf{D}}_j \sigma) \mathbf{M}_j}, \quad (5.9)$$

where $\tilde{\mathbf{D}}$ is the normalized depth, and σ is a transparency constant with $\sigma \rightarrow \infty$ being completely opaque as only the closest primitive is rendered. This formulation enables gradient flow from pixels to surface points and allows image-based optimization.

5.2.3 3D Autolabeling Pipeline

The general idea of our autolabeling approach is to exploit weak labels and strong differentiable priors to recover labels of higher complexity. While this idea is generic, we focus specifically on cuboid autolabeling of driving scenes.

We present a schematic overview in Fig. 5.5 where we run multiple loops of the annotation pipeline. In the first loop, the CSS label pool solely consists of synthetic labels and the trained CSS network is therefore not well-adapted to real imagery. The results are noisy NOCS predictions which are reliable only for *well-behaved* object instances in the scene. Therefore, we define a curriculum in which we first focus on *easy* annotations and increase the difficulty over more loops. We define the difficulty of a label by measuring the pixel sizes, the amount of intersection with other 2D labels, and whether the label touches the border of an image (often indicating object truncation). We also establish thresholds for these criteria to define a curriculum of increasing difficulty.

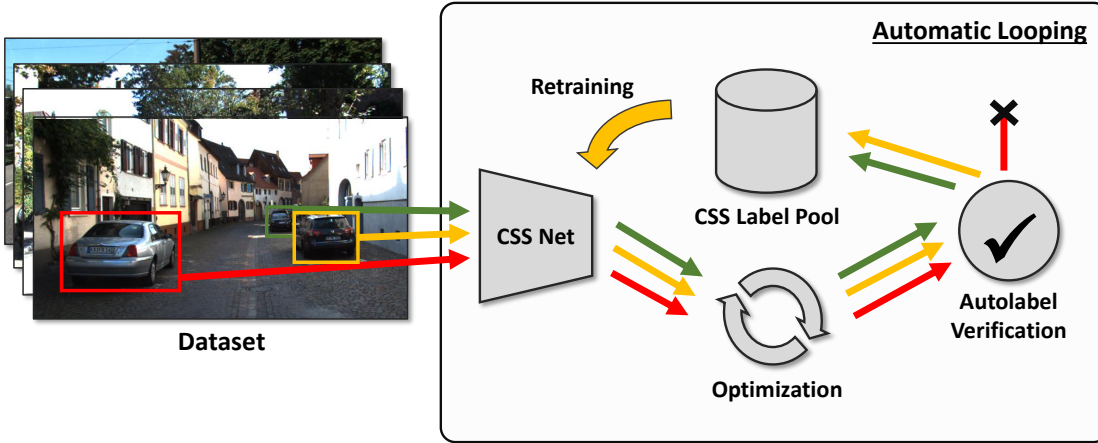


Figure 5.5: Automatic annotation pipeline. We fetch frames from the dataset and separately process each 2D detection using our CSS network and differentiable optimization procedure. Afterwards, we perform a verification to discard incorrect autolabels before saving them into our CSS label pool. Once all frames are processed, we retrain our CSS network and begin the next loop over the dataset.

CSS Network The network is largely based on DPOD from Chapter 4. It is derived from ResNet18 and adopts an encoder-decoder structure, processing 128×128 input patches to output a NOCS map (similar to UV maps) of the same size and a 3D shape vector. Before executing the first annotation loop, our CSS network must learn to infer 2D NOCS maps and shape vectors from patches. As mentioned, we bootstrap such a mapping from a synthetic dataset. In total, we extract around 8k patches and, having access to CAD models, also create the necessary regression targets. We demonstrate some frames and training data in Fig. 5.6b.

5.2.3.1 Initialization and Optimization

Here, we describe the process represented in Fig. 5.1 in more detail. For a given patch we infer the 2D NOCS map \mathcal{M} and shape vector \mathbf{z} . We decode \mathbf{z} into an SDF and retrieve the 3D surface points $\mathbf{p} = \{p_1, \dots, p_n\}$ of the object model (as described in Section 3.2) in its local frame, for which we compute the NOCS coordinates $\mathbf{p}^c = \{p_1^c, \dots, p_n^c\}$. We also project the 3D LIDAR points $\mathbf{l} = \{l_1, \dots, l_k\}$ contained inside the camera frustum onto the patch and collect the corresponding NOCS coordinates \mathbf{l}^c . To estimate an initial pose and scale, we establish 3D-3D correspondences between \mathbf{p} and \mathbf{l} . For each p_i , we find its nearest neighbor based on NOCS distances:

$$j^* = \arg \min_j \|p_i^c - l_j^c\|_2 \quad (5.10)$$

and keep it if $\|p_i^c - l_{j^*}^c\| < 0.2$. Finally, we run Procrustes [119] with RANSAC to estimate pose (R, t) and scale s .

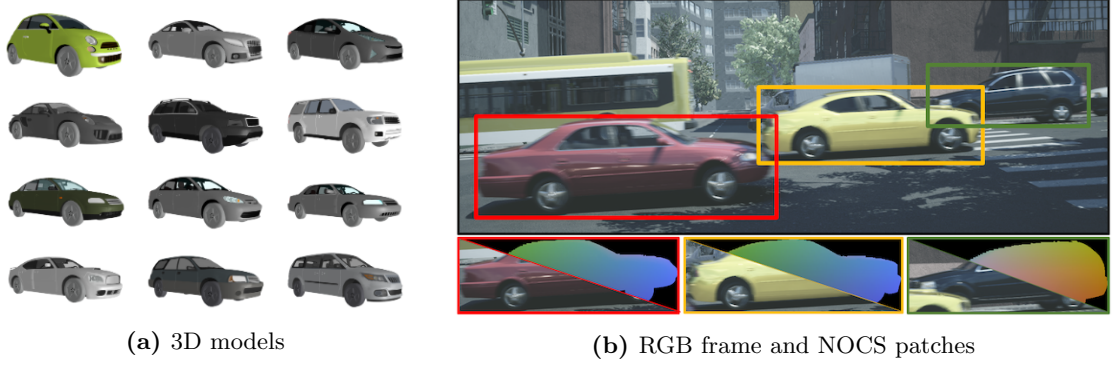


Figure 5.6: Synthetic PD dataset. (a) Cars from the PD dataset that were used to train our DeepSDF shape space. (b) Top: Random RGB frame. Bottom: Patches used for CSS training.

On this basis, we apply our differentiable optimization over complimentary 2D and 3D evidence. While the projective 2D information provides strong cues about the orientation and shape, 3D points allow reasoning over the scale and translation. At every iteration we decode the current shape vector estimate \hat{z} , extract surface points p_i , and transform them based on the current estimates of the pose and scale:

$$\hat{p}_i = (\hat{R} \cdot \hat{s}) \cdot p_i + \hat{t}. \quad (5.11)$$

Given these surface model points in the scene frame, we compute the individual losses as follows.

2D loss: We employ our differentiable SDF renderer to produce a rendering \mathcal{R} for which we seek maximum alignment with NOCS map \mathcal{M} . Since our predicted \mathcal{M} can be noisy (especially in the first loop), minimizing dissimilarity $\min\|\mathcal{M} - \mathcal{R}\|$ can yield suboptimal solutions. Instead, for each rendered spatial pixel r_i in \mathcal{R} we determine the closest NOCS space neighbor in \mathcal{M} , named m_j within a small radius θ , and set them in correspondence if their NOCS distance is below a predefined threshold. The loss is then defined as the mean distance over all such correspondences C_{2D} in the NOCS space:

$$loss_{2D} = \frac{1}{|C_{2D}|} \sum_{(i,j) \in C_{2D}} \|\mathcal{R}(r_i) - \mathcal{M}(m_i)\|_2. \quad (5.12)$$

3D loss: For each \hat{p}_i , we determine the nearest neighbor from l and keep it if it is closer than 0.25m. As the initializations are usually good, we avoid outliers in the optimization with such a tight threshold. The loss is then calculated as the mean distance over all correspondences C_{3D} :

$$loss_{3D} = \frac{1}{|C_{3D}|} \sum_{(i,j) \in C_{3D}} \|\hat{p}_i - l_j\|_2. \quad (5.13)$$

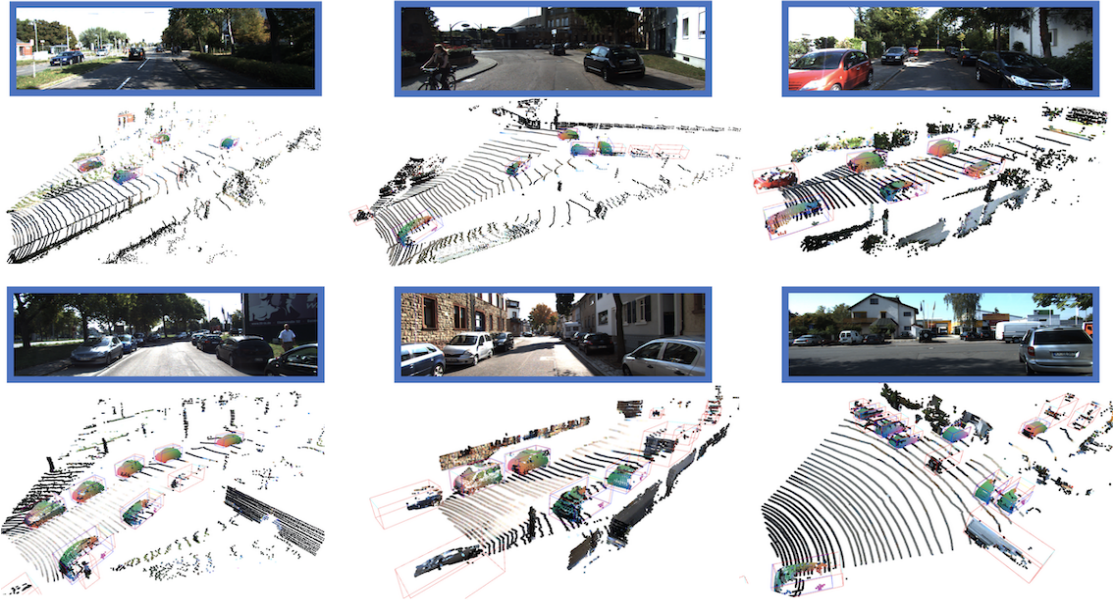


Figure 5.7: Qualitative results of our labeling pipeline. We mark the ground truth cuboids in red and our predictions in blue. We can achieve rather tight fits that lead to cuboids of slightly different sizes compared with the ground truth.

Altogether, the final criterion is the sum of both losses:

$$loss = loss_{2D} + loss_{3D}. \quad (5.14)$$

As the loss terms have similar magnitudes we did not consider a need for any additional balancing.

5.2.3.2 Verification and CSS Retraining

Our optimization framework will inevitably output incorrect results at times, so we need to ensure that the influence of badly-inferred autolabels is minimized. To this end, we enforce geometric and projective verification aiming to remove incorrect autolabels with the largest impact. To achieve this, we measure the amount of LIDAR points that are in a narrow band (0.2m) around the surface of an autolabel and reject it if less than 60% are outside this band. Furthermore, we define a projective constraint where autolabels should be rejected if the rendered mask’s IoU with the provided 2D label is below 70%.

All autolabels that remain after the verification stage are gathered and added to the CSS label pool. After the first loop, we obtain a mixture of synthetic and real samples that are then used to retrain and improve the robustness of our CSS network. Over multiple self-improving loops, we keep growing and retraining, which results in better CSS predictions, better initializations, and more accurate autolabels.

5.3 Experimental Evaluation

We evaluate our approach on the well-established KITTI3D dataset [113], including 7481 frames with accompanied cuboid labels for the “Car” category, which we focus on. We consider the most-widely used 3D metrics for driving datasets: BEV IoU and 3D IoU from KITTI3D as well as the distance-thresholded metric (NS) from NuScenes [120], which decouples location from scale and orientation. All three metrics are utilized to evaluate Average Precision for matches at certain cutoffs, and we threshold BEV and 3D IoUs at 0.5 whereas NS is computed at distance cutoffs of 0.5m and 1m.

The KITTI3D metrics are often evaluated at a strict threshold of 0.7. After thorough inspection, we observed that it is difficult to infer the correct cuboid size from tight surface fits. The KITTI3D cuboids have a varying amount of spatial padding and 3D detection methods learn these offsets. Therefore, we opted to relax the threshold aiming to facilitate a fairer comparison with respect to the estimated tight cuboids. Figure 5.7 represents several examples of appropriate autolabel estimates among which some do not pass the 0.7 3D IoU criterion.

Implementation Details We use PyTorch [121] to implement the whole pipeline. For each 2D-labeled instance we run 50 iterations and use the ADAM optimizer for the pose variables with a learning rate of 0.03 whereas SGD is applied to the scale and shape with smaller learning rates (0.01 and 0.0005) and no momentum to avoid observed overshooting behavior. It takes approximately 6 seconds to autolabel a single instance on a Titan V GPU, and one autolabeling loop takes 1-2 hours to complete for all frames when parallelizing on 2 GPUs.

Data Augmentation Our synthetic bootstrapping requires many kinds of augmentation to allow for initial domain transfer. Given a set of ground truth poses with associated CAD models that we extract from our synthetic PD dataset (see Figure 5.6b), we use our DeepSDF network and our differentiable renderer to project the models onto the screen. Instead of rendering the colors, we render the models’ normalized coordinates (NOCS) represented as RGB channels (Figure 5.8c). Additionally, we render object normals (Figure 5.8b), which are subsequently used for 0-isosurface projection. Our augmentation module takes RGB crops (Figure 5.8a) and normal maps and applies 2D and 3D augmentations. 2D augmentations are based on the `torchvision.transforms` module operations and include random 10° rotations, horizontal flips, cropping, changes

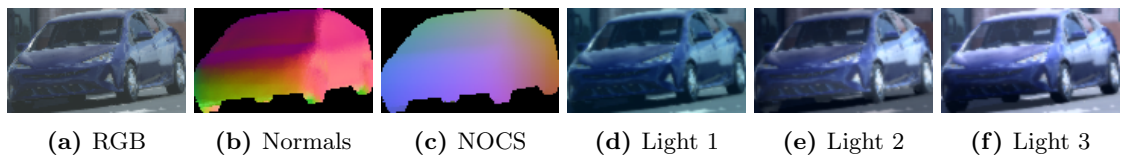


Figure 5.8: Data input modalities: (a) input RGB image, (b) rendered normal map, (c) rendered NOCS. Light module outputs: (d, e, f).

Table 5.1: Cuboid autolabel quality when inputting into the CSS network (a) 2D ground truth boxes, (b) RCNN detections, and (c) Mask-RCNN detections. We run two self-improving loops to slowly incorporate more labels into the pool.

Loop	Diff.	KITTI GT				RCNN				MASK-RCNN			
		BEV@0.5	3D@0.5	NS@0.5	NS@1.0	BEV@0.5	3D@0.5	NS@0.5	NS@1.0	BEV@0.5	3D@0.5	NS@0.5	NS@1.0
1	E	78.09	63.53	85.59	95.58	78.45	63.71	85.85	95.62	78.46	63.69	86.27	95.76
2	E	77.84	62.25	82.40	90.84	80.57	60.11	86.05	94.62	80.70	63.96	86.52	94.31
	M	59.75	42.23	60.27	77.91	61.17	42.37	64.11	85.85	63.36	44.79	64.44	85.24

in brightness, contrast, and saturation. Moreover, normal maps provide us with local surface information that we use in conjunction with simple Phong shading. Thus, we can generate lighting based on different illumination types (namely ambient, diffusive, and specular) during training. Examples are shown in Figures 5.8d, 5.8e, 5.8f. Note how the bottom and top sides of the car change illumination between frames (d) and (f).

5.3.1 Correctness of Autolabels

The most important quantitative criterion is the actual correctness of the estimated cuboids. Although our method is fully automatic, we have access to KITTI3D 2D ground truth boxes and therefore evaluate two scenarios. Firstly, we obtain 2D boxes from KITTI3D for autolabeling and then, utilize their predefined criteria to determine whether an annotation is considered easy or moderate. Secondly, we employ the detectron2 implementation [122] of Mask-RCNN [123] using a ResNeXt101 backbone trained on COCO to evaluate the applicability of off-the-shelf object detectors for full automation. In the detection scenario, we run separate experiments for boxes and masks and apply following difficulty criteria similar to KITTI3D: easy if label height > 40 px and not

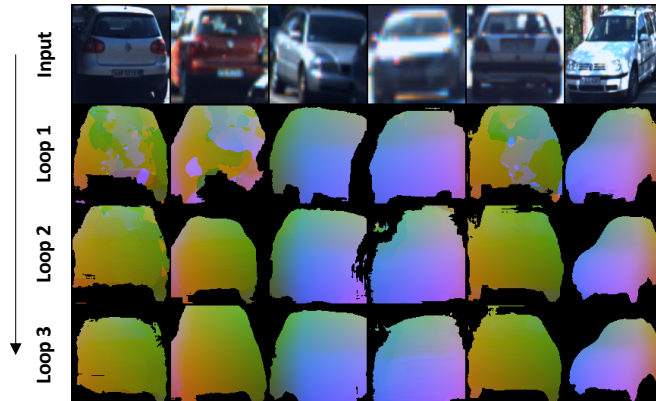


Figure 5.9: NOCS prediction quality of our network over consecutive loops for the same patch. Initially, the predictions are rather noisy because of the synthetic domain gap. Within each subsequent autolabeling loop the predictions become more accurate overall.

Table 5.2: The performance comparison of the 3D object detectors trained on the true KITTI labels vs. our autolabels. Concerning the BEV metric, the detectors trained on autolabels alone achieve the results equal to the current state of the art. In the case of the 3D AP metric, the competitive results are achieved in both considered variants at the IoU 0.5 threshold.

Method	2D AP @ 0.5/0.7		3D AP @ 0.5/0.7		BEV AP @ 0.5/0.7	
	Easy	Moderate	Easy	Moderate	Easy	Moderate
PointPillars [124] (GT Labels)	- / -	- / -	94.8 / 81.1	92.4 / 68.2	95.1 / 92.1	95.1 / 84.7
PointPillars [124] (Autolabels)	- / -	- / -	90.7 / 22.4	71.1 / 13.3	94.9 / 81.0	88.5 / 59.8
MonoDIS [125] (GT Labels)	96.1 / 95.5	92.6 / 86.5	45.7 / 11.0	32.9 / 7.1	52.4 / 17.7	37.2 / 11.9
MonoDIS [125] (Autolabels)	96.7 / 85.8	86.2 / 67.6	32.9 / 1.23	22.1 / 0.54	51.1 / 15.7	34.5 / 10.52

touching other labels or image borders; moderate if height > 25 px and not having an IoU > 0.30 with any other label.

We present the obtained results in Table 5.1. As expected, the first loop with a purely synthetically-trained inference on easy real samples does not yield a considerable difference between the three scenarios. All of them are impacted by noisy CSS predictions and start from the same RANSAC initialization, although the detection-derived labels are tighter and slightly less influenced by CSS background noise. Overall, each scenario achieves a BEV AP close to 80% and a 3D AP of approximately 60% over all easy samples. We execute a second loop over the dataset using the retrained CSS network and observe that the results for the easy samples stabilized. Additionally, we note that we can recover approximately 60% BEV AP over all considered moderate samples. Noteworthy, the estimated NS scores are quite high, indicating that most autolabels (more than 90%) are within one meter of the real location.

However, we observe a drop of around 20 points across all metrics for the harder samples. As our method is reliant on 3D-3D RANSAC, it requires a minimum set of inliers to achieve proper solutions. We often observe that this made correspondence finding difficult for occasional samples due to occlusion and distance, thus impacting recall.

Fig. 5.9 shows the increasing quality of the estimated NOCS predictions over multiple loops. Overall, we observe a rather fast diffusion into the target domain and that executing two loops is sufficient to stabilize the results.

5.3.2 Ablation

We aim to investigate how much the initial estimates from 3D-3D RANSAC benefit from our optimization. To this end, we consider the easy ground truth boxes from KITTI3D and utilize the synthetic CSS network to analyze the first annotation loop with the worst initialization. As represented in Table 5.3, the RANSAC baseline provides rather good localization which is best captured by the NS metrics (81.36% and 95.45%). Nonetheless, the pose-optimized autolabels yield a significant jump in 3D IoU (41.85% vs. 63.42%), suggesting that we recover substantially better rotations, given that the NS scores are

similar. When ablating over the other variables we observe rather mixed results in which certain metrics increase or decrease.

When ablating over the losses, we note a drastic drop in the 3D metrics when optimizing only in 2D. Intuitively, our differentiable renderer aligns the data rather appropriately in the image space; however, both scale and translation are freely drifting. Therefore, optimizing the 3D loss results in strong spatial alignment. Nonetheless, optimizing the sum of both losses trades BEV AP (80.61 to 78.09) for 3D AP (60.92 to 63.53).

5.3.3 Autolabeling for 3D Object Detection

Since autolabels are usually not the final goal but rather a means to an end, we investigate the applicability of our labels to the task of 3D object detection. We evaluate the quality of our labels concerning both a traditional LIDAR-based detection setting and a purely monocular setting, based on several recent works that achieved high quality results on the KITTI dataset [126, 125, 124].

We implemented a version of the current state-of-the-art monocular detector MonoDIS [125] and ensured that we can reproduce the reported results. Additionally, we utilize the official implementation of PointPillars [124], a state-of-the-art LIDAR-only detector. To train MonoDIS, we follow the training schedule proposed in [125]. Concerning PointPillars training, we accelerate the training by means of 8 V100 GPUs and a batch size of 16. Accordingly, we scale the learning rate by a factor of 8. We evaluate the obtained results on the MV3D train/val split [127]. While training on autolabels, we do not change any hyperparameters defined in the baseline protocols.

We present the comparison results in Table 5.2 and depict several qualitative detections obtained from autolabel-trained detectors in Fig. 5.10.

Remarkably, concerning the BEV metric, both detectors trained on autolabels alone achieve competitive performance compared with detectors trained on true KITTI labels at both the 0.5 and 0.7 IoU thresholds. This indicates that our autolabeling pipeline is capable of highly accurate localization of cuboids.



Figure 5.10: Detections from the autolabel-trained detectors. We draw local 3D frames to identify correct orientation.

Table 5.3: Ablation study over each optimization variable and each separate loss.

Config	BEV@0.5	3D@0.5	NS@0.5	NS@1.0
RANSAC	77.00	41.85	81.36	95.45
(R, t)	77.19	63.42	86.20	95.53
$(R, t), s$	77.23	62.92	86.01	95.32
$(R, t), s, \mathbf{z}$	78.09	63.53	85.59	95.58
2D loss	18.08	11.09	18.35	46.19
3D loss	80.61	60.92	85.63	95.49

Considering the 3D AP metric, the obtained results are in line with the conclusions represented in Table 5.1. At the more tolerant IoU 0.5 threshold, our autolabel trained detectors performed within 70 – 90% of the true labels. Occasionally missing detections and poor shape estimates do not deteriorate the overall performance.

At the IoU 0.7 threshold, the detector performance worsens. We observe that this is not caused by poor predictions, but by the fact that KITTI labels are often inflated with respect to the estimated cuboids. Therefore, at the strictest thresholded 3D IoU, we observe a corresponding drop in precision for detectors trained on our “tight” autolabels.

5.4 Conclusion

In this chapter, we presented a novel view on parametric 3D instance recovery in the wild based on a self-improving autolabeling pipeline, purely bootstrapped from synthetic data and off-the-shelf detectors. Fundamental to our approach is the combination of dense surface coordinates with a shape space, and our contribution towards differentiable rendering of SDFs. We show that our approach can recover a substantial amount of cuboid labels with high precision, and that these labels can be used to train 3D object detectors with results close to the state of the art.

6 RGB-D 6D Pose Estimation Dataset

Among the most important prerequisites for creating and evaluating 6D pose detectors are the datasets with labeled 6D poses. With the advent of deep learning, demand for such datasets is growing continuously. However, existing datasets are scarce and typically have restricted setups, such as a single object per sequence, or they focus on specific object types, such as textureless industrial parts. Besides, two significant components are often ignored: training using only available 3D models instead of real data and scalability, i.e., training one method to detect all objects rather than training one detector per object. Other challenges, such as occlusions, changing light conditions and changes in object appearance, as well as precisely defined benchmarks are either not present or are scattered among different datasets. In this chapter, we present a dataset for 6D pose estimation mainly targeting the mentioned challenges. It features 33 objects (17 toy, 8 household and 8 industry-relevant objects) over 13 scenes of various difficulty. We also present a set of explicit benchmarks aiming to test various desired detector properties. For each of the benchmarks we set a baseline using the state-of-the-art DPOD 6D object detector from Chapter 4.

6.1 Introduction

It is remarkable that most deep learning-based 6D pose detectors use a single neural network per object, in contrast to 2D object detectors, e.g., YOLO [39], SSD [98] or R-CNNs [128, 129, 130, 131], which use one network for all object classes. One of the reasons for this issue is the unavailability of a proper dataset with a variety of sequences and objects and well-defined benchmarks. This is natural since producing large number of 3D models and scenes with annotated poses is an expensive and time-consuming task. Nevertheless, training one network per object defeats the scalability aspect that is a natural characteristic of deep neural networks. One of the central aspects of our proposed dataset is the ability to test methods' scalability by introducing corresponding benchmarks and to encourage training of detectors on renderings of 3D models instead of using real data.

Another problem that stems from the unavailability of large-scale 6D pose estimation datasets is overfitting. It is not a secret that when it comes to deep learning-based 6D object detectors, training them on real data yields the best results. However, since existing datasets are rather small, 6D pose estimators trained on real data from existing datasets often overfit. Unfortunately, since training images are inevitably very similar to the test ones, the results do not reveal this problem still providing very good estimates. This makes it very difficult to properly evaluate the actual power of the method and undermines fair comparison of various approaches. What is even more disappointing, is



Figure 6.1: HomebrewedDB scene examples. Our dataset features 13 RGB-D annotated scenes of various difficulty. The reconstructed 3D models of the objects are rendered on top of RGB images with obtained ground truth poses.

that when overfitted deep learning-based 6D object detectors are applied to real data containing the training objects placed in a different environment, they often fail to generalize providing completely unreliable estimates and making their possible industrial usage rather improbable. A sound alternative to using real annotated sequences for training are synthetic data. Textured synthetic CAD models can be easily rendered to compile a dataset of needed complexity and variability. However, this fact was only used in few studies, most notably SSD6D [50] and DPOD (Chapter 4). As we will show in this chapter, despite of showing worse results on the test set when compared to methods trained on real data due to the domain gap, training from synthetic data results in network with better generalization properties.

Our dataset contains 13 full-circle scenes filmed with both PrimeSense Carmine 1.09 (structured light) and Microsoft Kinect 2 (time-of-light) RGB-D cameras resulting in a total of 34,830 fully annotated frames with poses for all objects in all frames. The complexity of the scenes varies from simple (several separated objects per scene) to heavily-cluttered and occluded (objects close together or on top of each other and also mixed with other objects not present as 3D models). Another aspect that is not addressed in the other datasets is strong variation of the illumination, including not only changes in light intensity, but also in light color. Driven by the fact that in industry objects can undergo severe appearance changes, we created a benchmark where the object appearance is altered compared to the one in available 3D models.

To summarize, the contributions of this chapter are as follows:

1. **33 highly accurately reconstructed 3D models** of toys, household objects and low-textured industrial objects of sizes varying from 10.1 to 47.7 cm in diameter.
2. **13 sequences**, each containing 1340 frames filmed using 2 different RGB-D sensors. Scenes span a range of complexity from simple (3 objects on a plain background) to complex (highly occluded with 8 objects and extensive clutter) (Fig. 6.3). Also, two sequences feature drastic light changes or contain objects with altered textures (Fig. 6.4).

3. **Precise 6D pose annotations** for dataset objects in the scenes, which were obtained using an automated pipeline (see Section 6.3.5).
4. **A set of benchmarks** to facilitate comprehensive evaluation of object detection and 6D pose estimation methods.

6.2 Related Datasets

In this section, we discuss the most popular datasets for 6D pose estimation. Given the fact that determining ground truth 6D poses from RGB images is an ambiguous task requiring manual interventions, it is not surprising that the majority of 6D pose datasets are made with the use of RGB-D cameras. Because these cameras provide depth images aligned with color images, the task of 6D pose estimation becomes simpler and more automated.

LineMOD Dataset. One of the most widely used 6D pose datasets is LineMOD by Hinterstoisser et al. [58] containing objects embedded in cluttered scenes. It was acquired with PrimeSense Carmine RGB-D sensor and comprises 15 objects in total. For each sequence, poses of only one object are annotated. The target objects with the existing ground truth poses are either not occluded, or are subject to very slight occlusions. The original template matching method [58], which was published alongside the dataset, performed poorly on RGB and depth images separately, while the results obtained on RGB-D images were extremely good and still remain hard to achieve by many modern deep-learning-based methods. However, this method scales poorly and cannot handle occlusions well. LineMOD dataset also has an extension called OCCLUSION dataset created by Brachmann et al. [100] to address the lack of occluded test data. In this extension of the original LineMOD additional manual annotations of 6D poses for all the objects in each frame were included. Due to the necessity of intensive manual labor, it was done for only a limited number of frames. Availability of RGB and depth images as well as 3D CAD models with original object colors resulted in probably the widest use of the LineMOD dataset of all those targeting 6D pose estimation. Deep learning methods use RGB images from this dataset for object detection and 6D pose estimation.

T-LESS Dataset. T-LESS [132] is a more recent dataset that is gaining popularity. It contains 30 textureless industrial objects and 20 RGB-D scenes captured with three synchronized cameras (PrimeSense Carmine 1.09 and Kinect 2 RGB-D cameras and Canon RGB camera). The objects in this dataset have strong inter-object similarity. The acquired scenes vary from simple (less clutter and several objects per scene) to complex (heavily cluttered, with piles of objects, mimicking a typical robotic bin-picking scenario). Both hand-designed CAD models and reconstructed 3D models are included in T-LESS. Training images contain isolated objects on black backgrounds, while test images capture entire scenes with labeled 6D poses for each object in each frame. The structure of this dataset is exceptional, but due to symmetry, low texture and the industrial nature of the objects it is very challenging, which might be the reason why it

has not gained in popularity at the pace it deserves. Truly inspired by T-LESS [132], we prepared our dataset similar to this one in terms of structure. However, our dataset covers a wider range of object types, spanning toys, household objects as well as industrial objects. Also, we aimed to build a dataset with occlusion challenges, and that goes well beyond OCCLUSION, thereby providing many more frames and scenes for this task.

Other Relevant Datasets. The following datasets have been sporadically used in some publications, but their systematic use has not been achieved. Tejani et al. [133] contains 2 textureless and 4 textured objects with 700 frames of test images. A characteristic of this dataset is that it includes multiple instances of the same object. Clutter and occlusions are moderate, which makes it not particularly challenging. Doumanoglou et al. [134] presented a bin-picking dataset with 183 test images and only two objects from the Tejani dataset, but multiple instances of them. The Challenge and Willow datasets [135] contain a larger number of objects (176), but a relatively small number of test images (353). The TUW dataset [136] is similar with 17 objects in 224 test images. Datasets that are suited for robotic tasks are the Rutgers [137], the Amazon Picking Challenge [138], and the BIGBird [139] datasets. Datasets addressing the light change challenge are TUD-Light and Toyota Light, both used and referenced in the benchmark paper by Hodan et al. [140]. Another recent dataset that came out with the PoseCNN detector [101] is YCB-Video. Unlike T-LESS, which contains images of the scenes from all viewpoints, this dataset resembles LineMOD, containing short videos depicting several household objects in the scene. The large number of video sequences (92) as well as the presence of occlusions in the test scenes make this dataset quite attractive.

Our dataset could be considered to be between OCCLUSION and T-LESS. It contains high-quality annotations, a multitude of objects and a large number of scenes and test images. In contrast to the recent work of Hodan et al. [140], where the authors tried to unify 8 different datasets and have concentrated on evaluation of RGB-D methods, we are more interested in RGB methods, even though full RGB-D images are available in our dataset. With scalability at its core, we design benchmarks in which one network is trained either for all the available objects or only for the objects present in a particular scene. Additionally, we introduce scenes with severe environment changes, including changes in light color and intensity as well as changes in object appearance. We believe that this dataset will push forward research on scalable 6D object detection and domain adaptation.

6.3 HomebrewedDB Dataset Creation

The following sections cover the dataset creation, including calibration of RGB-D sensors, reconstruction of 3D models, depth correction, acquisition of image sequences and creation of ground truth annotations.



Figure 6.2: Rendered reconstructed 3D models of HomebrewedDB.

6.3.1 Calibration of RGB-D Sensors

For the footage of validation and test sequences we used two RGB-D sensors: the structured-light PrimeSense Carmine 1.09 and the time-of-flight Microsoft Kinect 2. Intrinsic and distortion parameters of both sensors were estimated during the calibration procedure. We used ArUco board [141], which yielded better calibration results in comparison to the classical checkerboard pattern, and the corresponding intrinsic calibration module from OpenCV [142]. As a result of calibration, the root-mean squared reprojection error calculated at the corners of the ArUco markers is ≤ 0.5 and ≤ 0.3 px for Carmine and Kinect 2 respectively. Intrinsic and distortion parameters for both sensors are provided with the dataset. Because depth and color images were obtained from two different cameras, depth-to-color registration was performed using OpenNI 2.2 Driver for Carmine and Windows SDK 2.0 for Kinect 2. Given that the scenes were recorded independently with each of the sensors, there was no need in extrinsic calibration of the cameras.

6.3.2 Sequence Acquisition

In total, we acquired 1 handheld and 2 turntable sequences for each of the scenes with each RGB-D sensor. Turntable sequences capture a full 360° rotation of a markerboard with objects on it using a camera mounted on a tripod. Each turntable sequence has 170 RGB and depth images filmed with elevation angles of 30° and 45° . Together with the ground truth 6D pose labels they form a validation dataset. In contrast, the test sequences were recorded in handheld mode. There were two major reasons for the handheld recording instead of using a controlled setup similar to those in T-LESS [132] or BigBIRD [139]. The first clear advantage is the close resemblance to regular camera use, while the second one is the ability to introduce more variation to camera poses in terms of considerable scale changes as well as in-plane rotation. For the test sequences, a total of 1000 RGB and corresponding depth images of each scene were captured with each sensor.

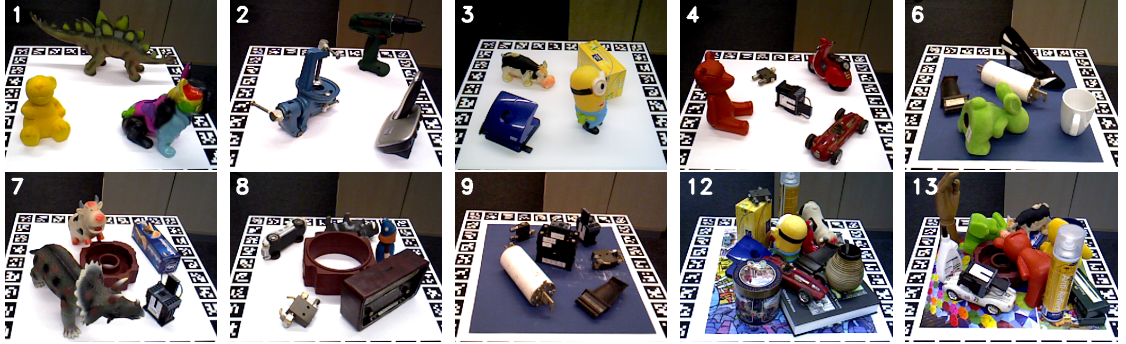


Figure 6.3: Sample RGB images from the sequences presented in the HomebrewedDB dataset. Complexity of the scenes varies in terms of number and size of objects, levels of occlusion and clutter.

While shooting the sequence, a full pass around the markerboard was made. Within the test sequences, the distance from the camera to an object was varied from 0.42 to 1.43 m, while the elevations were within 11° - 87° . Both color and depth images in the sequences that were recorded with the Carmine sensor have a default resolution of 640×480 px, while Kinect 2 RGB images are of size 1920×1080 px and depth images of 512×424 px. The depth images were resized to the dimension of RGB images during registration.

For each of the scenes (Figs. 6.3 and 6.4) target objects were placed on the markerboard with ArUco markers facilitating camera pose estimation. For the simpler scenarios, we placed the objects on a monochrome (white or dark-blue) Lambertian surface, and for the more complicated ones the objects resided on a multicolor reflective surface, being in some cases on top of each other. Also, more complicated sequences featured severe occlusions as well as objects not present in the dataset to make the scene more cluttered.

One new feature introduced in HomebrewedDB is a domain adaptation sequence aiming to test the robustness of detection and pose estimation methods with respect to significant changes in lighting conditions. To create it, we used a spotlight to repeatedly project series of light patterns with different colors and intensities onto the scene. We also created a sequence to evaluate robustness to considerable texture changes. For that we altered the textures of the objects by selectively painting some part of them with chalks of various colors.

6.3.3 3D Model Reconstruction

To obtain the 3D models presented in HomebrewedDB (Fig. 6.2), we first scanned each object from multiple viewpoints using Artec Eva [143], a structured light 3D scanner. We opted for Artec Eva since it provides precise depth measurements and high resolution texture maps, which are crucial for reconstructing high-quality 3D models. The following pipeline for converting the scans to completely reconstructed 3D models proceeded using Artec Studio software. First, raw meshes were reconstructed for each of the scanned viewpoints. Secondly, we manually removed unnecessary parts of the meshes and then



Figure 6.4: Sample RGB images from sequences belonging to the domain adaptation benchmark. From left to right: original sequence, illumination benchmark sequence, texture change benchmark sequence.

aligned them. Then we removed outliers and minor artifacts from the models. After that we proceeded with global optimization of the mesh structure, including inpainting minor holes in the model and as inducing smoothness of the mesh. Next, we back-projected high-resolution textures onto the resulting 3D model. Finally, we used MeshLab [144] to center and axis-align the models and computed surface normals as weighted sum of normals of the incident facets [145].

6.3.4 Depth Correction

Similar to works [146, 132], we observed that depth measurements by both Carmine and Kinect 2 have systematic errors: the measured depth values were always slightly different from those calculated from the markers in images captured with calibrated RGB cameras. Although it has been reported [146] that a single correction multiplier is sufficient to address the depth measurement error, we confirmed in our setup what Hodan et al. [132] had found – that first degree polynomial works better as a correction factor for the depth measurements in our setup. Using regularized least squares to account for noise in the measurements we derived the following linear depth-correction models: $d_c = 1.0391 \cdot d - 15.8$ for Carmine and $d_c = 1.0186 \cdot d - 13.1$ for Kinect 2 (measured in millimeters). After applying the corrections, we found that the mean absolute difference from expected depth had been reduced from 14.7 mm to 2.03 mm and from 5.81 mm to 2.66 mm for Carmine and Kinect 2 respectively. We applied the corrections models to the entire dataset, so that no further user action is required.

6.3.5 Creation of Ground Truth Annotations

The estimation of 6D ground truth objects poses for each frame in the sequence proceeded as follows. First, the markerboard pose was estimated, providing us with the camera trajectory around the scene. Then we obtained a dense 3D reconstruction by signed distance field fusion of depth maps of a scene with a method of Curless and Levoy [147], using all the images in a sequence.

The next step was to estimate a rigid body transformation of a 3D model from its own coordinate system to the coordinate system of the scene (i.e. markerboard). Given

Table 6.1: Differences between the depth of object rendered models at the ground truth poses and the captured depth (in mm). μ_δ and σ_δ is the mean and the standard deviation of the differences, $\mu_{|\delta|}$ and $med_{|\delta|}$ is the mean and the median of the absolute differences.

Sensor	μ_δ	σ_δ	$\mu_{ \delta }$	$med_{ \delta }$
Carminc	0.11	6.25	1.71	2.56
Kinect 2	0.22	7.38	0.87	9.12

that the locations of the objects with respect to the markerboard did not change when imaged with different sensors, it was sufficient to get the object pose in the markerboard coordinate system and then transfer it to a new sensor or camera coordinate system, thereby avoiding performing reconstruction for each new sensor. For the purpose of estimating 3D model poses in the reconstructed scene we used a method by Drost et al. [148], which is based on the point-pair feature representation of a target model used for local matching via an efficient voting scheme on a reduced 2D search space. Having a camera pose in each image estimated from the markerboard as well as having object poses in the markerboard coordinate system, 6D object poses for each of the frames can easily be computed. However, rendering the 3D models on top of RGB images revealed that even though the method by Drost et al. [148] gives a good initial estimate of a pose, in many cases there remain visible discrepancies, which must be mitigated in the process of further refinement.

To improve the poses, we opted for 2D edge-based ICP [54] refinement. For each object in the sequence we automatically selected RGB images where the object was not occluded. This was done by rendering all the objects in the scene with the estimated initial poses and calculating the fraction of visible pixels for the target object. Multiview consistency was enforced such that all the camera poses stayed fixed, and optimization was only done for the object pose in the scene coordinate system. Edge-based refinement was performed on RGB images because depth measurements were relatively noisy and we observed minor misregistrations between depth and RGB images, particularly for those captured with Kinect 2.

6.3.6 Accuracy of Ground Truth Poses

To evaluate the accuracy of the computed ground truth poses, we followed the same procedure as introduced by Hodan et al. [132]. We rendered the 3D objects using computed ground truth poses and for each pixel pair with valid depths in both rendered and captured images we computed the difference $\delta = d_c - d_r$, where d_c and d_r are captured and rendered depth values, respectively. The statistics obtained over the whole test set are presented in Tab. 6.1. As in T-LESS [132], differences exceeding 5 cm were omitted from the statistics as outliers. In HomebrewedDB, such measurements amounted to 3.7%, mostly caused by the clutter objects occluding the target objects, sensor mea-

surement noise, or minor discrepancies between reconstructed 3D models and real-world objects.

From the presented results it can be seen that rendered depth maps align well with the depth maps obtained with Carmine, resulting in mean depth difference close to zero and absolute mean of differences ≤ 2 mm. For Kinect 2 depth differences mean and absolute mean values also stay very close to zero: the absolute median value is notably higher than the absolute mean, signifying that the distribution of the depth differences is left-skewed. As noted in T-LESS [132], this might be caused by slight misregistration of RGB and depth images captured by Kinect 2, as well as higher magnitude of noise in the measurements of this depth sensor based on the time-of-flight principle.

6.4 Benchmarks and Experiments

In this section, we present a set of benchmarks assessing the performance of a 6D object detector with respect to a variety of different conditions. In particular, such aspects as scalability and resistance to occlusions, different illumination conditions and changes in object texture are tested.

6.4.1 Evaluation Metrics

We use standard metrics for evaluating performance in 2D object detection: precision, recall and mean average precision (mAP). Conventionally, we consider an object to be correctly detected if the intersection over union (IoU) between the ground-truth and predicted bounding boxes is ≥ 0.5 . To evaluate the correctness of the estimated 6D poses, as others have done [20, 50], we use the ADD score, which is defined as the average Euclidean distance between the model vertices transformed with ground truth and predicted poses:

$$m = \operatorname{avg}_{\mathbf{x} \in \mathcal{M}} \left\| (\mathbf{R}\mathbf{x} + \mathbf{t}) - (\hat{\mathbf{R}}\mathbf{x} + \hat{\mathbf{t}}) \right\|_2, \quad (6.1)$$

where \mathcal{M} is a set of vertices of a 3D model, (\mathbf{R}, \mathbf{t}) and $(\hat{\mathbf{R}}, \hat{\mathbf{t}})$ are ground truth and predicted rotation and translation, respectively. As mentioned in [20], a predicted pose is considered to be correct if ADD calculated with this pose is less than 10% of a model diameter. However, in case of more complicated scenes, there is only a small fraction of poses falling into this category. Therefore, we also report ADD for the thresholds of 30% and 50% to give a broader overview of pose quality as well as to give an estimate of proportion of the poses which could be still a subject for further refinement.

6.4.2 Scalability Benchmark

The first benchmark was introduced with a goal to evaluate the method’s scalability with respect to the number of objects. The main requirement is to train a single network for all the objects available in the dataset and test it on a set of sequences containing all the objects. For this purpose we jointly evaluate a method on the sequences from 1 to 8,

Table 6.2: Scalability benchmark. Results of object detection and pose estimation.

Obj. ID	ADD 10%	ADD 30%	ADD 50%	Precision	Recall	mAP
1	23.04	68.30	81.04	0.86	0.98	0.85
2	17.85	58.02	75.87	0.62	0.89	0.56
3	14.18	55.41	72.40	0.93	0.92	0.88
4	9.09	36.74	55.43	0.66	0.79	0.53
5	11.70	43.23	58.60	0.93	0.98	0.91
6	0.00	3.51	12.13	0.67	0.68	0.46
7	15.15	44.12	60.74	0.66	0.68	0.45
8	18.67	42.74	52.28	0.33	0.24	0.08
9	2.26	13.71	27.58	0.60	0.62	0.37
10	3.15	19.63	30.96	0.89	0.86	0.76
11	0.71	4.07	9.87	0.82	0.98	0.80
12	0.50	3.74	7.48	0.47	0.40	0.19
13	2.25	20.22	35.63	0.58	0.62	0.36
14	2.88	24.04	40.19	0.41	0.52	0.21
15	0.00	1.88	5.52	0.33	0.40	0.14
16	0.00	4.70	9.94	0.48	0.36	0.17
17	0.72	3.37	17.35	0.63	0.42	0.26
18	0.16	2.24	4.32	0.49	0.63	0.32
19	7.70	28.63	45.88	0.94	0.92	0.89
20	23.22	63.60	75.73	0.92	0.96	0.91
21	8.37	36.12	50.66	0.39	0.23	0.09
22	10.04	35.97	55.02	0.76	0.78	0.59
23	16.18	62.49	82.20	0.95	0.99	0.95
24	4.08	29.25	55.78	0.19	0.15	0.03
25	9.76	39.50	50.62	0.75	0.88	0.66
26	15.01	51.83	68.98	0.78	0.79	0.63
27	13.47	51.78	71.47	0.68	0.76	0.52
28	26.17	63.30	78.07	0.80	0.92	0.73
29	13.97	38.59	57.68	0.93	0.94	0.89
30	48.63	88.71	96.34	0.80	0.98	0.79
31	6.43	21.85	36.68	0.83	0.86	0.74
32	0.00	5.95	10.71	0.23	0.17	0.04
33	11.40	45.69	64.99	0.93	0.97	0.91

which form a minimal subset of sequences with all 33 objects present. Object detection and pose estimation results for this benchmark are reported separately for each object.

From the results presented in Table 6.2, it can be seen that the best detection and pose estimation performance is achieved for bigger objects with distinct textures and geometric features (e.g., 28, 30), while detection of smaller low-textured or glossy industrial objects (e.g., 12, 13, 14) is a considerable challenge for DPOD. Besides, pose estimation results demonstrate that the DPOD detector does not scale particularly well for this task: for 17 out of 33 objects, ADD with 10% threshold is under 10%, and there are no instances for which the achieved score was higher than 50%.

6.4.3 Scene Benchmarks

Our dataset presents a collection of 13 scenes with a varying degree of difficulty, and each of these scenes represents a separate benchmark, where all the objects it contains are used for training. Except for the scenes with altered illumination conditions or texture changes (i.e., 10 and 11), we include all the scenes in the per-scene benchmarks. All the

Table 6.3: Result of object detection and pose estimation presented on two benchmarks: (1) per scene benchmark spanning over 11 scenes of the dataset and the (2) domain adaptation benchmark evaluating the detector’s generalization capabilities.

		Per Scene											Texture / Illumination		
Scene ID		1	2	3	4	5	6	7	8	9	12	13	5	10	11
Pose	ADD 10%	50.85	45.08	33.88	27.25	25.40	19.19	25.85	11.08	7.60	8.68	13.36	25.40	16.73	16.77
	ADD 30%	81.34	78.46	64.53	53.10	54.00	45.39	39.99	29.78	20.12	25.68	33.41	54.00	40.32	40.83
	ADD 50%	88.71	85.69	75.46	66.98	61.95	55.26	46.76	39.64	29.24	34.49	45.33	61.95	51.66	49.72
Detect.	Precision	0.79	0.62	0.76	0.76	0.65	0.46	0.68	0.51	0.26	0.33	0.13	0.65	0.57	0.43
	Recall	0.95	0.64	0.82	0.92	0.80	0.68	0.84	0.62	0.33	0.40	0.20	0.80	0.63	0.53
	mAP	0.82	0.48	0.72	0.78	0.64	0.36	0.64	0.41	0.14	0.20	0.04	0.64	0.42	0.32

object detection and pose estimation scores are averaged over all the objects present in the scene. In Table 6.3 the objects detection and pose estimation performance per scene is presented. As expected, DPOD demonstrates significantly better performance if both tasks in the sequences with the smaller number of objects, as well as less significant occlusions and no clutter. Also, low scores in both detection and pose estimation are reported for scene 8, which is composed exclusively of industrial objects.

6.4.4 Domain Adaptation Benchmark

The main goal of introducing the domain adaptation benchmark is to test the robustness of a method to significant changes in lighting conditions and objects textures. It is composed of three scenes (5, 10 and 11) with the same set of objects, but differing in terms of illumination and the color of object surfaces. Scene 5 was captured in ambient lighting conditions with no alteration of the objects’ appearance. In contrast, in scene 10 we used a spotlight to project light of different colors and intensity onto the imaged objects to introduce considerable variations in illumination, whereas in scene 11, we applied paint on the objects’ surfaces to alter their texture. For this benchmark, object detection and 6D pose estimation scores are presented per scene.

As can be seen from results in Table 6.3, performance in both detection and pose estimation is notably better in the cases of no added illumination or texture changes. Under normal conditions the resulting poses are 37% more accurate based on ADD with a 10% threshold when compared to those under altered conditions. Also, object detection performance falls far behind under altered conditions compared to normal conditions, resulting in 46% and 59% lower mAP scores for the scenes with variations in light and texture, respectively. These results suggest that there is a lot of room for further adaptation of the DPOD detector to changing environments.

6.4.5 Drawbacks of Training on Real Data

The main reason we exclusively use synthetic data for training in our benchmarks comes from the inability of the detectors trained on a small-scale set of real data to generalize to different environments, which may differ in background, illumination, texture and other scene characteristics.

Table 6.4: Pose estimation results in terms of ADD 10% metric on LineMOD sequences (LM) and HomebrewedDB (HB) sequence with the same objects.

	Dataset	Method	Benchvise	Phone	Driller
Real	LM	YOLO6D [40]	81.80	47.74	63.51
		DPOD	95.34	74.24	97.72
	HB	YOLO6D [40]	15.30	6.50	0.10
		DPOD	57.24	33.09	62.82
Synthetic	LM	SSD6D + Ref. [56]	44.30	26.20	26.90
		DPOD	66.76	29.08	66.60
	HB	SSD6D + Ref. [56]	59.40	29.30	25.10
		DPOD	70.89	35.56	66.42

To support our claim, we selected two state-of-the-art detectors, YOLO6D [40] and our DPOD, trained on real LineMOD data and tested them on our new sequence containing three LineMOD objects – benchvise, drill and phone (Scene 2 in Fig. 6.3). This sequence contains a minimal number of occlusions and no clutter and may be regarded as one of the simplest in the dataset. Besides, it was captured with the same camera as LineMOD sequences (PrimeSense Carmine 1.09), making it similar in terms of color scheme and noise. Performance comparison of both detectors on the LineMOD sequence and our sequence can be seen in the "Real" section of Table 6.4. Despite demonstrating very good performance when tested on sequences from LineMOD, when run on our sequence with LineMOD objects, both detectors experience a significant drop in pose estimation accuracy in terms of the ADD 10% metric for all the objects. Specifically, the pose estimation accuracy of DPOD dropped more than 2 times for the phone, while for YOLO6D [40] there were nearly no correctly predicted poses for the drill.

As the second part of the experiment, we evaluated two detectors designed for training on synthetic data, DPOD and SSD6D with model-based refinement [56], on the same new sequence with LineMOD objects. From the results presented in the "Synthetic" section of Table 6.4 one can see that there is no significant difference between the results obtained on LineMOD and HomebrewedDB sequences, meaning that there is little to no overfitting to a particular dataset. In 5 out of 6 cases, the detectors demonstrated better performance on a simpler HomebrewedDB sequence, thus showing more predictable behavior than in the case of training on real data. Moreover, when trained on synthetic data, DPOD can boast higher ADD 10% scores for all the three objects in the HomebrewedDB test sequence. This fact once again supports the claim that training detectors on synthetic data leads to better generalization, in contrast to training on limited real data, when even seemingly insignificant changes in the environment turn out to be a decisive factor leading to a significant decrease in pose estimation accuracy.

6.5 Conclusion

In this chapter, we presented a new challenging dataset for 6D object detection targeting training from synthetic data and covering the most important properties a solid object detector should have, namely scalability with respect to the number of objects and robustness to occlusions, illumination and appearance changes. This new dataset contains 33 objects spanning 13 scenes of various difficulty. To be able to compare the detectors to each other, we defined a set of benchmarks that test all of the above mentioned properties. Finally, we developed and presented a comparably simple, yet robust and fully automated pipeline that we used to build our dataset. We hope it will allow other researchers to be able to create their own datasets and thus promote research in 6D pose estimation.

Part II

Domain Adaptation in Depth and RGB

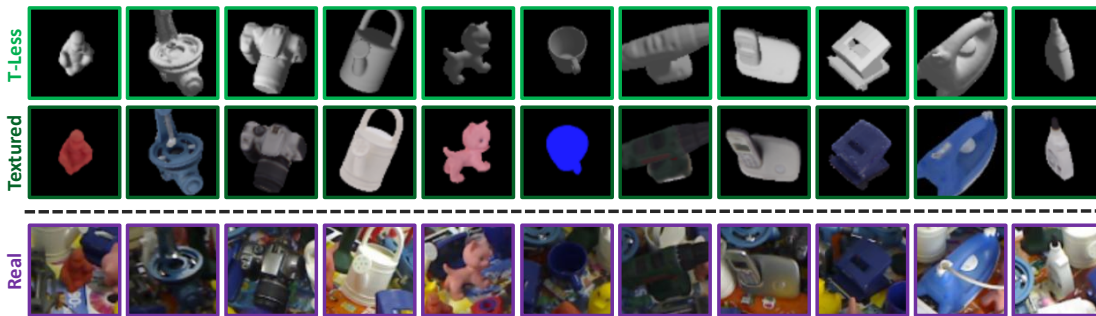


Figure 6.5: Realism gap. Top: synthetic textureless models. Middle: synthetic textured models. Bottom: real images. Networks trained on one type of data significantly underperform when tested on another domain due to a large visual discrepancy.

Recent progress in computer vision has been dominated by deep neural networks trained over large datasets of annotated data. Collecting those is however a tedious if not impossible task. In practice, and especially in the industry, 3D CAD models are widely available, but access to real physical objects is limited and often impossible (e.g., one cannot capture new image datasets for every new client, product, part, environment, etc.). It thus became common to leverage such data to train recognition methods e.g., by rendering huge datasets of relevant synthetic images and their annotations.

However, the development of exhaustive, precise models behaving like their real counterparts is often as costly as gathering annotated data (e.g., acquiring precise texture information to render proper images from CAD data actually imply capturing and processing images of target objects). As a result, the salient discrepancies between model-based samples and target real ones (known as *realism gap*) still heavily impair the application of synthetically-trained algorithms to real data (cf. Fig. 6.5). Research in *domain adaptation* thus gained impetus the last years. Various solutions have been proposed, but most of them require access to real relevant data (even if unlabeled) or access to synthetic models too precise for scalable real-world use-cases (e.g., access to realistic textures for 3D models).

In this part, we introduce two methods tackling the problem of domain gap. The first method is based on an idea of the reverse domain adaptation. Instead of the common pipeline of training networks to map synthetic images into the real domain, we train networks to map real images back to the synthetic domain. This not only allows to improve the overall performance of the system, but also introduces some long-standing benefits: the downstream network can be trained purely on synthetic data and never has to be retrained, as opposed to the standard pipelines.

The second method utilizes an adversarial domain randomization procedure to make the task network robust to the possible appearance changes defined by specifically designed differentiable modules. The procedure uses a newly introduced deception network that modifies input images while still preserving their geometrical meaning. The task network is trained together with the deception network utilizing a min-max procedure. As a result, the final task network becomes much more robust to possible image changes.

7 Reverse Domain Adaptation

While convolutional neural networks are dominating the field of computer vision, one usually does not have access to the large amount of domain-relevant data needed for their training. Therefore, it has become a common practice to use available synthetic samples along domain adaptation schemes to prepare algorithms for the target domain. In this chapter, we introduce a reverse domain adaptation pipeline tackling this problem from a different angle: we map unseen target samples into the synthetic domain used to train task-specific methods. Denoising the data and retaining only the features these recognition algorithms are familiar with, our solution greatly improves their performance. As this mapping is easier to learn than the opposite one (i.e., to generate realistic features to augment the source samples), we demonstrate how our whole solution can be trained purely on synthetic depth or RGB data and still perform better than methods trained with domain-relevant information (e.g., real images or realistic textures for the 3D models). Applying our approach to object recognition from texture-less CAD data, we present the depth and RGB generative networks which fully utilize the purely geometrical information to learn robust features and to achieve a more refined mapping for unseen images.

7.1 Introduction

In this chapter, we introduce an approach tackling domain adaptation and realism gap from a different angle. It is composed of a generative network to map unseen real samples toward a relevant, easily-available synthetic domain with a goal to improve recognition

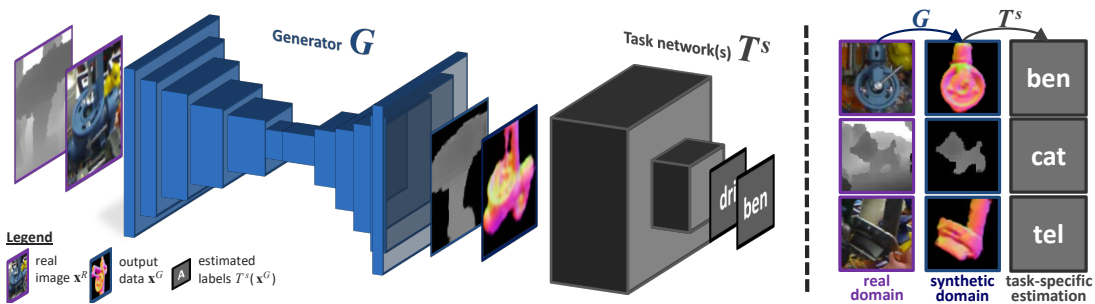


Figure 7.1: Real \rightarrow synthetic mapping. Trained on augmented data from 3D models, our network G can map real scans (either depth or RGB) to the synthetic domain (either depth or surface normals). The pre-processed data can then be handed to various recognition methods (T^S) to improve their performance.

for methods themselves trained on this noiseless synthetic modality (see Fig. 7.1). We demonstrate that it is sensible to train task-specific networks on noiseless information so they learn clean discriminative features, and then develop a pre-processing mapping function from real to synthetic data; rather than to focus on developing or learning pseudo-realistic noise models to train against.

Applied to CAD-based recognition in both depth and color pictures, our approach is based on the assumptions that real-world images can be mapped to the synthetic domain; and that, in absence of any real training data, this mapping can be learned by recovering the synthetic samples altered by a stochastic noise source. Since our method only needs to eliminate noise and retain features, it performs better than usual generative solutions for domain adaptation, which learn the more difficult task of generating complex features to mimic the target data. As long as the synthetic domains contain all relevant features and as long as those features are contained in real images, our approach successfully enhances recognition, as demonstrated through our empirical evaluation. In summary, we are making the following contributions:

- **Synthetic modality regression for domain adaptation** – We propose a novel framework to learn a mapping from unseen real depth and RGB data to relevant synthetic domains, denoising and recovering the information needed for further recognition. Our solution not only covers the real-synthetic gap, but also takes care of cross-modality mapping. More specifically, we present how color images can be mapped to normal maps, to help pose-regression and classification in absence of reliable texture information for training.
- **Decoupling domain adaptation from recognition** – Most domain adaptation schemes constrain the methods training by adding pseudo-realistic or noisy features to the training set, editing the architecture or losses, etc. In our framework, task-specific algorithms simply learn on available, relevant synthetic data, while separately our network G is trained on noisy data to map them into the selected synthetic domain. This decoupling makes training more straightforward, and allows G to be used along any number of recognition methods. We furthermore observe better results compared to recognition methods directly trained on augmented data and even to solutions using real information for training.
- **Performance in complete absence of real training data** – Domain adaptation approaches usually assume the realism gap to be already partially bridged, requiring access to some target domain images or realistic textured synthetic models. Opting for recognition tasks with texture-less CAD data for only prior, we demonstrate how our pipeline can be trained on purely synthetic data and still generalize well to real situations. For that we leverage an extensive augmentation pipeline, used as a noise source applied to training samples so our solution learns to denoise and retain the relevant features.
- **Multi-task network with self-attentive distillation for RGB** – To tackle a more challenging RGB domain mapping we develop a custom generator with

multiple convolutional decoders for each relevant synthetic modality (e.g., normal maps, semantic masks, etc.), and a distillation module on top making use of self-attention maps to refine the final outputs.

7.2 Methodology

Formalizing our problem, let $X_c^S = \{\mathbf{x}_{c,i}^S \mid \forall i \in N_c^S\}$ be a dataset made of a number N_c^S of uncluttered, noiseless training samples \mathbf{x}_c^S of class c . Let $X^S = \{X_c^S \mid \forall c \in C\}$ be the complete clean training dataset. We similarly define X^R , the set of target C -related real data, completely unavailable for training. Note that samples x^R can also be of a different modality than \mathbf{x}^S (e.g., \mathbf{x}^R being color images while \mathbf{x}^S being normal maps, when no textures were available to render synthetic color images). Finally, let $T(\mathbf{x}; \theta_T) \rightarrow \tilde{\mathbf{y}}$ be any recognition algorithm which given a sample \mathbf{x} returns an estimate $\tilde{\mathbf{y}}$ of a task-specific label or feature \mathbf{y} (e.g., class, pose, mask image, hash vector, etc.). We define as T^S the method trained on noiseless X^S .

Given this setup, our pipeline trains a function G purely on synthetic data to learn a mapping from complex C -related instances to their corresponding clean signal. To achieve this when no domain-relevant data is available for training, we describe in this section how G is trained against a data augmentation pipeline $A(\mathbf{x}^S, z) \rightarrow \mathbf{x}_z^A$, with noise vector z , randomly defined at every training iteration, and the resulting noisy data \mathbf{x}_z^A . Our training approach assumes that G removes the artificially introduced noise z such that only the original synthetic signals \mathbf{x}^S are retained. Thus, G can be seen as a noise filter that removes unneeded elements in input data, and can be also applied over the domain X^R of real samples as long as synthetic information can be extracted from them. We demonstrate that, in the case of CAD-based visual recognition, we can indeed define a new generative method G fully utilizing the synthetic modalities, and a complex and stochastic augmentation pipeline A to train G against, such that G maps real images into the learned synthetic domain with high accuracy. We even demonstrate how this process $T^S(G(\mathbf{x}^R)) = \tilde{\mathbf{y}}^G$ is more accurate compared to $T^A(\mathbf{x}^R) = \tilde{\mathbf{y}}^A$, with the task-specific algorithm T^A directly trained on data augmented by A . Though we focus the rest of the chapter on CAD-based visual recognition for clarity, the principles behind our solution can be directly applied to other use-cases.

7.2.1 GAN-Based Architecture for Depth

We start with describing the architectural details of our pipeline for depth data G_{DEP} shown in Fig. 7.2. Working in the depth domain is beneficial since it automatically solves the problem with texture-less data by only considering the geometry of the models. Following recent works in domain adaptation [80, 67, 65, 6, 66], we adopt a generative adversarial architecture. We first define a generator function $G_{DEP}(\mathbf{x}; \theta_G) \rightarrow \mathbf{x}^G$, parametrized by a set of hyper-parameters θ_G and conditioned by image \mathbf{x} to generate the output version \mathbf{x}^G . During training, the task of G_{DEP} is to restore the noiseless depth data from its augmented version, i.e., to obtain $\mathbf{x}^G \simeq \mathbf{x}^S$. Then, during testing, provided

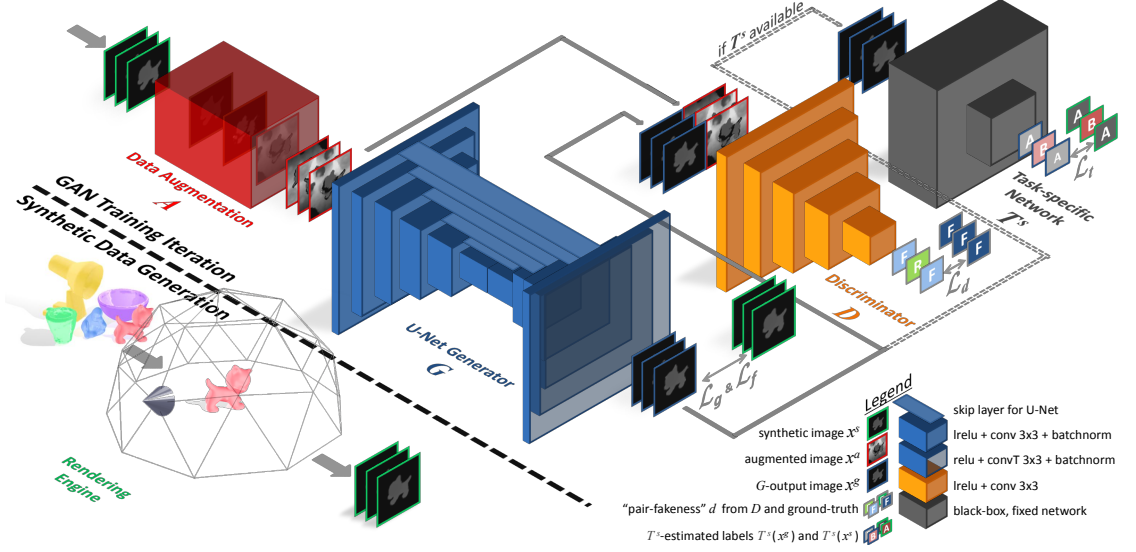


Figure 7.2: Training of the depth-processing network G_{DEP} . Following the conditional GAN architecture, a generator G_{DEP} is trained against a discriminator D to recover the original noiseless image from a randomly augmented, synthetic one. Its loss combines similarity losses \mathcal{L}_g and \mathcal{L}_f , the conditional discriminator loss \mathcal{L}_d , and optionally a feature-similarity loss \mathcal{L}_t if a task-specific method T^s is provided at training time.

a real image \mathbf{x}^R , G_{DEP} generates an image \mathbf{x}^G which can be passed to T^S for better recognition results. We oppose to G_{DEP} a discriminator network $D(\mathbf{x}_z^A, \mathbf{x}; \theta_D) \rightarrow d$, which estimates the probability d that \mathbf{x} is the original noiseless sample \mathbf{x}^S and not the recovered image $\mathbf{x}^G = G(\mathbf{x}_z^A)$ [67]. The typical objective for such a solution is:

$$\mathcal{L}_{DEP} = \arg \min_G \max_D \alpha \mathcal{L}_d(G_{DEP}, D) + \beta \mathcal{L}_g(G_{DEP}) \quad (7.1)$$

$$\text{with } \mathcal{L}_d(G_{DEP}, D) = \mathbb{E}_{\mathbf{x}^S, z} [\log D(\mathbf{x}_z^A, \mathbf{x}^S; \theta_D)] + \mathbb{E}_{\mathbf{x}^S} \left[\log \left(1 - D(\mathbf{x}_z^A, G(\mathbf{x}_z^A); \theta_G) \right) \right] \quad (7.2)$$

$$\mathcal{L}_g(G_{DEP}) = \mathbb{E}_{\mathbf{x}^S, z} \left[\|\mathbf{x}^S - G(\mathbf{x}_z^A; \theta_G)\|_1 \right] \quad (7.3)$$

As explained in [67, 6, 66], the conditional loss function \mathcal{L}_d , weighted by a coefficient α represents the cross-entropy error for a classification problem where $D(\mathbf{x}_z^A, \mathbf{x}; \theta_D)$ estimates if $(\mathbf{x}_z^A, \mathbf{x})$ is a “fake” or “real” pair (i.e., \mathbf{x} generated by G_{DEP} from \mathbf{x}_z^A , or \mathbf{x} original sample from X^S).

This is complemented by a simple similarity loss \mathcal{L}_g , an $L1$ distance weighted by a parameter β , to force the generator to stay close to the ground-truth. However, since the images we are comparing are supposed to be noiseless with no background, this loss can be augmented with another one focusing on the foreground similarity. We thus introduce a complementary *foreground* loss \mathcal{L}_f weighted by a factor γ , inspired by the

content-similarity loss of Bousmalis et al. [66]. Given m^S the binary foreground mask obtained from \mathbf{x}^S ($m_{ij}^S = 1$ if $\mathbf{x}_{ij}^S \neq 0$ else $m_{ij}^S = 0$) and \odot the Hadamard product:

$$\mathcal{L}_f(G_{DEP}) = \mathbb{E}_{\mathbf{x}^S, z} \left[\left\| (\mathbf{x}^S - G_{DEP}(\mathbf{x}_z^A; \theta_G)) \odot m^S \right\|_1 \right] \quad (7.4)$$

In the case that the target recognition method $T^S(\mathbf{x}) \rightarrow \tilde{\mathbf{y}}$ is provided ready-to-use for the GAN training, a third task-specific loss can be applied (similarly to [66, 80], but with a fixed pre-trained network). Weighted by another parameter δ , \mathcal{L}_t can be used to guide G_{DEP} , to make it more *aware* of the information this specific T^S tries to uncover:

$$\mathcal{L}_t(G_{DEP}) = \mathbb{E}_{\mathbf{x}^S, z} \left[\left\| T^S(\mathbf{x}^S) - T^S(G_{DEP}(\mathbf{x}_z^A; \theta_G)) \right\|_2 \right] \quad (7.5)$$

This formulation has two advantages: no assumptions are made regarding the nature of $\tilde{\mathbf{y}} = T^S(\mathbf{x})$, and no ground-truth \mathbf{y} is needed since \mathcal{L}_t only depends on the difference between the two estimations made by T^S . Unlike PixelDA [66], our training is thus both uncoupled from the real domain (no real image \mathbf{x}^R used) and completely unsupervised (no prior on the nature of the labels/features, and neither ground-truth \mathbf{y}^R of \mathbf{x}^R nor \mathbf{y}^S of \mathbf{x}^S used). Taking into account the newly introduced \mathcal{L}_f and optional \mathcal{L}_t , the expanded loss guiding our method toward its objective is thus finally:

$$\mathcal{L}_{DEP} = \alpha \mathcal{L}_d(G_{DEP}, D) + \beta \mathcal{L}_g(G_{DEP}) + \gamma \mathcal{L}_f(G_{DEP}) + \delta \mathcal{L}_t(G_{DEP}) \quad (7.6)$$

Following the standard procedure [68], the minmax optimization is achieved by alternating at each training iteration between (1) fixing θ_G to train D over a batch of $(\mathbf{x}_z^A, \mathbf{x}^S)$ and $(\mathbf{x}_z^A, G_{DEP}(\mathbf{x}_z^S; \theta_G))$ pairs, updating through gradient descent θ_D to maximize \mathcal{L}_d ; (2) fixing θ_D to train G_{DEP} , updating through gradient descent θ_G to minimize the combined loss.

7.2.2 Multi-Modal U-Net Architecture for RGB

While working in depth domain automatically solves a problem with texture-less data and has a smaller realism gap, mainly defined by the sensor’s imperfections, algorithms working in RGB data are more desirable due to the widespread availability of RGB sensors. Since the domain gap is larger in the RGB domain, it requires a more advanced network architecture (G_{RGB}) to achieve competitive results (see Fig. 7.3). As demonstrated by previous works in multi-task learning [149, 150, 151, 152, 153, 154], it is often advantageous to train a network on several tasks (even when considering only one), as the synergy between the tasks can make each of them perform better, and make the common layers for feature extraction more robust and focused on abstract, cross-modality information.

We thus adopt such a scheme to guide our generator G_{RGB} in its main task of extracting the chosen synthetic features from noisy samples. Not limited to the scarce, sometimes imprecise, annotations of real training datasets, we can rely on a multitude of different synthetically-rendered modalities. For industrial CAD-based recognition, G_{RGB} would learn to map real images into a geometrical domain (normal and/or depth

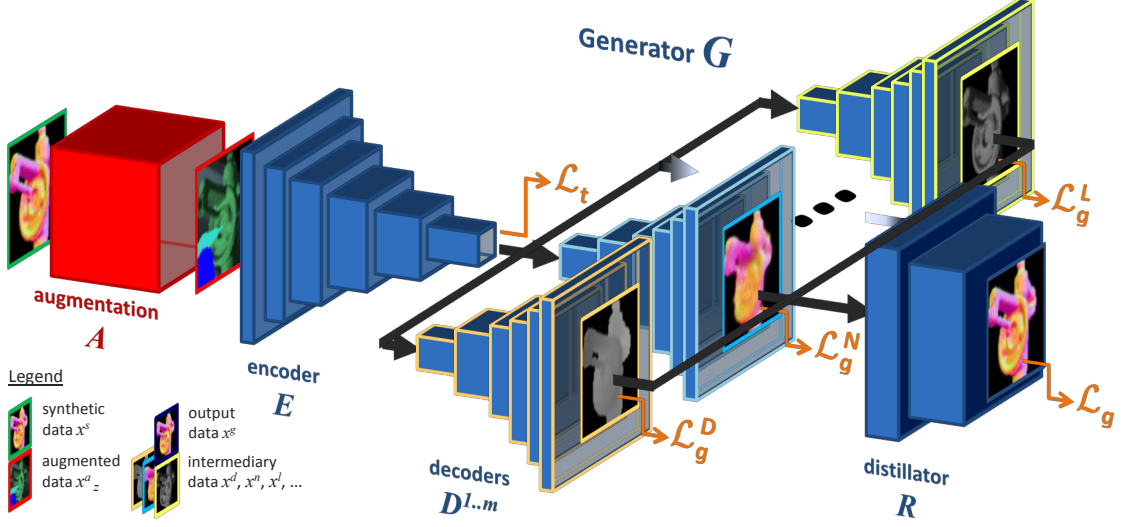


Figure 7.3: Training of the RGB multi-modal network G_{RGB} . Taking full advantage of available synthetic data e.g., texture-less CAD models, G consists of a custom multi-modal pipeline with self-attentive distillation, trained to recover noiseless geometrical and semantic modalities from randomly augmented synthetic samples.

maps), using for sub-tasks the regression of depth and normal maps, semantic or contour mask, etc. (cf. Section 7.2.3).

Inspired by previous multi-modal generative pipelines [152, 153, 154], our network is composed of a single convolutional encoder E and m decoders D^{mod} , with m number of sub-tasks. For our implementation, we only consider up to 4 sub-tasks – normal and depth regression, semantic segmentation, foreground lightness evaluation – though it would be straightforward to add more (e.g., contour extraction as in [154]). Each intermediary modality is fully decoded in order to be compared to its synthetic ground-truth. Each generative loss \mathcal{L}_g^{mod} (L1 distance for images, cross-entropy for binary masks) is back-propagated through its decoder, then jointly through the common encoder (cf. Fig. 7.3).

A triplet loss with dynamic margin \mathcal{L}_t from Chapter 3 can be optionally added at the network bottleneck to improve the feature distribution in the embedding space, using task-specific metrics to push apart encoded features of images from semantically-different images, while bringing together features of similar elements.

$$\mathcal{L}_t(E) = \sum_{(\mathbf{x}_b, \mathbf{x}_p, \mathbf{x}_n) \in X} \max \left(0, 1 - \frac{\|E(\mathbf{x}_b) - E(\mathbf{x}_n)\|_2^2}{\|E(\mathbf{x}_b) - E(\mathbf{x}_p)\|_2^2 + m} \right) \quad (7.7)$$

with \mathbf{x}_b the input image used as binding anchor, \mathbf{x}_p a positive or similar sample, \mathbf{x}_n a negative or dissimilar one, and m the task-specific margin setting the minimum ratio for the distance between similar and dissimilar pairs of samples. For the task of instance classification and pose estimation (ICPE), we set $m = 2 \arccos(|\mathbf{q}_b \cdot \mathbf{q}_p|)$ if \mathbf{x}_b and \mathbf{x}_p are images of the same class, else $m = n$ (with \mathbf{q}_b and \mathbf{q}_p the pose quaternions corresponding

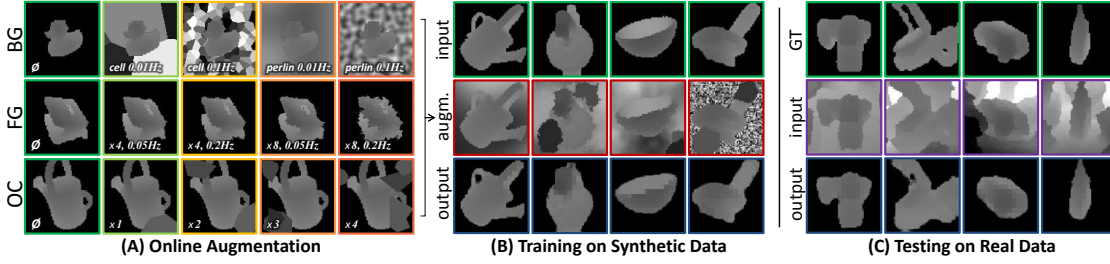


Figure 7.4: G_{DEP} augmentation, training, and testing results: (A) Depth augmentation examples (BG background noise; FG foreground distortion; OC occlusions) for different noise amplitudes or types; (B) Validation results, showing how our solution learns during training to recover the clean images (*input*) from their augmented versions (*augm.*); (C) Test results on real data (compared to ground-truth GT).

to \mathbf{x}_b and \mathbf{x}_p , and $n > \pi$ a fixed margin), following the dynamic margin loss definition from Chapter 3.

Further distinguishing our solution from usual multi-modal autoencoders, we add skip connections from each encoding block to its reciprocal decoding block. As demonstrated in previous works [152, 7], passing high-resolution features from the contracting layers along the outputs of previous decoding blocks not only improves the training by avoiding vanishing gradients, but also guides the decoding blocks in upsampling and localizing the features. We observe a clear performance boost from this change, as shown in Table 7.6.

Distillation with Self-Attention

If training the target decoder along others already improves its performance by synergy, several works [154, 151, 155] demonstrated how one can further take advantage of multi-modal architectures by adding a distillation module on top of the decoders, merging their outputs to distill a final result. In their work [154], *Pad-Net* authors present several distillation strategies, with the most efficient one making use of the attention mechanism [156, 157, 158] to better weigh the cross-modality merging, bringing forward the most relevant features for the final modality.

Using this insight, we built our own module R to refine the target results from the partially re-encoded intermediary modalities by using self-attention computations [159]. This mechanism, adapted by Zhang et al. [160] for image generation and is used to efficiently model relationships between widely-separated spatial regions. Instantiating and applying this process to each re-encoded modality, we sum the resulting feature maps, before decoding them to obtain the final output. This new distillation process not only allows to pass messages between the intermediary modalities, but also between distant regions in each of them.

Our distillator is trained jointly with the rest of the generator, with a final generative loss \mathcal{L}_g (L1 distance here) applied to the distillation results. Not only our whole generator

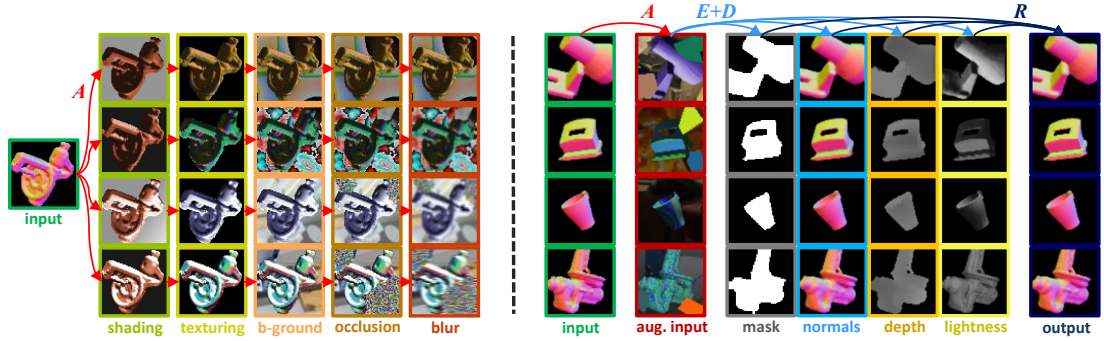


Figure 7.5: G_{RGB} augmentation and training results. On the *left*, we demonstrate how normal maps are step by step transformed into complex, random color images by our online augmentation pipeline. On the *right*, we present how G_{RGB} is trained on these images, learning to map them back to the noiseless geometrical information.

can thus be efficiently trained in a single pass, but no manual weighing of the sub-task losses is needed, as the distillator implicitly covers it.

7.2.3 Learning from Purely Geometrical CAD Data

A key prior in this work is the unavailability of target real images (not only target labels). Only synthetic depth images \mathbf{x}^S are used to train our solution, and presumably the recognition method. The only requirement is thus the availability of the synthetic data used to train T^S , or at least the 3D models of the class objects C to generate X^S from them.

7.2.3.1 Synthetic Data Generation

Depth Domain To generate synthetic depth images we follow the procedure from Chapter 3. In particular, we render objects from the viewpoints defined as vertices of an icosahedron centered on the target object. By repeatedly subdividing each triangular face of the icosahedron, additional viewpoints, and thus a denser representation, can be created. Furthermore, in-plane rotations are added at each position by rotating the camera around the axis pointing at the object center. Rotation invariance of the objects is also taken into account (e.g., for pose estimation applications), as rotation-invariant objects might look exactly the same from different poses, confusing recognition methods. To deal with this, the pipeline is configured to trim the poses for those objects, such that every image is unique.

RGB Domain Without any relevant texture information, usual dataset rendering and training methods for the color domain cannot be directly applied. Since only geometrical information is made available, we select the surface normal and/or depth domains as target modality for the mapping performed by G_{RGB} . For this reason we use a simple renderer to generate noiseless normal and depth maps for each class c from a large set

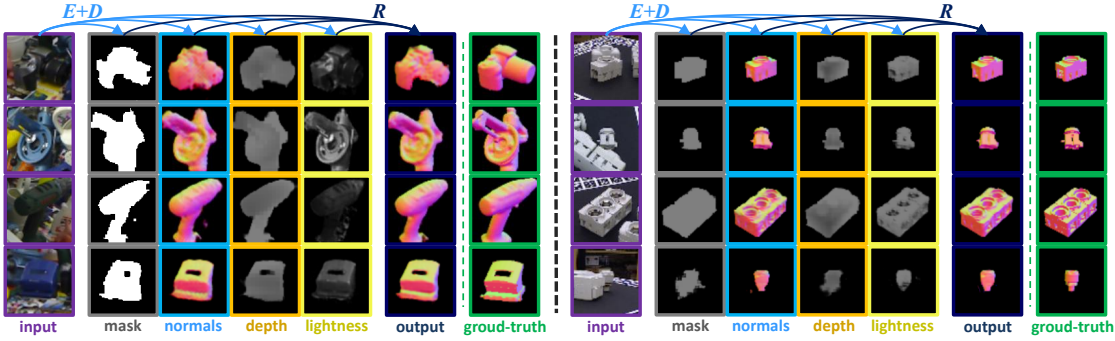


Figure 7.6: G_{RGB} qualitative results. Intermediary and final mappings when applying G_{RGB} trained on purely synthetic data to real samples, on LineMOD [3] and T-LESS [5] datasets.

of relevant viewpoints again following the procedure from Chapter 3. This dataset of geometrical mappings is both used as ground-truth for the final outputs of G_{RGB} and some of its sub-tasks, and as inputs for the augmentation pipeline A deployed when no color data is available to train G_{RGB} .

7.2.3.2 Online Data Augmentation

$A(\mathbf{x}^S, z) \rightarrow \mathbf{x}_z^A$ is an extensive online augmentation pipeline we designed, parametrized by a noise vector z randomly sampled at every call from a k -dimensional finite set \mathbb{Z}^k , with k the number of augmentation parameters. To make up for the complete lack of appearance and clutter information, A follows the principle of *domain randomization* [87], i.e., it is meant to add enough visual variability to the training inputs so that the trained method can generalize to real unseen samples. This means conceiving an augmentation pipeline with a large enough $|\mathbb{Z}^k|$. Depending on the type of the input data, our augmentation pipeline A comprises the following components:

- **Simple random shading (RGB):** A first takes the provided normal maps and convert them into color images by applying simple Blinn-Phong shading [161] as described in Algorithm 1. Randomly sampling ambient and directional light sources, as well as diffusion and specular color factors for the objects, the provided surface normals are used to compute the diffuse and specular lightness maps through direct matrix products. Since distance information is lost in normal maps, this shading model is simplified by supposing the light sources at an infinite distance, hence the same light source vector for every surface point. This way, one can easily simulate an infinity of lighting conditions, returning the resulting lightness map.

Lighting parameters, defined through the augmentation noise vector z , are sampled using uniform distributions e.g., $a \sim \mathcal{U}(0.05, 0.3)$, $d \sim \mathcal{U}(0.1, 0.8)$, $s \sim \mathcal{U}(0, 0.1)$, $s_p \sim \mathcal{U}(0.9, 1.1)$, etc.

- **Stochastic texturing (RGB):** Given the lack of relevant texture information, random texture maps are procedurally generated using noise functions. In partic-

Algorithm 1: Approximate Blinn-Phong shading [161] from normal maps

Input: $N \in \mathbb{R}^{h \times w \times 3}$ normal map, $L \in \mathbb{R}^3$ directional light vector, $a \in \mathbb{R}^{\#}$ RGB ambient light coefficient, $d \in \mathbb{R}^{\#}$ RGB diffusion coefficient, $s \in \mathbb{R}^3$ RGB specularity coefficient, $s_p \in \mathbb{R}$ specular hardness, $f_x \in \mathbb{R}^2$ pixel focal range used to render N

Output: $M \in \mathbb{R}^{h \times w \times 3}$ color lightness map

```

/* - Simplification #1: we recover an approximate viewer vector
   V from N indices and f_x. */
/* - Simplification #2: we suppose the light source at +∞
   distance, hence the same L for every surface point. */
/* - Note: we use Einstein notation for matrix-vector operations.
   */
/*
/* View vector approximation: */
1 V ← { (j, i, 1) }_{j=0, i=0}^{h,w};
2 V_j ← - (V_j - h/2) / f_{x,j}; V_i ← - (V_i - w/2) / f_{x,i};
3 V ← V / ||V||;
/* Computation of half-way vector map: */
4 H ← V + L;
/* Diffuse shading: */
5 D^{ij} ← N^{ij}_k L^k;
/* Specular shading: */
6 S^{ij} ← (N^{ij}_k H^k_{ij})^{s_p};
/* Adding all contributions (given e_{ijc} = e_i ⊗ e_j ⊗ e_c) : */
7 M^{ijc} ← min ( max(a · e_{ijc} + d · D^{ij} e_c + s · S^{ij} e_c, 0) , 1);
8 return M

```

ular, Perlin noise [163], cellular noise [164], and white noise are used, sampled from the uniform distribution $\mathcal{U}(0.0001, 0.1)$. Downsampled to 2D vector maps, the original normals are used to index the generated textures; to achieve a more “organic” appearance, with patterns sometimes following some of the shape features.

- **Background noise / addition (Depth / RGB):** To teach G to focus on the representations of the captured objects and ignore the rest, A fills the background of the training data with several noise types commonly used in procedural content generation, e.g., fractal Perlin noise [163], cellular noise [164], and white noise. Noise frequencies for all modalities are sampled from the uniform distribution $\mathcal{U}(0.0001, 0.1)$. In case of the RGB scenario, backgrounds are added to the rendered images using random patches from a publicly available image dataset (i.e. MSCOCO [165]). Lightness maps from the shading step are furthermore used to homogenize the background brightness.

Algorithm 2: Random polygon generation [162]

```

Input:  $z \in \mathbb{Z}^k$  noise vector
Output:  $p = \{p_i \in \mathbb{R}^2\}_{i=0}^{N_{vert}}$  polygon points
/* occlusion parameters sampling: */
1  $c_x, c_y, r_{ave}, N_{vert}, \epsilon, \sigma \leftarrow \text{sampleFromVector}(z)$ ;
/* angle steps generation: */
2  $sum = 0$ ;
3 for  $i \in \{1, \dots, N_{vert}\}$  do
4 |  $\delta\theta_i \leftarrow \mathcal{U}(2\pi/N_{vert} - \epsilon, 2\pi/N_{vert} + \epsilon)$ ;
5 |  $sum \leftarrow sum + step$ ;
6 end
/* steps normalization: */
7  $k \leftarrow sum/(2\pi)$ ;
8 for  $i \in \{1, \dots, N_{vert}\}$  do
9 |  $\delta\theta_i \leftarrow \delta\theta_i/k$ ;
10 end
/* polygon points generation: */
11  $\theta_1 \leftarrow \mathcal{U}(0, 2\pi)$ ;
12 for  $i \in \{1, \dots, N_{vert}\}$  do
13 |  $r \leftarrow \mathcal{N}(r_{ave}, \sigma)$ ;
14 |  $p_i \leftarrow (c_x + r \cos(\theta_i), c_y + r \sin(\theta_i))$ ;
15 |  $\theta_i \leftarrow \theta_i + \delta\theta_i$ 
16 end
17 return  $p$ 

```

- **Random occlusion (Depth / RGB):** Occlusions are introduced to further simulate clutter, but also so that G can learn to recover hidden or lost geometrical information. Based on [162], occluding polygons are generated by walking around the image, taking random angular steps and random radii at each step; then by painting it on top of the images with random noise or textures if provided.

The polygon’s complexity is defined by two parameters: σ (“spikeyness”), which controls how much point coordinates vary from the radius r_{ave} , and ϵ (“irregularity”), which sets an error to the default uniform angular distribution. Variables c_x and c_y define the polygon center; r_{ave} its average radius; $\delta\theta$ and θ are a vector of angle steps, and a vector of angles respectively; and $h \times w$ are the image dimensions (equal to 64×64 px here). The pseudocode is listed in Algorithm 2.

Occlusion parameters are also set by the noise vector z . In our experiments, the following sampling distributions are used (with \mathcal{B} – Bernoulli, \mathcal{U} – Uniform, and \mathcal{N} – Gaussian):

- $\mathcal{B}(\mathcal{U}(0, h/4), \mathcal{U}(h/4, l))$ for c_x , $\mathcal{B}(\mathcal{U}(0, w/4), \mathcal{U}(w/4, l))$ for c_y ;
- $\mathcal{U}(10, l/4)$ for r_{ave} , with $l = \min(h, w)$;

Algorithm 3: Foreground distortion

```

Input:  $\mathbf{x} \in \mathbb{R}^{h \times w}$  depth image,  $z \in \mathbb{Z}^k$  noise vector
Output:  $\mathbf{x}_a \in \mathbb{R}^{h \times w}$  augmented depth image
/* sampling distortion parameters from the noise vector: */
1  $f_X, f_Y, f_Z, wr_{XY}, wr_Z \leftarrow \text{sampleFromVector}(z)$ ;
/* generating 3D vector field: */
2  $\mathbf{v}_d[0] \leftarrow \text{fast2DNoise}(f_X)$ ;
3  $\mathbf{v}_d[1] \leftarrow \text{fast2DNoise}(f_Y)$ ;
4  $\mathbf{v}_d[2] \leftarrow \text{fast2DNoise}(f_Z)$ ;
/* applying 3D distortion: */
5 for  $i \in \{1, \dots, h\}$  do
6   for  $j \in \{1, \dots, w\}$  do
7      $\mathbf{x}_a(i, j) \leftarrow \mathbf{x}(i + wr_{XY} * \mathbf{v}_d[0][i, j],$ 
8        $j + wr_{XY} * \mathbf{v}_d[1](i, j))$ 
9        $+ wr_Z * \mathbf{v}_d[2](i, j)$ 
6   end
5 end
10 return  $x_a$ 

```

- $\mathcal{U}(3, 10)$ for N_{vert} ;
- $\mathcal{U}(0, 0.5)$ for σ .

- **Foreground distortion (Depth):** Similarly to Simard et al. [166], images undergo random distortions (as an inexpensive way to simulate sensor noise, objects wear-and-tear, etc.). The foreground distortion component warps the input image using a random vector field. As a first step, three Perlin noise images are generated to form a three-dimensional offset vector field \mathbf{v}_d . The first two dimensions are used for depth value distortions in X and Y axes of the image space, whereas the third dimension is applied to the Z image depth values directly. Since values of v_d vary between -1 and 1 , warping factors $wr \in \mathbb{R}^3$ that control the distortion effect are introduced. Once the 3D offset vector generated, it is applied to the input image \mathbf{x} to generate the output image \mathbf{x}_a . The pseudocode is presented in Algorithm 3.

In our pipeline, the noise frequencies as well as the warping factors are defined by the randomly sampled vector z . More precisely, for the presented experiments, f_X and f_Y are sampled at each iteration from the uniform distribution $\mathcal{U}(0.0001, 0.1)$, f_Z from $\mathcal{U}(0.01, 0.1)$ (higher frequencies), wr_{XY} from $\mathcal{U}(0, 10)$, and wr_Z from $\mathcal{U}(0, 0.005)$.

- **Blur (RGB):** To reproduce possible motion blur or unfocused images, Gaussian, uniform or median blur is applied with variable intensity.

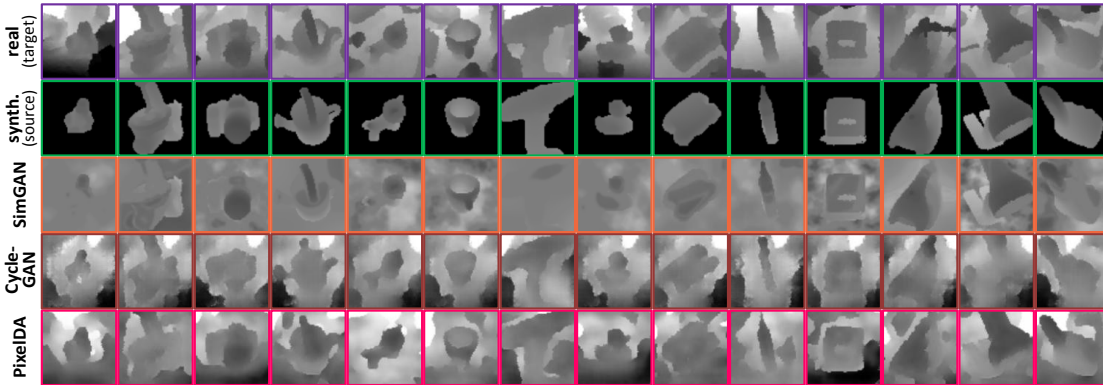


Figure 7.7: Qualitative results of depth domain adaptation GANs [6, 7, 8] on LineMOD [3]. First row contains indicative real images from the target domain; second row contains the synthetic depth images provided as sources; followed by the corresponding GANs outputs below.

7.3 Experimental Evaluation

Pursuing the application to CAD-based recognition tasks, we evaluate our strategy on the tasks of localized object classification and pose estimation. For both presented implementations of our method, i.e., depth- and RGB-based, we quantitatively and extensively evaluate them through a comparison of their performance to state-of-the-art solutions depending on the available training modalities, and through an ablation study.

7.3.1 Experimental Setup

Instance Classification (IC) on T-LESS

As a first task, we consider localized classification on T-LESS [5], a dataset of industrial objects with texture-less CAD models and RGB-D images from different complex scenes. Strong textural and geometrical similarities between the objects and heavy occlusions make it a challenging dataset for geometry-based classification. We select the first 3 scenes and their objects, building a set of 5,514 image patches of objects occluded up to 60%.

Instance Classification and Pose Estimation (ICPE) on LineMOD

LineMOD [3] contains 15 mesh models of distinctive textured objects, along their RGB-D sequences and camera poses. We take advantage of this dataset to demonstrate how texture information is too often taken for granted in CAD-based application, and how its absence can heavily impact usual methods (e.g., in industrial settings). To demonstrate that our method is tailored neither to a dataset nor to a recognition method, we perform an evaluation utilizing the 3D pose estimation and instance classification method from Chapter 3.

Table 7.1: Quantitative results on different tasks and datasets: (A) Classification accuracy of different instances of the IC network over a subset of T-LESS (5 objects); (B) Classification and angular accuracy of different instances of the ICPE method over LineMOD (15 objects). Instances were trained on various data modalities (noiseless synthetic for T^S ; synthetic augmented for T^A ; or real for T^R) and tested on the real datasets X_{test}^R with different pre-processing (none; pre-processing by G_{DEP}^A trained on synthetic augmented data; or by G_{DEP}^R same method trained using real data).
Inference (A) IC on T-LESS (B) ICPE on LineMOD

Input	Method	Classification accuracy	Angular error		Classification accuracy
			Median	Mean	
X_{test}^R	T^S	20.48%	93.48°	100.28°	7.71%
X_{test}^R	T^A	83.35%	13.45°	30.07°	82.14%
$G_{DEP}^A(X_{test}^R)$	T^S	93.01%	13.74°	31.14°	94.77%
X_{test}^R	T^R	95.92%	12.13°	27.80°	95.49%
$G_{DEP}^R(X_{test}^R)$	T^S	96.67%	11.64°	24.31°	98.44%

7.3.2 Real Depth \rightarrow Synthetic Depth

We first demonstrate the effectiveness of our depth pipeline G_{DEP} on a set of different tasks, and show the benefits of decoupling this operation from recognition itself. As a first experiment, we define an instance classification task on the T-LESS patch dataset.

One could argue that directly training T on augmented synthetic data could be more straight-forward than training G_{DEP} against A and plugging it in over T^S afterwards. To prove that our solution not only has the advantage of uncoupling the training of recognition methods to the data augmentation but also improves the end accuracy, we introduce T^A the algorithm trained on augmented data from A . We additionally define method T^R trained on 50% of the real data. During test time, real depth patches are either directly handed to the classifiers, pre-processed by our generator G_{DEP}^A exclusively trained on synthetic data, or pre-processed by a generator G_{DEP}^R trained on real annotated data X_{train}^R . Thus, G_{DEP}^R here serves as a theoretical upper performance bound. Evaluation is done for different combinations of pre-processing and recognition modalities, computing the final classification accuracy for each. Results are shown in Table 7.1-A.

To demonstrate how our method generalizes both to different datasets with diverse classes and environments and to distinctive task-specific applications, we also reproduce the same experimental protocol on a more complex ICPE task. The ICPE task uses the feature-descriptor dataset X_{db}^S to classify instances from LineMOD and estimate their 3D poses. Once again, the networks are either handed unprocessed test data, data pre-processed by G_{DEP}^A , or data pre-processed by G_{DEP}^R . If the class of each returned descriptor agrees with the ground-truth, we compute the angular error between their poses. Once this procedure performed on the entire set X_{test}^R , the classification and angular accuracy are used for the comparison shown Table 7.1-B.

Table 7.2: Comparison to opposite domain-adaptation GANs. Given the two recognition tasks “(A) Instance Classification on T-LESS (5 objects)” and “(B) Instance Classification and Pose Estimation on LineMOD (15 objects)” (defined for the experiment in Table 7.1), we train several modalities of the networks T against diverse domain-adaptation GANs trained on real data X_{train}^R (50% of the datasets), and compare their final accuracy with our results.

		(A) IC on T-LESS		(B) ICPE on LineMOD			
		Classification accuracy		Angular error		Classification accuracy	
		Trained on	Applied to	Median	Mean		
Requiring Real Data	<i>CycleGAN</i>		X_{test}^R	40.97%	71.10°	86.73°	14.72%
	<i>SimGAN</i>		X_{test}^R	59.20%	20.44°	43.36°	73.20%
	<i>PixelDA</i>		X_{test}^R	89.75%	18.15°	39.06°	90.31%
	X^S		$G_{DEP}^R(X_{test}^R)$	96.67%	11.64°	24.31°	98.44%
	X^S		$G_{DEP}^A(X_{test}^R)$	93.01%	13.74°	31.14°	94.77%

For both experiments, we consistently observe the positive impact the pre-processing has on the recognition. The performance of the algorithms using our modality G_{DEP}^A (trained exclusively on synthetic data) matches the results of those plugged to G_{DEP}^R , trained on real data. This demonstrates the effectiveness of our advanced depth data augmentation pipeline. Improvements could be done to match this higher-bound baseline by tailoring the augmentation pipeline to a specific sensor or environment. Our current augmentation pipeline has however the advantage of genericity. While G_{DEP}^R is only trained for the sensor and background type(s) of the provided real dataset, our solution G_{DEP}^A has been trained over A for domain invariance.

We extended this study by also performing ICPE on the real LineMOD scans with their backgrounds perfectly removed. The accuracy using T^S trained on pure synthetic data was only 67.32% for classification, and 24.56° median / 51.58° mean for the pose estimation. This is well below the results on images processed by our method; attesting that our pipeline not only does background subtraction but also effectively uses the CAD prior to recover clean geometry from noisy scans, improving recognition.

Finally, Table 7.1 also reveals the accuracy improvements obtained by decoupling data augmentation and recognition training. Recognition methods T^S trained on “pure” noise-free X^S perform better when used on top of our solution, compared to their respective architectures T^A directly trained on augmented data. The results are even comparable to the respective T^R trained on real images. This confirms our initial intuition regarding the advantages of teaching recognition methods in a noise-free controlled environment, to then map the real data into this known domain. Besides, this decoupling allows once again for greater reusability. Augmentation needs to be done only once to train G_{DEP} , which can then be part of any number of recognition pipelines.

7.3.2.1 Comparison to Usual Domain Adaptation GANs

As previous GAN-based methods to bridge the realism gap are using a subset of real data to learn a mapping from synthetic to realistic, it seems difficult to present a fair

Table 7.3: Quantitative ablation study. Given the two recognition tasks “(A) Instance Classification on T-LESS (5 objects)” and “(B) Instance Classification and Pose Estimation on LineMOD (15 objects)”, we train both networks on noiseless data and evaluate them on the outputs of different modalities of G_{DEP}^A . Each is either trained (A) with different augmentation combinations (BG background noise; FG foreground distortion; OC occlusions; SI sensor simulation); or (B) with different loss combinations ($\mathcal{L}_d + \mathcal{L}_g$ vanilla GAN loss; \mathcal{L}_f foreground-similarity loss; \mathcal{L}_t task-specific loss).

		(A) IC on T-LESS	(B) ICPE on LineMOD		
Modality		Classification accuracy	Angular error		Classification accuracy
			Median	Mean	
(i) Aug.	BG	79.93%	17.64°	38.90°	83.78%
	$BG+FG$	89.42%	15.25°	34.90°	92.33%
	$BG+FG+OC$	93.01%	13.74°	31.14°	94.77%
(ii) Losses	$\mathcal{L}_d + \mathcal{L}_g$	91.09%	14.49°	33.91°	92.89%
	$\mathcal{L}_d + \mathcal{L}_g + \mathcal{L}_f$	92.17%	14.34°	32.21°	93.39%
	$\mathcal{L}_d + \mathcal{L}_g + \mathcal{L}_f + \mathcal{L}_t$	93.01%	13.74°	31.14°	94.77%

comparison to our opposite solution, trained on synthetic data only. We opted for a practical study on the aforementioned tasks, considering the end results given the same recognition methods and test sets. Selecting prominent solutions, SimGAN [6], CycleGAN [7] and PixelDA [66], we trained them on X^S and X_{train}^R (50% of the real datasets) so they learn to generate pseudo-realistic images to train the methods T on. For each task, we measure the accuracy on X_{test}^R , comparing with T^S applied to $G_{DEP}^A(X_{test}^R)$ and $G_{DEP}^R(X_{test}^R)$. The qualitative results are shown in Fig. 7.7.





As SimGAN is designed to refine the pre-existing content of images and not to generate new elements like backgrounds and occlusions, we help this method by filling the images with random background noise beforehand. Still, the refiner does not seem able to deal with the lack of concrete information and fails to converge properly. Unlike the other candidates, CycleGAN neither constrains the original foreground appearance nor tries to regress semantic information to improve the adaptation. Even though the resulting images are filled with some pseudo-realistic clutter, the target objects are often distorted beyond recognition, impairing the task networks training. Finally, PixelDA results look more realistic while preserving most of the semantic information, out of the box. This is achieved by training T^S along its GAN.

These observations are confirmed by the results presented in Table 7.2, which attest of the effectiveness of our reverse processing compared to state-of-the-art methods for these challenging tasks.

7.3.2.2 Ablation of the Solution Components

Performing an ablation study for our depth-based pipeline, we first demonstrate the importance of each component of the depth data augmentation pipeline on the final results. Using the same two tasks (“IC on T-LESS” and “ICPE on LineMOD”), we train

Table 7.4: Quantitative comparison of recognition pipelines, depending on the available training data, for the task of localized instance classification on T-LESS [5] (11 objects).

Training data	Method	Classification accuracy
	T^R	99.34%
	DANN PixelDA	60.58% 63.12%
	Iro et al.	36.03%
	T^A $G_{RGB} \rightarrow T^S$	53.81% 71.78%

three different instances of our method using various degrees of data augmentation. The results are shown in Table 7.3-A, with a steady increase in recognition accuracy for each augmentation component added to the training pipeline.







We set up another experiment to evaluate the influence of the supplementary losses on the end results, as displayed in Table 7.3-B. It shows for instance that the classification error rate on LineMOD drops from 7.11% with the vanilla losses to 5.23% with \mathcal{L}_f and \mathcal{L}_t , i.e., a 26% relative drop which is rather significant, given the upper-bound error rates, cf. Table 7.1. If our augmentation pipeline and vanilla GAN already achieve great results, \mathcal{L}_f and \mathcal{L}_t can further improve them, depending on the sensor type for \mathcal{L}_f or the specific tasks for \mathcal{L}_t , and so covering a wider range of sensors and applications.

Leveraging these components, our depth pipeline not only learns purely from synthetic data how to qualitatively denoise and declutter depth scans, but also considerably improves the performance of recognition methods using it to pre-process their input. We demonstrated how our depth solution makes such algorithms, simply trained on rendered images, almost on a par with the same methods trained in a supervised manner, on images from the real target domain.

7.3.3 Real RGB \rightarrow Synthetic Normals / Depth

In this section, we evaluate the performance our RGB pipeline given the two pre-defined evaluation tasks by comparing it with state-of-the-art methods depending on the available training data (real images, corresponding annotations, CAD models, corresponding realistic textures, or real images from a different domain). For each setup, the same task-specific network is used, trained by itself against our augmentation pipeline A (with texturing augmentation disabled for pre-textured data), or along some auxiliary generators or sub-networks for domain adaptation (e.g., for *PixelDA* [66] or DANN [82]; for T^S used with a pre-trained monocular-RGB-to-depth generator [167]).

Table 7.5: Quantitative comparison of recognition pipelines, depending on the available training data, for the task of localized instance classification and pose estimation (ICPE) on LineMOD [3] with method T from Chapter 3.

Training data	Method	Angular error		Classification accuracy
		Median	Mean	
 RGB ANNOT	T^R	9.50°	12.42°	99.72%
 RGB CAD+TEX	DANN PixelDA	14.33° 15.38°	30.45° 35.17°	89.84% 91.06%
 RGB CAD	DANN PixelDA	43.63° 95.14°	68.59° 97.36°	40.13% 35.39%
 CAD+TEX	T^{S+T} T^{A+T}	88.62° 70.18°	92.35° 84.22°	43.62% 49.11%
 RGB ² CAD	Iro et al.	52.43°	71.69°	41.49%
 CAD	T^A $G_{RGB} \rightarrow T^S$	41.23° 13.37°	67.50° 27.46°	34.38% 91.28%

For both tasks, we consistently observe the positive impact of our RGB pipeline on recognition, as shown in Tables 7.4-7.5. Despite being trained on the scarcest data, with the largest domain gap, our generator G_{RGB} brings the performance of the task-specific methods T^S above other solutions trained on more relevant information. The accuracy improvement is even more apparent for the pose regression task, as our pipeline precisely recovers geometrical features. It also appears clear that decoupling data augmentation and recognition training is beneficial, as illustrated by the accuracy difference between the two last lines of each table. This follows our initial intuition on the logic of teaching task methods in the available clean synthetic domain, while learning in parallel a mapping to project real data into this prior domain. This separation furthermore makes it straightforward to train new task-specific methods, with G_{RGB} ready to be plugged on top.

7.3.3.1 Ablation of the Solution Components

Table 7.6 presents the results of an extensive ablation study done on our RGB multi-modal network architecture G_{RGB} . By consolidating several state-of-the-art works on generative networks [152, 153, 154, 160], we developed a robust architecture to tackle extreme domain mappings (e.g., real RGB to synthetic normals).

We can observe how the addition of decoders for auxiliary tasks improves the final output by synergy. The inclusion of self-attention mechanism (SA layers) in the distillation module further enhances this effect, weighting the contribution of features between intermediary modalities, but also between distant internal regions. Finally, the benefits

Table 7.6: Architectural ablation study, with the “ICPE on LineMOD” task.

Encoder	Decoders				Distill.	Layers		Losses		Angular error		Classification accuracy
	D^N	D^D	D^M	D^L		R	\overrightarrow{SA}	\overrightarrow{skip}	$\mathcal{L}_g^{1..m}$	\mathcal{L}_t	Median	
✓	✓						✓	✓		15.75°	32.80°	87.35%
✓	✓						✓	✓	✓	15.76°	33.76°	88.04%
✓	✓	✓			✓	✓	✓	✓	✓	14.32°	30.31°	89.00%
✓	✓		✓		✓	✓	✓	✓	✓	14.48°	30.71°	89.32%
✓	✓	✓	✓		✓	✓	✓	✓	✓	14.22°	29.26°	89.67%
✓	✓	✓	✓	✓			✓	✓	✓	14.66°	30.83°	88.59%
✓	✓	✓	✓	✓	✓			✓	✓	16.07°	33.22°	87.69%
✓	✓	✓	✓	✓	✓		✓	✓	✓	14.43°	29.56°	90.38%
✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	13.37°	27.46°	91.28%

of passing messages directly between each encoder block and their opposite block for each decoder $D^{1..m}$, through the use of *skip* layers (cf. U-Net architectures [152, 7]), is clearly highlighted in the table, as well as the use of a triplet loss \mathcal{L}_t at the bottleneck to improve the quality of the embedding space.

All in all, our RGB network G_{RGB} relies on a powerful multi-task architecture, structured to tackle real-to-synthetic mapping challenges, by utilizing any available synthetic modalities to learn robust features. One could easily build on this solution by considering additional or more use-case relevant sub-tasks (e.g., contour regression, part segmentation, etc.).

7.4 Conclusion

In this chapter, we presented a novel strategy for complex domain adaptation scenarios. We tackle the domain mapping from the opposite direction, using our custom generator to denoise unseen real samples and retain only the recognition-relevant features available during training. This strategy was used to develop effective solutions for both depth and RGB domain. As a result, without accessing any real data or constraining the recognition methods during training, our solutions outperform other unsupervised and even supervised methods.

8 Network-Driven Domain Randomization

In this chapter, we present an approach to tackle domain adaptation between synthetic and real data. Instead of employing brute-force domain randomization, i.e., augmenting synthetic renderings with random backgrounds or changing illumination and colorization, we leverage the task network as its own adversarial guide toward useful augmentations that maximize the uncertainty of the output. To this end, we design a min-max optimization scheme where a given task competes against a special deception network to minimize the task error subject to the specific constraints enforced by the deceiver. The deception network samples from a family of differentiable pixel-level perturbations and exploits the task architecture to find the most destructive augmentations. Unlike GAN-based approaches that require unlabeled data from the target domain, our method achieves robust mappings that scale well to multiple target distributions from source data alone. We apply our framework to the tasks of digit recognition on enhanced MNIST variants, classification and object pose estimation on the Cropped LineMOD dataset as well as semantic segmentation on the Cityscapes dataset and compare it to a number of domain adaptation approaches, thereby demonstrating similar results with superior generalization capabilities.

8.1 Introduction

We propose a general framework that performs guided randomization with the help of an auxiliary deception network trained in a similar min-max fashion as GAN networks. This is done in two alternating phases, as illustrated in Fig. 8.1. In the first phase, the synthetic input is fed to our deception network responsible for producing augmented images that are then passed to a recognition network to compute the final task-specific loss with provided labels. Then, instead of minimizing the loss, we maximize it via gradient reversal [81] and only back-propagate an update to the deception network parameters. The deception network parameters are steering a set of differentiable modules M_1, \dots, M_N , from which augmentations are sampled. In the next phase, we feed the augmented images to the recognition network together with the original images to minimize the task-specific loss and update the recognition network. In this way, the deception network is encouraged to produce domain randomization by confusing the recognition network and making it resilient to such random changes. By adding different modules and constraints we can influence how much and which parts of the image the deception network alters. In this way, our method outputs images completely independent from

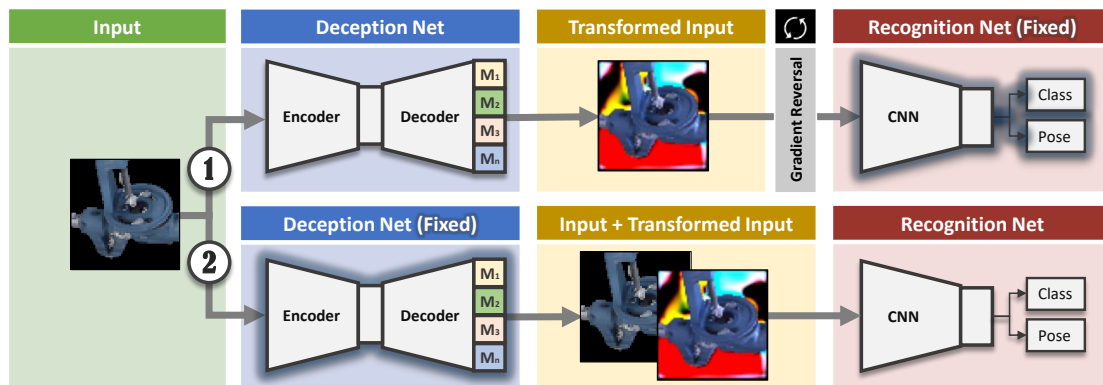


Figure 8.1: Training pipeline. Training is performed in two alternating phases. **Phase 1:** The weights of the deception network are updated, while those of the recognition network are frozen. The recognition network’s objective is maximized instead of being minimized, forcing the deception network to produce increasingly confusing images. **Phase 2:** The generated deceptive images provided by the deception network, whose weights are now frozen, are passed to the recognition network and its weights are updated such that the loss is minimized. As a result of this min-max optimization, the input images are automatically altered by the deception network, forcing the recognition network to be robust to these domain changes.

the target domain and therefore generalizes much better to new unseen domains than related approaches. In summary, our contributions are:

- **DeceptionNet framework** that performs a min-max optimization for guided domain randomization;
- **Various differentiable pixel-level perturbation modules** employed in such a framework suited for synthetic data;
- **Novel sequences:** MNIST-COCO and Extended Cropped LineMOD (Based on HomebrewedDB) that allow to demonstrate our strong generalization capabilities to unseen domains.

In the experimental section we show that steered randomization by leveraging the network structure generalizes much better to new domains than unsupervised approaches with access to the target data while performing comparably well to them on known target domains.

8.2 Methodology

As outlined, our approach towards steered domain randomization is essentially an extension of the task algorithm. Therefore, we have the actual task network $T(\mathbf{x}; \theta_T) \rightarrow \hat{\mathbf{y}}$, which, given an input image \mathbf{x} , returns an estimated label $\hat{\mathbf{y}}$ (e.g., class, pose, segmentation mask, etc.), and (2) a deception network D that takes the source image \mathbf{x}^S and returns the deceptive image \mathbf{x}^D , which, when provided to the task net $T(D(\mathbf{x}^S)) \rightarrow \hat{\mathbf{y}}^D$,

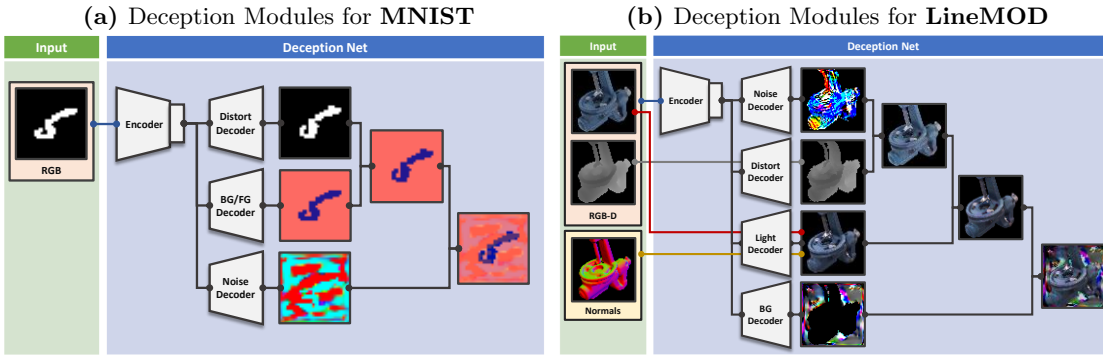


Figure 8.2: Architecture of the deception networks used for the presented experiments. For the case of MNIST classification, three deception modules are used: the *distortion module* applying elastic deformations on the image, the *BG/FG module* responsible for generating background and foreground colors, and the *noise module* additionally distorting the image by applying slight noise. The LineMOD dataset requires a more sophisticated treatment and features four deception modules: noise and distortion (applied on depth channel only), modules similar to the previous case, pixel-wise BG module and light module generating different illumination conditions based on the Phong model.

maximizes the difference between $\hat{\mathbf{y}}^D$ and \mathbf{y}^S . While the recognition network architectures are standard and follow related work [168, 81], we will first focus on our structured deception network, and then describe the optimization objective and the training.

To formalize our pipeline similar to Chapter 7, let $X_c^S := \mathbf{x}_{c,i}^S \forall i \in N_c^S$ be a source dataset composed of N_c^S source images \mathbf{x}_c^S for an object of class \mathbf{c} . Then, $X^S := X_c^S \forall \mathbf{c} \in C$ is the source dataset covering all object classes C . A dataset of real images X^R (not used by us for training) is similarly defined.

8.2.1 Deception Modules

The deception network D follows the encoder-decoder architecture where input \mathbf{x}^S is encoded to a lower-dimensional 2D latent space vector \mathbf{z} and given as an input to multiple decoding modules M_1, \dots, M_n . The final output of D is then a weighted sum of decoded outputs $\mathbf{x}^D := \sum_i \mathbf{w}_i \cdot M_i(\mathbf{z})$ where $\mathbf{w}_i \in [0, 1]^{H \times W}$ act as spatial masking operations. While such a formulation allows for flexibility, the decoders must follow a set of predefined constraints to create meaningful outputs and leverage an inherent image structure instead of finding trivial mappings to decrease the task performance (e.g., by decoding always to 0). Note that our proposed framework is general and, thus, requires instantiations of the deception network for specific datasets. Similar to architecture search, discovering the "best" instantiation is infeasible, but good ones can be found by analyzing the data source. After a reasonable experimentation we settled on certain configurations for MNIST (RGB) and LineMOD (RGB-D), depicted in Fig. 8.2. We

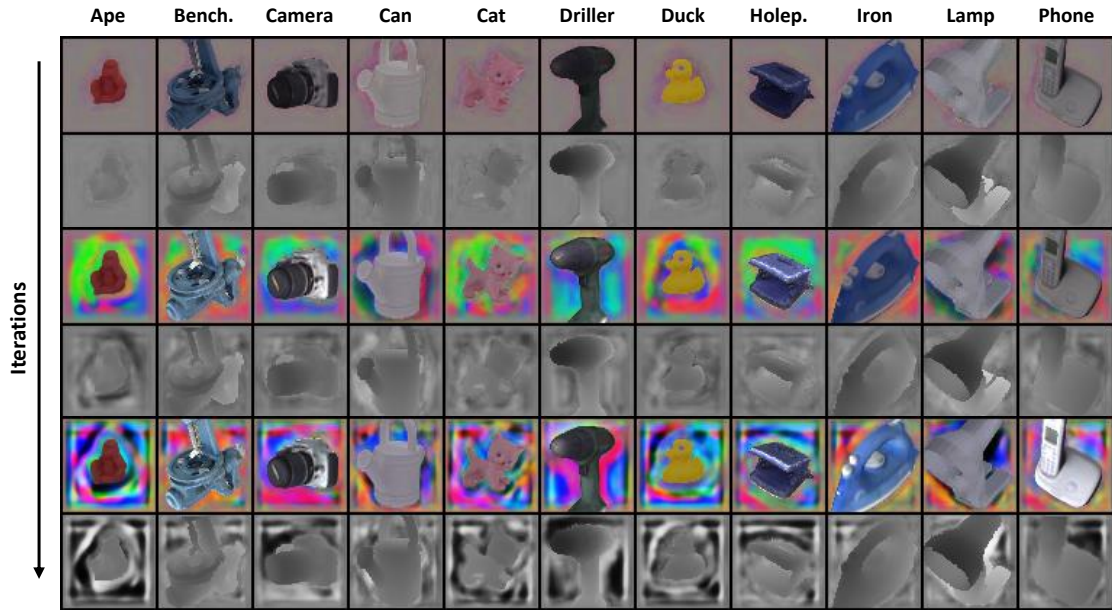


Figure 8.3: Deceptive images x^d over consecutive iterations. The output becomes increasingly more complex for T .

continue by providing more detail on the used decoder modules and their constraint ranges.

8.2.1.1 Background Module (BG)

Since our source images have black backgrounds, they hardly transfer over to the real world with infinite background variety, resulting in a significant accuracy drop. [169, 170] tackle this problem by rendering objects on top of images from large-scale datasets (e.g., MS COCO [165]).

Instead, our background module produces its output by chaining multiple upsampling and convolution operations. While the output is rather simple at start, the module regresses very complex and visually confusing structures in the advanced stages of training.

For MNIST, we used a simpler variant that outputs a single RGB background color $\in [0, 1]$ and an RGB foreground bias $\in [0.1, 0.9]$ (restricted not to intersect with the background color). To form the output, we first apply the background color and then add the foreground bias using the mask. We ensure that the final values are in the range $[0, 1]$.

8.2.1.2 Distortion Module (DS)

The module is based on the idea of the elastic distortions first presented in [166]. Essentially, a 2D deformation field is randomly initialized from $[-1, 1]$ and then convolved with a Gaussian filter of standard deviation σ . For large values of σ , the resulting field

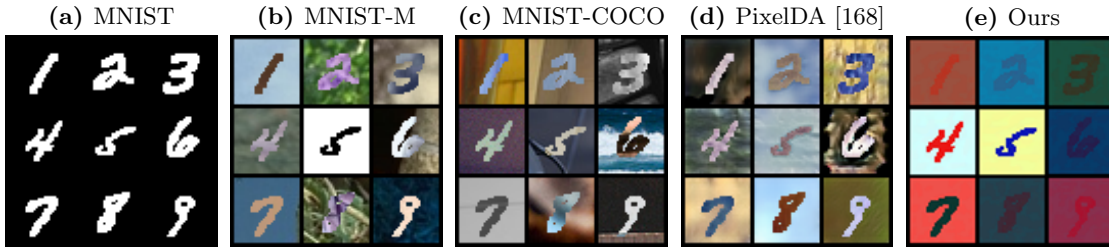


Figure 8.4: Example samples of the MNIST modalities. MNIST (Source), MNIST-M (Target), and MNIST-COCO (Generalization) on the left; and example augmentation images generated by PixelDA and our method respectively.

approaches 0, whereas smaller values of σ keep the field mostly random. However, the moderate values of σ make the resulting field perform elastic deformations, where σ defines the elasticity coefficient. The resulting field is then multiplied by a scaling factor α , which controls the deformation intensity.

Our implementation closely follows the described approach but we use the decoder output as the distortion field and apply resampling, similar to spatial transformer networks [171]. We fix $\sigma = 4$, but learn both $\alpha \in (0, 5]$ and the general decoder parameters. This means that the network itself controls where and how much to deform the object.

8.2.1.3 Noise Module (NS)

Applying slight random noise augmentation to the network input during training is common practice. In a similar fashion, we use the noise decoder to add generated values to the input. The noise decoder regresses a tensor of the input size with values in the range $[-0.01, 0.01]$, which are then added to the input of the module.

8.2.1.4 Light Module (L)

Another feature not well covered by synthetic data is proper illumination. Recent methods [169, 170, 172] prerender a number of synthetic images featuring different light conditions. Here, we instead implement differentiable lighting based on the simple Phong model [173], which is fully operated by the network. While more complex parametric and differentiable illumination models do exist, we found this basic approach to already work quite well.

The module requires surface information which is provided in form of normal maps. From this, we generate three different types of illumination, namely ambient, diffusive, and specular. The light decoder outputs a block of 9 parameters that are used to define the final light properties, i.e., a 3D light direction, an RGB light color (restricted to the range of $[0.8, 1]$), and a weight for each of the three illumination types ($\mathbf{w}_a \in [0.6, 1]$, $\mathbf{w}_d \in [0, 1]$, $\mathbf{w}_s \in [0, 1]$).

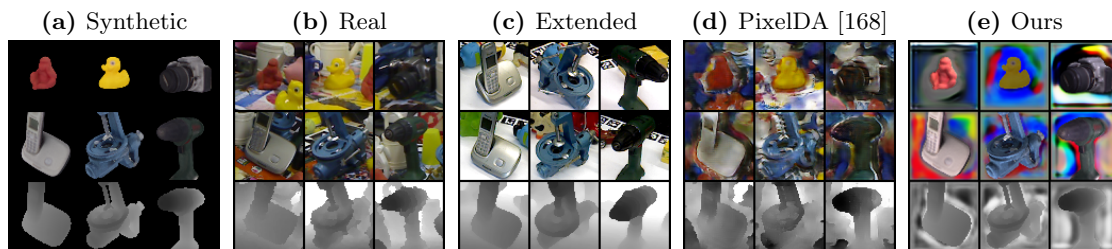


Figure 8.5: Example samples of the LineMOD modalities. Synthetic (Source), Real (Target), and Extended (Generalization) on the left; and example augmentation images generated by PixelDA and our method respectively.

8.2.2 Optimization Objective

The optimization objective of the deception network is essentially the loss of the recognition network; however, instead of minimizing it, we maximize it by updating the parameters in the direction of the positive gradient. This is achieved by adding a gradient reversal layer [81] between the deception and recognition nets as shown in Fig. 8.1. The layer only negates the gradient when back-propagating, thereby resulting in the reversed optimization objective for a given loss. Therefore, the general optimization objective can be written as follows:

$$\min_{\theta_T} \max_{\theta_D} \mathcal{L}_t(T(D(\mathbf{x}; \theta_D)), \mathbf{y}; \theta_T) \quad (8.1)$$

$$\text{subject to} \quad C_{M_n} \quad \text{for } \mathbf{n} = 1, \dots, N_m \quad (8.2)$$

where \mathbf{x} is the input image, \mathbf{y} is the ground truth label, T is the task network, \mathcal{L}_t is the task loss, D is the deception network, and C_m denotes the hard constraints defined by the deception modules enforced by projection after a gradient step. In this framework, the deception network’s objective only depends on the objective of the recognition task and can, therefore, be easily applied to any other task.

8.2.3 Training Procedure

We use two different SGD solvers, where the actual task network has a learning rate of 0.001 with a decaying factor of 0.95 every 20000th iteration. The learning rate of the deception network was found to work well with a constant value of 0.01. We train with a batch size of 64 for all the experiments and we stop training after 500 epochs. During the experimentation, we also discovered that concatenating real and perturbed images led to a consistent improvement in numbers. In Fig. 8.3 we show the evolution of samples over multiple training epochs while the deceptive capabilities steadily increase.

8.3 Experimental Evaluation

In this section, we conduct a series of experiments to compare the capabilities of our pipeline with the state-of-the-art domain adaptation methods. We first compare our-

selves against these baselines for the problem of adaptation and will then compare in terms of generalization. We will conclude with an ablative analysis to measure the impact of each module and modality on the final performance.

As the first dataset, we used the popular handwritten digits dataset MNIST as well as MNIST-M, introduced in [82] for unsupervised domain adaptation (depicted in Figs. 8.4a 8.4b). MNIST-M blends digits from the original monochrome set with random color patches from BSDS500 [174] by simply inverting the color values for the pixels belonging to the digit. The training split containing 59001 target images is then used for domain adaptation. The remaining 9001 target images are used for evaluation. That means that around 86% of the target data is used for training. Note that while MNIST is not technically synthetic, its clean and homogeneous appearance is typical for synthetic data.

The second dataset is the Cropped LineMOD dataset [175] consisting of small centered, cropped 64×64 patches of 11 different objects in cluttered indoor settings displayed in various of poses. It is based on the LineMOD dataset [3] featuring a collection of annotated RGB-D sequences recorded using the Primesense Carmine sensor and associated 3D object reconstructions. The dataset also features a synthetic set of crops of the same objects in various poses on a black background. We will treat this Synthetic Cropped LineMOD as the source dataset and the Real Cropped LineMOD as the target dataset. Domain adaptation methods use a split of 109208 rendered source images and 9673 real-world target images, 1000 real images for validation, and a target domain test set of 2655 images for testing. We show examples in Figs. 8.5a and 8.5b.

The last dataset pair we used for the experiments is SYNTHIA [176] and Cityscapes [177]. SYNTHIA is a collection of pixel-annotated road scene frames rendered from a virtual city. Cityscapes is its real counterpart acquired in the street scenes of 50 different actual cities. Following a common evaluation protocol, we used a subset of 9400 SYNTHIA images, also known as SYNTHIA-RAND-CITYSCAPES, as the source data and 500 Cityscapes validation images as the target data.

8.3.1 Adaptation Tests

All domain adaptation methods use a significant portion of the target data for training, making the resulting mapped source images very similar to the target images (e.g., Fig. 8.4b vs 8.4d and Fig. 8.5b vs 8.5d). A common benchmark for domain adaptation is then to compare the performance of a classifier trained on the mapped data against a classifier trained on the source data only (lower baseline) and against a classifier trained directly on the target data (upper baseline).

Our approach is generally disadvantaged since we can structure our domain mapping only through the source data and the deception architecture. To show that our learned randomization is indeed guided, we additionally implement an unguided randomization variant that applies train time augmentation similar to the related work. It employs the same modules and constraints as our deception network, but its perturbations are conditioned on random values in each forward pass instead of latent codes from the input.

Table 8.1: Baseline tests. While performing slightly worse than the leading state-of-the-art domain adaptation methods using target data, we still manage to achieve very competitive performance without access to target data.

		MNIST \rightarrow MNIST-M	Synthetic Cropped LineMOD \rightarrow Real Cropped LineMOD	
	Model	Classification Accuracy (%)	Classification Accuracy (%)	Mean Angle Error ($^{\circ}$)
	Source (S)	56.6	42.9	73.7
S	Unguided	83.1	53.1	52.6
	Ours	90.4	95.8	51.9
$S + T$	CycleGAN [7]	74.5	68.2	47.5
	MMD [178, 179]	76.9	72.4	70.6
	DANN [82]	77.4	99.9	56.6
	DSN [8]	83.2	100	53.3
	DRIT [180]	91.5	98.1	34.4
	PixelDA [168]	95.9	99.9	23.5
	Target (T)	96.5	100	12.3

8.3.1.1 Classification on MNIST

In Table 8.1 we collect the results of the most relevant methods tested on the MNIST \rightarrow MNIST-M scenario and split them according to the type of data used. Since domain adaptation methods use both source and target data for training, they are allocated to a separate group ($S + T$). Both our method and the unguided randomization variant only have access to the source data and are therefore grouped in S . The task network follows the architecture presented in [81], which is also used by the other methods. The task’s objective \mathcal{L}_t is a simple cross entropy loss between the predicted and the ground truth label distributions.

We can identify three key observations: (1) our method shows very competitive results (90.4% classification) and is on par with the latest domain adaptation pipelines: DSN – 83.2%, DRIT – 91.5% and PixelDA – 95.9%. Moreover, we outperform most of the methods by a significant margin despite the fact that they had access to a large portion of the target data to minimize the domain shift. (2) Guiding the randomization leads to 7% higher accuracy which supports our claim convincingly. (3) Surprisingly, unguided randomization (with appropriate modules) alone is in fact enough to outperform most methods on MNIST.

8.3.1.2 Classification and Pose Estimation on LineMOD

As before, the domain adaptation methods are trained on a mix of source (Synthetic Cropped LineMOD) and target (Real Cropped LineMOD) data and we compare to the predefined baselines. We use the common task network for this benchmark from [81]

Table 8.2: Generalization tests. Our method generalizes well to the extended datasets, while the adaptation methods underperform due to overfitting.

		MNIST \rightarrow MNIST-COCO	Synthetic Cropped LineMOD \rightarrow Extended Real Cropped LineMOD	
Model		Classification Accuracy (%)	Classification Accuracy (%)	Mean Angle Error ($^\circ$)
Source (S)		57.2	63.1	78.3
S	Unguided	85.8	77.2	48.5
	Ours	89.4	99.0	46.5
S+T	DSN [8]	73.2	45.7	76.3
	PixelDA [168]	72.5	76.0	84.2
Target (T)		96.1	100	14.7

and the associated task loss:

$$\mathcal{L}_t(G) = \mathbb{E}_{\mathbf{x}^s, \mathbf{y}^s} \left[-\mathbf{y}^{s\top} \log \hat{\mathbf{y}}^d + \xi \log(1 - \mathbf{q}^{s\top} \hat{\mathbf{q}}^d) \right] \quad (8.3)$$

where the first term is the classification loss and the second term is the log of a quaternion rotation metric [181]. ξ weighs both terms whereas \mathbf{q}^s and $\hat{\mathbf{q}}^d$ are the ground truth and predicted quaternions, respectively.

The results in Table 8.1 present a more nuanced case. On this visually complex dataset, unguided randomization performs only above the lower baseline and is far behind any other method. Our guided randomization, on the other hand, with -95.8% classification and 51.9° angle error is competitive with those of the latest domain adaptation methods using target data: DSN -100% & 53.3° , DRIT -98.1% & 34.4° , and PixelDA -99.9% & 23.5° . Nonetheless, we believe that both DRIT and PixelDA are not fully reachable by target-agnostic methods like ours since the space of all needed adaptations (e.g., aberrations or JPEG artifacts) has to be spanned by our deception modules. The augmentation differences between PixelDA and our method (Figs. 8.5d and 8.5e) suggest the existence of some visual phenomena we are still not accounting for with our deception network.

8.3.2 Generalization Tests

For the second set of experiments, we test the generalization capabilities of our method as well as the competing approaches. The major advantage of our pipeline is its independence from any target domain by design. To support our case we designed two new datasets:

- **MNIST-COCO** The data collection follows the exact same generation procedure of MNIST-M and has the same exact number of images for both training and testing. The only difference here is that instead of the BSDS500 dataset, we use crops from MS COCO. Fig. 8.4e demonstrates some of the newly generated images.

Table 8.3: Module ablation. Evaluation of the importance of the deception network’s modules. BG – background, NS – noise, DS – distortion, L – light.

	MNIST → MNIST-M	Synthetic Cropped LineMOD → Real Cropped LineMOD	
Modules	Classification Accuracy (%)	Classification Accuracy (%)	Mean Angle Error (°)
None	56.6	42.9	73.7
BG	82.4	74.8	50.4
BG + NS	86.5	77.6	52.8
BG + NS + DS	90.4	78.7	48.2
BG + NS + DS + L	-	95.8	51.9

- **Extended Real Cropped LineMOD** Thanks to the help of the authors of the original LineMOD dataset [3], we were able to get some of the original LineMOD objects, namely "phone", "benchvise", and "driller". We used the setup from Chapter 6 and generated an annotated scene for each object. Each scene depicts a specific object placed on a white markerboard atop a turntable and coarsely surrounded by a small number of cluttered objects, slightly occluding the object at times. Each sequence contains 130 RGB-D images covering the full 360° rotation at an elevation angle of approximately 60°. Given the acquired and refined poses, we again crop the images in the same fashion as in the Cropped LineMOD dataset [175]. All 390 images are used for evaluation, with some examples shown in Fig. 8.5c.

For a comparison with the strongest related methods, i.e., DSN, DRIT, and PixelDA, we used open source implementations and diligently ensured that we are able to properly train and reproduce the reported numbers from Table 8.1. While the DRIT implementation worked well for the adaptation experiments, we failed to produce reasonably high numbers for the generalization experiment and chose to exclude it from the comparison.

Similar to before, we train them using the target data from MNIST-M and Real Cropped LineMOD. After the training is finished and the corresponding accuracies on the target test splits are achieved, we test them on the newly acquired dataset. While different, these extended datasets still bear a certain resemblance to the target dataset and we could expect to see a certain amount of generalization. For our randomization methods, we can immediately test on the new data, since retraining is not necessary.

As is evident from Table 8.2, the accuracy of our method on MNIST-COCO is very close to the MNIST-M number (90.4% and 89.4% respectively). For the case of Extended Real Cropped LineMOD, we get even better results than for the Real Cropped LineMOD for both accuracy and angle error: We only need to classify 3 objects instead of 11 with a much smaller pose space, and the scenes are in general cleaner and less occluded. These results underline our claim with respect to generalization. This is, however, not the case for the domain adaptation methods showing drastically worse results. Interestingly, we observe an inverse trend where better results on the original target data lead to a more

Table 8.4: Input modality ablation. Performance evaluation based on the input data type used: depth, RGB, or RGB-D.

	Synthetic Cropped LineMOD \rightarrow Real Cropped LineMOD		Synthetic Cropped LineMOD \rightarrow Extended Real Cropped LineMOD	
Input	Classification Accuracy (%)	Mean Angle Error ($^{\circ}$)	Classification Accuracy (%)	Mean Angle Error ($^{\circ}$)
D	73.3	36.6	78.7	34.9
RGB	84.8	57.4	85.9	49.4
RGB-D	95.8	51.9	99.0	46.5

significant drop. Despite of having a very high accuracy on the target data and the ability to generate additional samples that do not exist in the dataset, these methods present typical signs of overfit mappings that cannot generalize well to the extensions of the same data acquired in a similar manner. The simple reason for this might be the nature of these methods: they do not generalize to the features that matter the most for the recognition task, but to simply replicate the target distribution as close as possible. As a result, it is not clear what the classifier exactly focuses on during inference; however, it could very likely be the particular type of images (e.g., in case of MNIST-COCO) or a specific type of backgrounds and illumination (e.g., in case of Extended Real Cropped LineMOD). In contrast to domain adaptation methods, our pipeline is designed not to replicate the target distribution, but to make the classifier invariant to the changes that should not affect classification, which is the reason why our results remain stable.

8.3.3 Ablation Studies

In this section, we perform a set of ablation studies to gain more insight into the impact of each module inside the deception network. Obviously, our modules model only a fraction of possible perturbations and it is important to understand the individual contributions. Moreover, we demonstrate how well we perform provided different types of input modalities for the LineMOD datasets.

8.3.3.1 Deception Modules

We tested four different variations of the deception net that use varying combinations of the deception modules: *background (BG)*, *noise (NS)*, *distortion (DS)*, and *light (L)*. The exact combinations and the results on both datasets are listed in Table 8.3.

It can be clearly seen that each additional module in the deception network adds to the discriminative power of the final task network. The most important modules can also be easily distinguished based on the results. Apparently, the background module always makes a significant difference: the purely black backgrounds of the source data are drastically different from the real imagery. Another interesting observation is the strong impact the lighting perturbation has in the case of the Cropped LineMOD dataset. This enforces the notion that real sequences undergo many kinds of lighting changes that are

Table 8.5: Real-world application. Segmentation performance on SYNTHIA \rightarrow Cityscapes benchmark based on Intersection over Union (IoU) tested on 16 (**mIoU**) and 13 (**mIoU***) classes of the Cityscapes dataset. Our method outperforms *source* and *unguided* by a significant margin and remains competitive to the methods relying on the target data.

		Road	SW	BLDG	Wall	Fence	Pole	TL	TS	VEG	Sky	PRSN	Rider	Car	Bus	Mbike	Bike	mIoU	mIoU*
	Source (S)	3.8	10.2	46.3	1.8	0.3	19.1	4.0	7.5	71.8	72.2	44.6	3.4	24.9	5.2	0.0	2.5	19.8	22.8
S	Unguided	17.9	8.8	59.2	0.8	0.4	22.1	3.5	6.1	71.4	70.4	40.3	7.3	37.9	3.3	0.2	7.3	22.3	25.7
	Ours	51.4	17.8	62.5	1.6	0.4	22.6	6.0	11.9	70.9	73.5	42.1	8.2	40.9	8.1	3.9	18.4	27.5	32.0
S + T	FCNs Wld [182]	11.5	19.6	30.8	4.4	0.0	20.3	0.1	11.7	42.3	68.7	51.2	3.8	54.0	3.2	0.2	0.6	20.1	22.9
	CDA [183]	65.2	26.1	74.9	0.1	0.5	10.7	3.7	3.0	76.1	70.6	47.1	8.2	43.2	20.7	0.7	13.1	29.0	34.8
	Cross-City [184]	62.7	25.6	78.3	-	-	-	1.2	5.4	81.3	81.0	37.4	6.4	63.5	16.1	1.2	4.6	-	35.7
	Tsai et al. [185]	78.9	29.2	75.5	-	-	-	0.1	4.8	72.6	76.7	43.4	8.8	71.1	16.0	3.6	8.4	-	37.6
	ROAD-Net [186]	77.7	30.0	77.5	9.6	0.3	25.8	10.3	15.6	77.6	79.8	44.5	16.6	67.8	14.5	7.0	23.8	36.1	41.7
	LSD-seg [187]	80.1	29.1	77.5	2.8	0.4	26.8	11.1	18.0	78.1	76.7	48.2	15.2	70.5	17.4	8.7	16.7	36.1	42.1
	Chen et al. [188]	78.3	29.2	76.9	11.4	0.3	26.5	10.8	17.2	81.7	81.9	45.8	15.4	68.0	15.9	7.5	30.4	37.3	43.0
	Target (T)	96.5	74.6	86.1	37.1	33.2	30.2	39.7	51.6	87.3	90.4	60.1	31.7	88.4	52.3	33.6	59.1	59.5	65.5

not well-represented by synthetic renderings without any additional relighting. Note that the MNIST deception network does not employ lighting.

8.3.3.2 Input Modalities

For the task of simultaneous instance classification and pose estimation, we (as well as the other methods) always use the full RGB-D information. This ablation aims to show how well we fare provided only a certain type of data and the impact on the final results. Table 8.4 shows that RGB allows for better classification, whereas depth provides better pose estimates. We can further boost the classification enormously and reduce the pose error by combining both modalities.

8.3.4 Real-World Scenario

We demonstrate a real-world application of our approach on a more practical problem of semantic segmentation using the common SYNTHIA \rightarrow Cityscapes benchmark. Having only synthetic SYNTHIA renderings, we try to generalize to the real Cityscapes data by evaluating our method on 13 and 16 classes using the Intersection over Union (IoU) metric. This setup is particularly difficult since the domain gap problem here is intensified by a completely different set of segmentation instances and camera views. For a fair comparison, all methods use a VGG-16 base (FCN-8s) recognition network. The deception modules used in this case are as follows: 2D noise (NS), elastic distortion (DS), and light (L). Normal maps for the light module are generated from the available synthetic depth data. The sample outputs from each of the above-mentioned modules are shown in Fig. 8.6.

Table 8.5 shows that even without access to target domain data, our pipeline remains competitive with the methods relying on target data, showing mIoU of 27.5% and mIoU* of 32% (16 and 13 classes) – well above *source* and *unguided*. The results also confirm the generality of the approach with respect to the different task architectures and datasets.

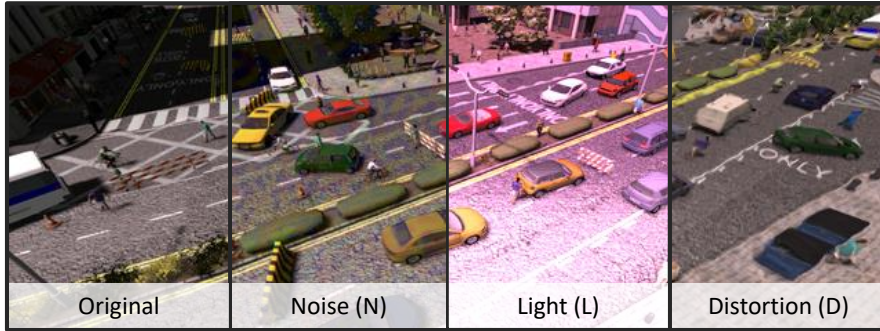


Figure 8.6: Deceptive augmentations. Augmentations applied for the SYNTHIA \rightarrow Cityscapes scenario.

8.4 Conclusion

In this chapter, we presented a new framework to tackle the domain gap problem when no target data is available. Using a task network and its objective, we show how to extend it with a simple encoder-decoder deception network and bind both in a min-max game in order to achieve guided domain randomization. As a result, we obtain increasingly more robust task networks. We demonstrate a comparable performance to domain adaptation methods on two datasets and, most importantly, show superior generalization capabilities where the domain adaptation methods tend to drop in performance due to overfitting to the target distribution. Our results suggest that guided randomization, because of its simple but effective nature, should become a standard procedure to define baselines for domain transfer and adaptation techniques.

9 Conclusion and Outlook

In this final chapter, we present a summary of the presented methods, analyze their advantages and limitations, and propose new directions for future research.

9.1 Summary

In this thesis, we covered the problems of 3D object pose estimation and domain adaptation and presented a number of industry-relevant solutions. First, multiple pose estimation methods of various complexity were proposed. We started from a 3D pose estimation pipeline based on manifold learning and nearest-neighbor search. To efficiently improve its the scalability properties and the robustness of the feature space, we introduced the loss with dynamic margin and made the pipeline end-to-end by combining it with direct pose regression. Second, we proposed a dense correspondence-based 6D pose estimation solution that works in RGB and can be trained entirely on synthetic data while showing state-of-the-art results. Moreover, this method is robust to occlusions and clutter and can run real-time on a mobile device. Next, we used this pose estimation solution to build a full 9D pose estimation pipeline that not only estimates the full 6D object’s pose but also its shape and scale. To make this possible, it uses a differentiable SDF shape database and a novel differentiable SDF renderer. We show that this approach can recover a substantial amount of cuboid labels with high precision, and that these labels can be used to train downstream 3D object detectors with results close to the state of the art.

In the second part of the thesis we proposed two solutions to tackle the problem of the domain gap between synthetic and real data. The first solution presents a reverse domain adaptation pipeline, where instead of making synthetic images look more realistic, it aims to map real images to the synthetic domain. This helps to completely decouple the domain adaptation network from the downstream solver, which subsequently improves the end downstream performance. Moreover, while this solution does not utilize any real data, it outperforms general GAN solutions using real data on the same tasks. Lastly, we proposed an adversarial domain randomization pipeline. As opposed to standard domain randomization, our method is driven by the downstream task. As a result, it only introduces meaningful augmentations that are most confusing for the downstream network. After this extended training procedure, the downstream network becomes much more robust to domain changes, outperforming standard domain randomization and being on par with the domain adaptation pipelines using real data.

9.2 Limitations and Future Work

The topic of pose estimation leaves many challenges for further explorations. While the proposed manifold learning pose estimation method scales well with respect to the number of objects, it does not provide the pose accuracy to compete with the correspondence-based methods. This is a common problem for template-based methods and to get feasible estimates, the usual solution is to use refinement as a post-processing step [55, 189]. However, doing so sufficiently harms the overall runtime making it difficult to deploy in real time. On the other hand, the presented correspondence-based methods provide very precise estimates, are robust to occlusions and clutter, and are real-time-capable, but when one network has to estimate correspondences for multiple objects, the accuracy drops dramatically as shown in Chapter 6. Moreover, there is no explicit treatment of symmetric objects, resulting in ambiguous and noisy estimates when a need for estimating poses of such objects appears. A method combining the strengths of template- and correspondence-based approaches would be extremely useful for industrial applications.

Our autolabeling pipeline allows to reliably estimate 9D bounding boxes and car shapes. However, it works on local 2D detections and does not use any global information about the other detected objects in the image. Such information would potentially allow to resolve ambiguous cases by global optimization, e.g., when only a small portion of a car is visible and one cannot reliably estimate its pose. Another interesting directions are the multi-view and tracking scenarios. Having multiple images and relations between them provides much more information compared to a monocular case and would allow to better deal with occlusions and recover the metric scale. Yet another direction could be an extension of the current differentiable DeepSDF database to non-rigid objects. While the current implementation works for any kind of rigid objects, it does not simply extend to objects like people and animals. To solve it one could try to encode all possible kinematic shape variations into DeepSDF, however it seems more reasonable to either integrate the support for articulated objects [190] or change the underlying shape representation, e.g., adopt the SMPL model [191] for people or SMAL [192] for animals.

The topic of the realism gap for CNN-based methods is underrepresented in the literature and many problems still remain unresolved. The majority of full-scale 6D pose estimation methods either train directly on real data or use networks pretrained on real data and freeze the layers to adapt to another domain and prevent overfitting. While the presented domain adaptation solutions show impressive performance when applied to localized image patches, they do not simply extend to full-sized images. Moreover, both pipelines require carefully designed augmentation modules to cover possible visual variations of the object. Unfortunately, one cannot guarantee that all possible combinations are indeed covered and many existing ones actually might never appear in reality directly affecting the performance. One of the promising directions aiming to solve this problem lies in using realistic (differentiable) rendering and physical engines to significantly improve the quality of the simulated data and decrease the realism gap [193, 4].

A Authored and Co-Authored Publications

Authored:

1. **S. Zakharov**, W. Kehl, B. Planche, A. Hutter, S. Ilic. *3D Object Instance Recognition and Pose Estimation Using Triplet Loss with Dynamic Margin*. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2017
2. **S. Zakharov***, B. Planche*, Z. Wu, A. Hutter, H. Kosch, S. Ilic. *Keep it Unreal: Bridging the Realism Gap for 2.5D Recognition with Geometry Priors Only*. International Conference on 3DVision (3DV), 2018 (*equal contribution)
3. B. Planche*, **S. Zakharov***, Z. Wu, A. Hutter, H. Kosch, S. Ilic. *Seeing Beyond Appearance - Mapping Real Images into Geometrical Domains for Unsupervised CAD-based Recognition*. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2019 (*equal contribution)
4. **S. Zakharov**, W. Kehl, S. Ilic. *DeceptionNet: Network-Driven Domain Randomization*. IEEE International Conference on Computer Vision (ICCV), 2019
5. **S. Zakharov***, I. Shugurov*, S. Ilic. *DPOD: 6D Pose Object Detector and Refiner*. IEEE International Conference on Computer Vision (ICCV), 2019 (*equal contribution)
6. **S. Zakharov***, W. Kehl*, A. Bhargava, A. Gaidon. *Autolabeling 3D Objects with Differentiable Rendering of SDF Shape Priors*. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2020 (*equal contribution)

Co-Authored:

1. B. Planche, Z. Wu, K. Ma, S. Sun, S. Kluckner, T. Chen, A. Hutter, **S. Zakharov**, H. Kosch, J. Ernst *DepthSynth: Real-Time Realistic Synthetic Data Generation from CAD Models for 2.5D Recognition*. International Conference on 3D Vision (3DV), 2017
2. M. Bui, **S. Zakharov**, S. Albarqouni, S. Ilic, N. Navab. *When Regression meets Manifold Learning for Object Recognition and Pose Estimation*. IEEE International Conference on Robotics and Automation (ICRA), 2018
3. R. Kaskman, **S. Zakharov**, I. Shugurov, S. Ilic. *HomebrewedDB: RGB-D Dataset for 6D Pose Estimation of 3D Objects*. IEEE International Conference on Computer Vision (ICCV) Workshops, 2019

Bibliography

- [1] R. Hartley and A. Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.
- [2] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9, 2008.
- [3] S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige, and N. Navab. Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes. In *Asian conference on computer vision (ACCV)*, 2012.
- [4] Parallel domain: Data generation for autonomy. <https://www.paralldomain.com/>.
- [5] T. Hodan, P. Haluza, Š. Obdržálek, J. Matas, M. Lourakis, and X. Zabulis. T-less: An rgb-d dataset for 6d pose estimation of texture-less objects. In *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2017.
- [6] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb. Learning from simulated and unsupervised images through adversarial training. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.
- [7] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.
- [8] K. Bousmalis, G. Trigeorgis, N. Silberman, D. Krishnan, and D. Erhan. Domain separation networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.
- [9] Y. Li, G. Wang, X. Ji, Y. Xiang, and D. Fox. Deepim: Deep iterative matching for 6d pose estimation. In *European Conference on Computer Vision (ECCV)*, 2018.
- [10] I. Goodfellow, Y. Bengio, and A. Courville. Deep learning. Book in preparation for MIT Press, 2016. URL: <http://www.deeplearningbook.org>.
- [11] M. A. Nielsen. Neural networks and deep learning. URL: <http://neuralnetworksanddeeplearning.com>, 2015.
- [12] A. Karpathy. Convolutional neural networks for visual recognition, 2015.

BIBLIOGRAPHY

- [13] S. C. Welch. Neural networks demystified, part 4: Backpropagation. URL: <https://www.youtube.com/watch?v=G1cnxUlrtek>.
- [14] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv*, 2016.
- [15] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient backprop. In *Neural networks: Tricks of the trade*. Springer, 2012.
- [16] B. Blaus. Anatomy of a multipolar neuron, 2013. URL: <https://commons.wikimedia.org/w/index.php?curid=28761830>.
- [17] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, 2015.
- [18] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence (TPAMI)*, 22(11), 2000.
- [19] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1. IEEE, 2005.
- [20] S. Hinterstoisser, S. Holzer, C. Cagniart, S. Ilic, K. Konolige, N. Navab, and V. Lepetit. Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes. In *International conference on computer vision (ICCV)*. IEEE, 2011.
- [21] S. Hinterstoisser, C. Cagniart, S. Ilic, P. Sturm, N. Navab, P. Fua, and V. Lepetit. Gradient response maps for real-time detection of textureless objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 34(5), 2012.
- [22] Y. Konishi, Y. Hanzawa, M. Kawade, and M. Hashimoto. Fast 6d pose estimation from a monocular image using hierarchical pose trees. In *European Conference on Computer Vision (ECCV)*. Springer, 2016.
- [23] R. Rios-Cabrera and T. Tuytelaars. Discriminatively trained templates for 3d object detection: A real time scalable approach. In *IEEE International Conference on Computer Vision (ICCV)*, 2013.
- [24] W. Kehl, F. Tombari, N. Navab, S. Ilic, and V. Lepetit. Hashmod: A hashing method for scalable 3d object detection. In *British Machine Vision Conference (BMVC)*, volume 1, 2015.
- [25] T. Hodaň, X. Zabulis, M. Lourakis, Š. Obdržálek, and J. Matas. Detection and fine 3d pose estimation of texture-less objects in rgb-d images. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015.

- [26] R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality reduction by learning an invariant mapping. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2. IEEE, 2006.
- [27] H. Guo, J. Wang, Y. Gao, J. Li, and H. Lu. Multi-view 3d object retrieval with deep embedding network. *IEEE Transactions on Image Processing*, 25(12), 2016.
- [28] E. Hoffer and N. Ailon. Deep metric learning using triplet network. In *International Workshop on Similarity-Based Pattern Recognition*. Springer, 2015.
- [29] P. Wohlhart and V. Lepetit. Learning descriptors for object recognition and 3d pose estimation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.
- [30] D. G. Lowe. Local feature view clustering for 3d object recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1. IEEE, 2001.
- [31] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. In *European conference on computer vision (ECCV)*. Springer, 2006.
- [32] S. Holzer, J. Shotton, and P. Kohli. Learning to efficiently detect repeatable interest points in depth data. In *European Conference on Computer Vision (ECCV)*. Springer, 2012.
- [33] E. Rosten, R. Porter, and T. Drummond. Faster and better: A machine learning approach to corner detection. *IEEE transactions on pattern analysis and machine intelligence (TPAMI)*, 32(1), 2008.
- [34] M. Ozuysal, M. Calonder, V. Lepetit, and P. Fua. Fast keypoint recognition using random ferns. *IEEE transactions on pattern analysis and machine intelligence (TPAMI)*, 32(3), 2009.
- [35] V. Lepetit and P. Fua. Keypoint recognition using randomized trees. *IEEE transactions on pattern analysis and machine intelligence (TPAMI)*, 28(9), 2006.
- [36] K. M. Yi, E. Trulls, V. Lepetit, and P. Fua. Lift: Learned invariant feature transform. In *European Conference on Computer Vision (ECCV)*, 2016.
- [37] A. Crivellaro, M. Rad, Y. Verdie, K. Moo Yi, P. Fua, and V. Lepetit. A novel representation of parts for accurate 3d object detection and tracking in monocular images. In *IEEE international conference on computer vision (ICCV)*, 2015.
- [38] M. Rad and V. Lepetit. Bb8: A scalable, accurate, robust to partial occlusion method for predicting the 3d poses of challenging objects without using depth. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.

BIBLIOGRAPHY

- [39] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *IEEE conference on computer vision and pattern recognition (CVPR)*, 2016.
- [40] B. Tekin, S. N. Sinha, and P. Fua. Real-time seamless single shot 6d object pose prediction. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.
- [41] J. Shotton, B. Glocker, C. Zach, S. Izadi, A. Criminisi, and A. Fitzgibbon. Scene coordinate regression forests for camera relocalization in rgb-d images. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013.
- [42] A. Guzman-Rivera, P. Kohli, B. Glocker, J. Shotton, T. Sharp, A. Fitzgibbon, and S. Izadi. Multi-output learning for camera relocalization. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.
- [43] J. Valentin, M. Nießner, J. Shotton, A. Fitzgibbon, S. Izadi, and P. H. Torr. Exploiting uncertainty in regression forests for accurate camera relocalization. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.
- [44] J. Taylor, J. Shotton, T. Sharp, and A. Fitzgibbon. The vitruvian manifold: Inferring dense correspondences for one-shot human pose estimation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2012.
- [45] E. Brachmann, F. Michel, A. Krull, M. Ying Yang, S. Gumhold, et al. Uncertainty-driven 6d pose estimation of objects and scenes from a single rgb image. In *IEEE conference on computer vision and pattern recognition (CVPR)*, 2016.
- [46] R. A. Güler, N. Neverova, and I. Kokkinos. Densepose: Dense human pose estimation in the wild. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.
- [47] A. Kendall, M. Grimes, and R. Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *IEEE International Conference on Computer Vision (ICCV)*, 2015.
- [48] F. Walch, C. Hazirbas, L. Leal-Taixe, T. Sattler, S. Hilsenbeck, and D. Cremers. Image-based localization using lstms for structured feature correlation. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.
- [49] A. Kendall and R. Cipolla. Geometric loss functions for camera pose regression with deep learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.
- [50] W. Kehl, F. Manhardt, F. Tombari, S. Ilic, and N. Navab. Ssd-6d: Making rgb-based 3d detection and 6d pose estimation great again. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.

- [51] P. Besl and D. McKay. Method for registration of 3-d shapes. In *Robotics-DL tentative*. International Society for Optics and Photonics, 1992.
- [52] Y. Chen and G. G. Medioni. Object modeling by registration of multiple range images. *Image Vis. Comput.*, 10(3), 1992.
- [53] C. Harris and C. Stennett. Rapid-a video rate object tracker. In *British Machine Vision Conference (BMVC)*, 1990.
- [54] T. Drummond and R. Cipolla. Real-time visual tracking of complex structures. *IEEE Transactions on pattern analysis and machine intelligence (TPAMI)*, 24(7), 2002.
- [55] W. Kehl, F. Manhardt, F. Tombari, S. Ilic, and N. Navab. Ssd-6d: Making rgb-based 3d detection and 6d pose estimation great again. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.
- [56] F. Manhardt, W. Kehl, N. Navab, and F. Tombari. Deep model-based 6d pose refinement in rgb. In *European Conference on Computer Vision (ECCV)*, 2018.
- [57] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*, 2017.
- [58] S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige, and N. Navab. Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes. In *Asian conference on computer vision (ACCV)*. Springer, 2012.
- [59] M. M. Loper and M. J. Black. OpenDR: An approximate differentiable renderer. In *European conference on computer vision (ECCV)*, 2014.
- [60] Y. U. Hiroharu Kato and T. Harada. Neural 3d mesh renderer. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.
- [61] S. Liu, T. Li, W. Chen, and H. Li. Soft rasterizer: A differentiable renderer for image-based 3d reasoning. In *IEEE International Conference on Computer Vision (ICCV)*, 2019.
- [62] T.-M. Li, M. Aittala, F. Durand, and J. Lehtinen. Differentiable monte carlo ray tracing through edge sampling. In *Conference and Exhibition on Computer Graphics & Interactive Techniques in Asia (SIGGRAPH Asia)*, 2018.
- [63] M. J. Landau, B. Y. Choo, and P. A. Beling. Simulating kinect infrared and depth images. *IEEE transactions on cybernetics*, 46(12), 2015.
- [64] B. Planche, Z. Wu, K. Ma, S. Sun, S. Kluckner, T. Chen, A. Hutter, S. Zakharov, H. Kosch, and J. Ernst. Depthsynth: Real-time realistic synthetic data generation

BIBLIOGRAPHY

- from cad models for 2.5 d recognition. In *International Conference on 3D Vision (3DV)*, 2017.
- [65] Y. Taigman, A. Polyak, and L. Wolf. Unsupervised cross-domain image generation. *International Conference on Learning Representations (ICLR)*, 2017.
- [66] K. Bousmalis, N. Silberman, D. Dohan, D. Erhan, and D. Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks. In *IEEE conference on computer vision and pattern recognition (CVPR)*, 2017.
- [67] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.
- [68] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2014.
- [69] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv*, 2015.
- [70] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.
- [71] C. Li and M. Wand. Precomputed real-time texture synthesis with markovian generative adversarial networks. In *European conference on computer vision (ECCV)*. Springer, 2016.
- [72] L. Yu, W. Zhang, J. Wang, and Y. Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *AAAI conference on artificial intelligence*, 2017.
- [73] X. Chen, Y. Duan, R. Houthoofd, J. Schulman, I. Sutskever, and P. Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.
- [74] P. Luc, C. Couprie, S. Chintala, and J. Verbeek. Semantic segmentation using adversarial networks. In *NeurIPS Workshop on Adversarial Training*, 2016.
- [75] D. Nie, R. Trullo, J. Lian, C. Petitjean, S. Ruan, Q. Wang, and D. Shen. Medical image synthesis with context-aware generative adversarial networks. In *International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*. Springer, 2017.
- [76] Y. Xue, T. Xu, H. Zhang, L. R. Long, and X. Huang. Segan: Adversarial network with multi-scale l1 loss for medical image segmentation. *Neuroinformatics*, 16(3-4), 2018.

- [77] J. Gauthier. Conditional generative adversarial nets for convolutional face generation. *Class Project for Stanford CS231N: Convolutional Neural Networks for Visual Recognition, Winter semester*, 2014(5), 2014.
- [78] E. L. Denton, S. Chintala, R. Fergus, et al. Deep generative image models using a laplacian pyramid of adversarial networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.
- [79] M.-Y. Liu and O. Tuzel. Coupled generative adversarial networks. In *Advances in neural information processing systems (NeurIPS)*, 2016.
- [80] L. Chongxuan, T. Xu, J. Zhu, and B. Zhang. Triple generative adversarial nets. In *Advances in neural information processing systems (NeurIPS)*, 2017.
- [81] Y. Ganin and V. Lempitsky. Unsupervised domain adaptation by backpropagation. In *International Conference on Machine Learning (ICML)*, 2015.
- [82] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky. Domain-adversarial training of neural networks. *Journal of Machine Learning Research (JMLR)*, 2016.
- [83] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell. Adversarial discriminative domain adaptation. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.
- [84] M. Rad, M. Oberweger, and V. Lepetit. Feature mapping for learning fast and accurate 3d pose inference from synthetic images. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.
- [85] M. Rad, M. Oberweger, and V. Lepetit. Domain transfer for 3d pose estimation from color images without manual annotations. In *Asian Conference on Computer Vision*. Springer, 2018.
- [86] F. Sadeghi and S. Levine. (cad)2rl: Real single-image flight without a single real image. *Robotics: Science and Systems(RSS)*, 2017.
- [87] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017.
- [88] K. Perlin. Noise hardware. *Real-Time Shading SIGGRAPH Course Notes*, 2001.
- [89] H. Su, C. R. Qi, Y. Li, and L. J. Guibas. Render for cnn: Viewpoint estimation in images using cnns trained with rendered 3d model views. In *IEEE International Conference on Computer Vision (ICCV)*, 2015.

BIBLIOGRAPHY

- [90] M. Bui, S. Albarqouni, M. Schrapp, N. Navab, and S. Ilic. X-ray posenet: 6 dof pose estimation for mobile x-ray devices. In *IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2017.
- [91] R. A. Güler, N. Neverova, and I. Kokkinos. Densepose: Dense human pose estimation in the wild. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.
- [92]
- [93] S. Peng, Y. Liu, Q. Huang, X. Zhou, and H. Bao. Pvnet: Pixel-wise voting network for 6dof pose estimation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.
- [94] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision (ECCV)*. Springer, 2014.
- [95] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE conference on computer vision and pattern recognition (CVPR)*, 2016.
- [96] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence (TPAMI)*, 22, 2000.
- [97] S. Hinterstoisser, V. Lepetit, P. Wohlhart, and K. Konolige. On pre-trained image features and synthetic images for deep learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.
- [98] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *European conference on computer vision (ECCV)*. Springer, 2016.
- [99] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Ieee, 2009.
- [100] E. Brachmann, A. Krull, F. Michel, S. Gumhold, J. Shotton, and C. Rother. Learning 6d object pose estimation using 3d object coordinates. In *European conference on computer vision*. Springer, 2014.
- [101] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. *Robotics: Science and Systems (RSS)*, 2018.
- [102] M. Oberweger, M. Rad, and V. Lepetit. Making deep heatmaps robust to partial occlusions for 3d object pose estimation. In *European Conference on Computer Vision (ECCV)*, 2018.

- [103] E. Brachmann, F. Michel, A. Krull, M. Ying Yang, S. Gumhold, et al. Uncertainty-driven 6d pose estimation of objects and scenes from a single rgb image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.
- [104] J. Lee, S. Walsh, A. Harakeh, and S. Waslander. Leveraging pre-trained 3d object detection models for fast ground truth generation. In *International Conference on Intelligent Transportation Systems (ITSC)*, 2018.
- [105] C. Huang. Adding a dimension: Annotating 3d objects with 2d data. 2018. <https://scale.com/blog/3d-cuboids-annotations>.
- [106] Z. Wang, H. Ling, D. Acuna, A. Kar, and S. Fidler. Object instance annotation with deep extreme level set evolution. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.
- [107] H. Ling, J. Gao, A. Kar, W. Chen, and S. Fidler. Fast interactive object annotation with curve-gen. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.
- [108] W. Chen, J. Gao, H. Ling, E. J. Smith, J. Lehtinen, A. Jacobson, and S. Fidler. Learning to predict 3d objects with an interpolation-based differentiable renderer. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.
- [109] A. Yuille and D. Kersten. Vision as bayesian inference: analysis by synthesis? *Trends in cognitive sciences*, 10(7), 2006.
- [110] A. Kundu, Y. Li, and J. M. Rehg. 3d-rcnn: Instance-level 3d object reconstruction via render-and-compare. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.
- [111] H. Wang, S. Sridhar, J. Huang, J. Valentin, S. Song, and L. J. Guibas. Normalized object coordinate space for category-level 6d object pose and size estimation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.
- [112] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove. DeepSDF: Learning continuous signed distance functions for shape representation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.
- [113] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.
- [114] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. CARLA: An open urban driving simulator. In *Conference on Robot Learning (CoRL)*, 2017.
- [115] A. Gaidon, Q. Wang, Y. Cabon, and E. Vig. Virtual worlds as proxy for multi-object tracking analysis. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

BIBLIOGRAPHY

- [116] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *Conference on Computer graphics and interactive techniques (SIGGRAPH)*, 1996.
- [117] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Conference on Computer graphics and interactive techniques (SIGGRAPH)*, 1987.
- [118] H. Pfister, M. Zwicker, J. van Baar, and M. Gross. Surfels: Surface elements as rendering primitives. In *Conference on Computer graphics and interactive techniques (SIGGRAPH)*, 2000.
- [119] P. Schönemann. A generalized solution of the orthogonal procrustes problem. In *Psychometrika*, 1966.
- [120] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.
- [121] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *Advances in neural information processing systems (NeurIPS) Workshops*, 2017.
- [122] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick. Detectron2. <https://github.com/facebookresearch/detectron2>, 2019.
- [123] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick. Mask R-CNN. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.
- [124] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.
- [125] A. Simonelli, S. Rota Bulo, L. Porzi, M. Lopez-Antequera, and P. Kotschieder. Disentangling monocular 3d object detection. In *IEEE International Conference on Computer Vision (ICCV)*, 2019.
- [126] F. Manhardt, W. Kehl, and A. Gaidon. Roi-10d: Monocular lifting of 2d detection to 6d pose and metric shape. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.
- [127] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia. Multi-view 3d object detection network for autonomous driving. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.
- [128] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *IEEE conference on computer vision and pattern recognition (CVPR)*, 2014.

- [129] R. Girshick. Fast r-cnn. In *IEEE International Conference on Computer Vision (ICCV)*, 2015.
- [130] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems (NeurIPS)*, 2015.
- [131] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *IEEE international conference on computer vision (ICCV)*, 2017.
- [132] T. Hodaň, P. Haluza, Š. Obdržálek, J. Matas, M. Lourakis, and X. Zabulis. T-LESS: An RGB-D dataset for 6D pose estimation of texture-less objects. *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2017.
- [133] A. Tejani, D. Tang, R. Kouskouridas, and T.-K. Kim. Latent-class hough forests for 3d object detection and pose estimation. In *European conference on computer vision (ECCV)*. Springer, 2014.
- [134] A. Doumanoglou, R. Kouskouridas, S. Malassiotis, and T.-K. Kim. 6d object detection and next-best-view prediction in the crowd. In *IEEE Computer Vision and Pattern Recognition (CVPR)*, volume 1, 2016.
- [135] Z. Xie, A. Singh, J. Uang, K. S. Narayan, and P. Abbeel. Multimodal blending for high-accuracy instance recognition. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2013.
- [136] A. Aldoma, T. Fäulhammer, and M. Vincze. Automation of “ground truth” annotation for multi-view rgb-d object instance recognition datasets. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2014.
- [137] C. Rennie, R. Shome, K. E. Bekris, and A. Ferreira De Souza. A dataset for improved rgb-d-based object detection and pose estimation for warehouse pick-and-place. *IEEE Robotics and Automation Letters (RA-L)*, 1, 02/2016 2016.
- [138] N. Correll, K. E. Bekris, D. Berenson, O. Brock, A. Causo, K. Hauser, K. Okada, A. Rodriguez, J. M. Romano, and P. R. Wurman. Analysis and observations from the first amazon picking challenge. *IEEE Transactions on Automation Science and Engineering*, 15(1), 2016.
- [139] A. Singh, J. Sha, K. S. Narayan, T. Achim, and P. Abbeel. Bigbird: A large-scale 3d database of object instances. In *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014.
- [140] T. Hodan, F. Michel, E. Brachmann, W. Kehl, A. GlentBuch, D. Kraft, B. Drost, J. Vidal, S. Ihrke, X. Zabulis, et al. Bop: Benchmark for 6d object pose estimation. In *European Conference on Computer Vision (ECCV)*, 2018.

BIBLIOGRAPHY

- [141] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6), 2014.
- [142] G. Bradski. The OpenCV Library. *Dr. Dobbs's Journal of Software Tools*, 2000.
- [143] Artec 3D. <https://www.artec3d.com/>. Accessed: 2019-03-23.
- [144] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia. MeshLab: an Open-Source Mesh Processing Tool. In V. Scarano, R. D. Chiara, and U. Erra, editors, *Eurographics Italian Chapter Conference*. The Eurographics Association, 2008.
- [145] N. Max. Weights for computing vertex normals from facet normals. *J. Graph. Tools*, 4(2), March 1999. URL: <http://dx.doi.org/10.1080/10867651.1999.10487501>, doi:10.1080/10867651.1999.10487501.
- [146] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012.
- [147] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '96*, New York, NY, USA, 1996. ACM. URL: <http://doi.acm.org/10.1145/237170.237269>, doi:10.1145/237170.237269.
- [148] B. Drost, M. Ulrich, N. Navab, and S. Ilic. Model globally, match locally: Efficient and robust 3d object recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010.
- [149] D. Eigen and R. Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.
- [150] P. Wang, X. Shen, Z. Lin, S. Cohen, B. Price, and A. L. Yuille. Towards unified depth and semantic prediction from a single image. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.
- [151] S. Gupta, J. Hoffman, and J. Malik. Cross modal distillation for supervision transfer. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
- [152] R. Kuga, A. Kanazaki, M. Samejima, Y. Sugano, and Y. Matsushita. Multi-task learning using multi-modal encoderdecoder networks with shared skip connections. In *IEEE International Conference on Computer Vision Workshop (ICCVW)*, 2017.

- [153] A. Kendall, Y. Gal, and R. Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *IEEE conference on computer vision and pattern recognition (CVPR)*, 2018.
- [154] D. Xu, W. Ouyang, X. Wang, and N. Sebe. Pad-net: Multi-tasks guided prediction-and-distillation network for simultaneous depth estimation and scene parsing. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.
- [155] N. Neverova, P. Luc, C. Couprie, J. Verbeek, and Y. LeCun. Predicting deeper into the future of semantic segmentation. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.
- [156] V. Mnih, N. Heess, A. Graves, et al. Recurrent models of visual attention. In *Advances in neural information processing systems (NeurIPS)*, 2014.
- [157] J. Ba, V. Mnih, and K. Kavukcuoglu. Multiple object recognition with visual attention. *arXiv*, 2014.
- [158] M.-T. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation. In *Conference on Empirical Methods in Natural Language Processing*, 2015.
- [159] J. Cheng, L. Dong, and M. Lapata. Long short-term memory-networks for machine reading. In *Conference on Empirical Methods in Natural Language Processing*, 2016.
- [160] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena. Self-attention generative adversarial networks. In *International Conference on Machine Learning (ICML)*. International Machine Learning Society (IMLS), 2019.
- [161] J. F. Blinn. Models of light reflection for computer synthesized pictures. In *Conference on Computer graphics and interactive techniques (SIGGRAPH)*, 1977.
- [162] M. Ounsworth. *Anticipatory Movement Planning for Quadrotor Visual Servoing*. PhD thesis, McGill University Libraries, 2015.
- [163] K. Perlin. Improving noise. In *ACM Transactions on Graphics (TOG)*, 2002.
- [164] S. Worley. A cellular texture basis function. In *Conference on Computer graphics and interactive techniques (SIGGRAPH)*. ACM, 1996.
- [165] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: common objects in context. In *European conference on computer vision (ECCV)*, 2014.
- [166] P. Y. Simard, D. Steinkraus, J. C. Platt, et al. Best practices for convolutional neural networks applied to visual document analysis. In *International Conference on Document Analysis and Recognition (ICDAR)*, 2003.

BIBLIOGRAPHY

- [167] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab. Deeper depth prediction with fully convolutional residual networks. In *International Conference on 3D Vision (3DV)*. IEEE, 2016.
- [168] K. Bousmalis, N. Silberman, D. Dohan, D. Erhan, and D. Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks. In *IEEE conference on computer vision and pattern recognition (CVPR)*, 2017.
- [169] W. Kehl, F. Manhardt, F. Tombari, S. Ilic, and N. Navab. Ssd-6d: Making rgb-based 3d detection and 6d pose estimation great again. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.
- [170] F. Manhardt, W. Kehl, N. Navab, and F. Tombari. Deep model-based 6d pose refinement in rgb. In *European conference on computer vision (ECCV)*, 2018.
- [171] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu. Spatial transformer networks. In *Advances in neural information processing systems (NeurIPS)*, 2015.
- [172] S. Hinterstoisser, V. Lepetit, P. Wohlhart, and K. Konolige. On pre-trained image features and synthetic images for deep learning. In *European Conference on Computer Vision Workshops (ECCVW)*, 2018.
- [173] B. T. Phong. Illumination for computer generated pictures. *Communications of the ACM*, 1975.
- [174] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *IEEE transactions on pattern analysis and machine intelligence (TPAMI)*, 2011.
- [175] P. Wohlhart and V. Lepetit. Learning descriptors for object recognition and 3d pose estimation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.
- [176] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. M. Lopez. The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
- [177] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
- [178] E. Tzeng, J. Hoffman, N. Zhang, K. Saenko, and T. Darrell. Deep domain confusion: Maximizing for domain invariance. *arXiv*, 2014.

- [179] M. Long, Y. Cao, J. Wang, and M. I. Jordan. Learning transferable features with deep adaptation networks. *International Conference on Machine Learning (ICML)*, 2015.
- [180] H.-Y. Lee, H.-Y. Tseng, J.-B. Huang, M. Singh, and M.-H. Yang. Diverse image-to-image translation via disentangled representations. In *European conference on computer vision (ECCV)*, 2018.
- [181] D. Q. Huynh. Metrics for 3d rotations: Comparison and analysis. *Journal of Mathematical Imaging and Vision*, 2009.
- [182] J. Hoffman, D. Wang, F. Yu, and T. Darrell. Fcns in the wild: Pixel-level adversarial and constraint-based adaptation. *arXiv*, 2016.
- [183] Y. Zhang, P. David, and B. Gong. Curriculum domain adaptation for semantic segmentation of urban scenes. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.
- [184] Y.-H. Chen, W.-Y. Chen, Y.-T. Chen, B.-C. Tsai, Y.-C. Frank Wang, and M. Sun. No more discrimination: Cross city adaptation of road scene segmenters. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.
- [185] Y.-H. Tsai, W.-C. Hung, S. Schulter, K. Sohn, M.-H. Yang, and M. Chandraker. Learning to adapt structured output space for semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.
- [186] Y. Chen, W. Li, and L. Van Gool. Road: Reality oriented adaptation for semantic segmentation of urban scenes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.
- [187] S. Sankaranarayanan, Y. Balaji, A. Jain, S. Nam Lim, and R. Chellappa. Learning from synthetic data: Addressing domain shift for semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.
- [188] Y. Chen, W. Li, X. Chen, and L. V. Gool. Learning semantic segmentation from synthetic data: A geometrically guided input-output adaptation approach. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.
- [189] M. Sundermeyer, Z.-C. Marton, M. Durner, M. Brucker, and R. Triebel. Implicit 3d orientation learning for 6d object detection from rgb images. In *European conference on computer vision (ECCV)*, 2018.
- [190] T. Jeruzalski, B. Deng, M. Norouzi, J. P. Lewis, G. Hinton, and A. Tagliasacchi. Nasa: Neural articulated shape approximation. *arXiv*, 2019.
- [191] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, and M. J. Black. Smpl: A skinned multi-person linear model. *ACM transactions on graphics (TOG)*, 34(6), 2015.

BIBLIOGRAPHY

- [192] S. Zuffi, A. Kanazawa, D. W. Jacobs, and M. J. Black. 3d menagerie: Modeling the 3d shape and pose of animals. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017.
- [193] T. Hodaň, V. Vineet, R. Gal, E. Shalev, J. Hanzelka, T. Connell, P. Urbina, S. Sinha, and B. Guenter. Photorealistic image synthesis for object instance detection. *IEEE International Conference on Image Processing (ICIP)*, 2019.