18th IEEE International Conference on Automation Science and Engineering (CASE), Mexico City, Mexico, August 2022.

Synthetic-to-Real Domain Adaptation using Contrastive Unpaired Translation

Benedikt T. Imbusch[1], Max Schwarz[1], and Sven Behnke[1]

***Abstract*— The usefulness of deep learning models in robotics is largely dependent on the availability of training data. Manual annotation of training data is often infeasible. Synthetic data is a viable alternative, but suffers from domain gap. We propose a multi-step method to obtain training data without manual annotation effort: From 3D object meshes, we generate images using a modern synthesis pipeline. We utilize a state-of-the-art image-to-image translation method to adapt the synthetic images to the real domain, minimizing the domain gap in a learned manner. The translation network is trained from unpaired images, i.e. just requires an un-annotated collection of real images. The generated and refined images can then be used to train deep learning models for a particular task. We also propose and evaluate extensions to the translation method that further increase performance, such as patch-based training, which shortens training time and increases global consistency. We evaluate our method and demonstrate its effectiveness on two robotic datasets. We finally give insight into the learned refinement operations.**

I. Introduction

Robotic systems need to address several key challenges in order to be able to autonomously act in complex environments. Among these are computer vision tasks like semantic segmentation, object recognition, and 6D pose estimation. Nowadays, these tasks are most commonly solved using deep learning techniques. With increasing computation resources available, more complex network architectures are developed, raising the need for increasing amounts of training data. Acquiring training data, however, often involves tedious manual annotation of images with semantic labels or 6D poses. It is typically not feasible to create custom datasets for every specific task at hand.

To overcome this issue, previous approaches successfully relied on fine-tuning of networks pre-trained on generic datasets, reducing the required annotation effort [1], [2]. Recently, approaches were introduced that generate synthetic training images, e.g. from 3D object meshes like *Stillleben* [3]. The benefit of such techniques is that ground truth data like 6D object poses or semantic segmentation masks are trivially available from the renderer, eliminating the need for manual annotation while providing highly accurate annotations. Although Stillleben yields good generalization to real test images on the *YCB-Video* dataset [4] for semantic segmentation [3], the achieved results are still considerably inferior compared to training on real images. The reason for this difference is the so-called domain gap between synthetic and real data, i.e. the discrepancy between the synthetic data distribution and the real data distribution. Therefore, the model learned by a segmentation network trained on synthetic data is able to only partly capture the real data distribution from which test data is sampled.

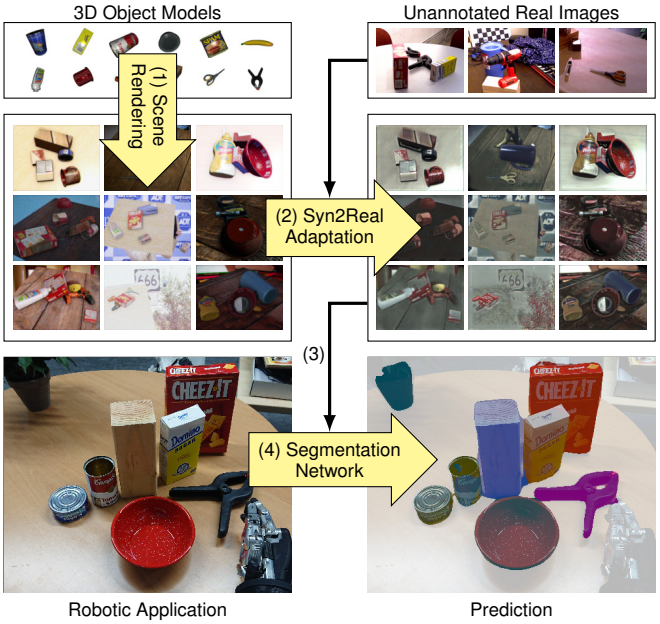

Fig. 1. Our method yields robust task performance in real settings, just from 3D object models and unannotated real images (top). We simulate and render plausible scenes from the 3D meshes (1). Our adaptation model aligns the synthetic and real image distributions more closely (2). The refined image dataset is used to train a task-specific network (3), which is applied in the target domain (4). None of these steps requires annotations.

We aim to obtain better results from purely synthetic data and therefore need to align the distributions more closely. We propose to perform this domain adaptation by learning a mapping from the synthetic to the real image distribution in an unsupervised manner. Specifically, this means that we only require synthetic data with ground truth and unannotated real data without direct correspondences between the images of both datasets. To learn this mapping, we apply the GAN-based *CUT* approach by Park *et al.* [5] in a patch-based manner. Key challenges to be addressed here include the handling of backgrounds and ensuring shape consistency. We evaluate the method on a semantic segmentation task on both the YCB-Video dataset and the *HomebrewedDB* dataset [6]. The individual steps of our method are visualized in Fig. 1. To understand how the observed performance improvements can be explained, we further examine deep image features of real, synthetic, and refined synthetic frames

[1]All authors are with Autonomous Intelligent Systems, University of Bonn, Germany. benedikt.imbusch@uni-bonn.de

using t-SNE embeddings. In short, our contributions include:

1) A multi-step method to obtain annotated training data from 3D object meshes and un-annotated images which can later be used for downstream applications like robotic manipulation, yielding performance close to training on real data,
2) a patch-based application of the CUT approach to domain adaptation for two robotics datasets, and
3) an analysis of the learned refinement operations using t-SNE embeddings.

In the following, we denote the synthetic data distribution as $\mathcal{X}$ and a set of samples thereof as $x \in X \subsetneq \mathcal{X}$. Likewise, the real data distribution is denoted by $\mathcal{Y}$. Thus, the learned mapping can be formalized as

$$f : \mathcal{X} \to \mathcal{Y}. \tag{1}$$

II. Related Work

Domain adaptation using deep neural networks is an established area of research. Wang and Deng [7] group the approaches into two main categories: heterogeneous and homogeneous domain adaptation. The former refers to the case when the domain gap arises from source and target domain having different feature spaces. In the latter case, both domains share their feature space but still the respective distributions $\mathcal{X}$ and $\mathcal{Y}$ do not match. In this work, we address the homogeneous case, as do the related approaches below.

Stein and Roy [8] apply domain adaptation to warehouse and outdoor scenes using the *CycleGAN* [9] approach. Similar to our method, they address a semantic segmentation task and use separate networks for domain adaptation and segmentation. However, the system is applied to robotic navigation and larger-scale scene understanding, compared to robotic manipulation in small-scale scenes in our case. This might explain why our initial experiments with CycleGAN did not yield satisfactory results. The CUT architecture employed by our approach is easier to train and generally yields better results [5].

In similar manner, Mueller *et al.* [10] successfully showed the use of CycleGAN-based domain adaptation for synthetic training data. Their application domain is hand pose tracking. To ensure accurate preservation of the hand poses, they propose *GeoConGAN*, adding a geometric consistency loss to the CycleGAN objective. Using the newer CUT approach, we are able to avoid geometric inconsistencies by applying it in a patch-based manner. Therefore, adding complexity in the form of another loss component calculated using an additional CNN appears not justified to us.

Shrivastava *et al.* [11] use a GAN approach with a patch-based discriminator for synthetic-to-real domain adaptation of hands and eyes with the goal of pose estimation. They use L1-regularization on (identity or more complex) transformed image features to constrain the GAN towards content preservation. The GAN's discriminator is trained on batches of refined images accumulated over time, to stabilize the adversarial learning. CUT's contrastive learning-based approach for content consistency appears far more flexible and data-adapting to us, compared to the proposed L1-regularization.

Bousmalis *et al.* [12] propose *PixelDA*, another GAN-based approach for domain adaptation. Like in our scenario, they apply it to small objects but focus more on classification and pose estimation while highlighting broader applicability. In addition to the standard setup based on a generator and a discriminator, they add a task-specific classifier to their model, trained on both synthetic and generator-refined synthetic images to support the domain adaptation. To maintain correspondences between the synthetic images and their refined versions, they propose to penalize content dissimilarities using a masked pairwise mean squared error, given depth data available from the renderer. The resulting generated backgrounds, mainly replacing black backgrounds, appear rather noisy to us. While this might even benefit generalization for classification, we expect that more consistent backgrounds are needed in our case. Additional experiments that we performed at full resolution with the CUT architecture have shown a detrimental effect of masking out the backgrounds in our application domain.

CyCADA by Hoffman *et al.* [13] is another domain adaptation approach derived from the idea of CycleGAN. This technique guides the adaptation process in two ways: The authors propose loss components for aligning the distributions both in the pixel space and the feature space. Besides, the authors suggest to use loss components specific to the subsequent deep learning task to enforce semantic consistency. Hoffman *et al.* [13] report better performance on a semantic segmentation task after domain adaptation than for the existing unsupervised adaptation approaches. However, the training is computationally costly and complex—compared to the far simpler but still very effective objective of CUT.

The *DLOW* technique proposed by Gong *et al.* [14] is based on the CycleGAN concept as well. It generalizes the idea of domain adaptation beyond mapping a source domain $\mathcal{S}$ to a target domain $\mathcal{T}$: The authors introduce a model for "domain flow generation". Intuitively, a parameter $z \in [0, 1]$ is introduced to control how far an image from $\mathcal{S}$ should be adapted towards $\mathcal{T}$. A mentioned key benefit of this technique is that learning the intermediate steps supports the domain adaptation process. In their experiments, Gong *et al.* [14] show improved results on a semantic segmentation task compared to plain CycleGAN domain adaptation. However, the improvement is not very substantial. The previously mentioned shortcomings of CycleGAN compared to CUT apply for this approach as well.

III. Method

We propose a method to obtain images for the training of a deep neural network for tasks like semantic segmentation, focusing on training data for robotic manipulation. It consists of multiple sub-steps, as illustrated in Fig. 1: First, we generate synthetic images using the Stillleben [3] library. Based on the image-to-image translation architecture CUT [5] and un-annotated real images, we then refine these synthetic images towards more realism. The refined images can then be used

for training the task network. In the following, we describe the components of our approach.

A. Stillleben

The Stillleben [3] library is a framework for generation & rendering of cluttered tabletop scenes. Stillleben operates on arbitrary input meshes and generates random arrangements through the use of a physics engine. The arranged scenes are then rendered with a modern physics-based-rendering (PBR) pipeline, producing realistic images. A post-processing step adds effects simulating the usage of a real camera, such as noise, chromatic aberration, white balancing errors, and over-/underexposure. A segmentation model trained with purely Stillleben-generated synthetic data has been shown to reach respectable performance on the YCB-Video dataset [3].

B. Contrastive Unpaired Translation (CUT)

Our domain adaptation approach is largely based on *Contrastive Unpaired Translation (CUT)* as introduced by Park *et al.* [5], which we briefly introduce here. It is an image-to-image translation technique, aimed at preserving the image content while adapting the appearance to the target domain. CUT is related to the well-known CycleGAN approach by Zhu *et al.* [9] which pursues the same objective. Both are GAN-based, can be used for unpaired image sets, and have to address the same key issue: Training a GAN for unpaired image-to-image translation is in general an under-constrained task. CycleGAN employs a second GAN for a reverse mapping from the target to the source domain. The method enforces correspondences between input and output image by a cycle-consistency loss that penalizes differences resulting from passing an image through both the forward and the reverse GAN subsequently. This avoids collapse of the generator, i.e. mapping all inputs to a single output in the target domain.

CUT removes the second GAN and replaces the cycle consistency loss with a contrastive loss on image patches, the so-called *PatchNCE* loss. In brief, the idea is to achieve content preservation by ensuring that a patch of the translated image has more information in common with the same patch in the source image (positive) than with N other patches from the source image (negatives). Technically, this is realized by training a small MLP classifier to select the positive from the $N+1$ source patches, given the translated patch. This is done in the GAN encoder's feature space, separately for each layer used. The concept is illustrated in Fig. 2. In CUT, the PatchNCE is also calculated for the image from the target distribution to stabilize the training by hinting the network to keep images from the target domain identical. The complete loss function thus is given by

$$\mathcal{L} = \mathcal{L}_{\text{GAN}}(G, D, X, Y) + \lambda_{\text{NCE}} \cdot (\mathcal{L}_{\text{PatchNCE}}(G, H, X) + \mathcal{L}_{\text{PatchNCE}}(G, H, Y)), \quad (2)$$

where G denotes the GAN's generator, D the discriminator, X the source image and Y the target image. H denotes the MLPs used for the PatchNCE. Park *et al.* [5] suggest to choose $\lambda_{\text{NCE}} = 1$.

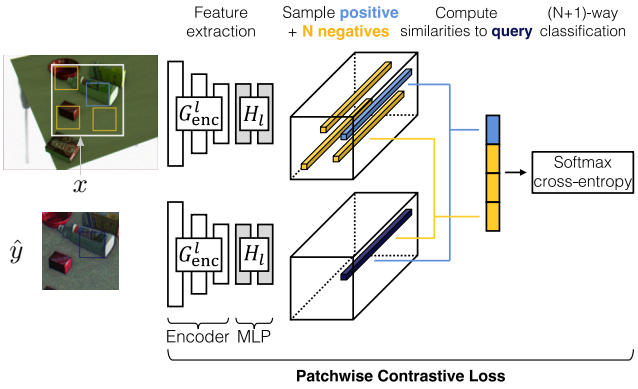

Fig. 2. The PatchNCE is calculated based on the selected patch x from the synthetic image and the corresponding generated refined image $\hat{y}$. From these smaller images, subpatches are selected for the calculation. Adapted from [5].

(a) synthetic (b) refined

Fig. 3. A synthetic image and a CUT-refined version of it with CUT trained on full-resolution images. Note how training at full resolution leads to deformations and hallucinations of objects in the background image.

C. Modifications to CUT

Compared to CycleGAN, we selected the CUT approach for its lesser complexity at better performance (see [5]).

Its authors propose to apply CUT to images at full resolution. However, we decided to train it in a patch-based way. There are several reasons for this: First, we have substantial variability in the images produced by Stillleben, especially with respect to the backgrounds, and a large set of real images. It seems sensible to us to use many of these images for the training in order to achieve good generalization to unseen images. Working at full resolution (640×480 for YCB-Video), however, induces long training times when following the learning duration of 400 epochs proposed by Park *et al.* [5]. Second, the changes to the source image we aim to achieve are at small scale. Ideally, our domain-adapted images reflect the visual properties induced by the camera used for the real images but are content-wise very close to the source data to keep the segmentation labels usable. Experimental results support this motivation. CUT models trained at full resolution often deform relevant objects or hallucinate parts of them in new places, as can be seen in Fig. 3. While the training is performed on patches, inference is still possible at full resolution, thanks to the GAN generator's architecture. We thus argue that it is sufficient and even beneficial to work on image patches.

The selected patch size has to be small enough to notably

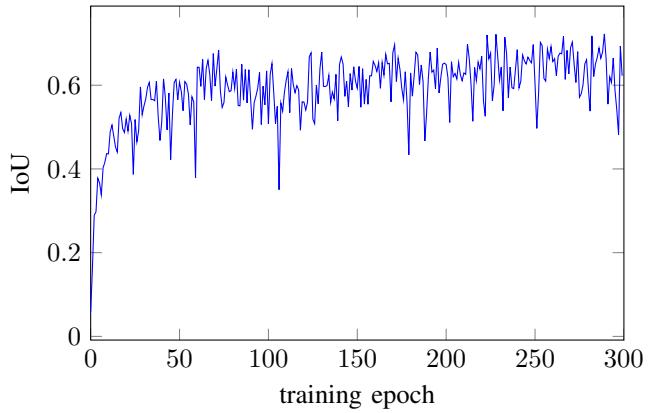

Fig. 4. Test IoU on YCB-Video for training on synthetic images over 300 epochs shows significant variance between epochs.

reduce the computation effort and prevent global effects. At the same time, it has to be large enough to still contain sufficient information and to ensure that sampling sub-patches for the PatchNCE is still possible in a meaningful way. We propose and evaluate patch sizes between 60^2 and 160^2 pixels. The patch selection is done by random cropping.

IV. Evaluation

A. Experimental Setup and Evaluation Metric

In many applications, the goal of visual domain adaptation is to create images that look appealing to the human eye. In our robotics use case, however, we are not mainly interested in well-looking images but in images that yield better results on subsequent data processing tasks than the original synthetic images.

Therefore, we evaluate our method on a semantic segmentation task similar to how it is done in [3]: We use *RefineNet* [15], an established network architecture for semantic segmentation, and train it from scratch on 450k images, subdivided into 300 epochs of 1500 images. The segmentation performance is evaluated on a test set of annotated real images by calculating the mean intersection over union (mean IoU or mIoU) over all classes present in the dataset. All IoU values in the following are calculated on the respective test sets.

We are interested in the performance on test data after training on three image sets: synthetic images from Stillleben, CUT-refined synthetic images from Stillleben, and also real images (disjoint from the test set) for comparison. For the refined images, we use the ground-truth labels provided by the renderer for the corresponding unrefined images. Ground-truth labels for both the training and test real images are provided as part of the datasets used for the evaluation.

As can be seen in Fig. 4, the performance on the test data differs significantly between epochs during the training of RefineNet. In the following, we therefore always visualize the distribution of IoU values over the 50 last training epochs instead of just indicating the IoU value for the final epoch.

To obtain the refined images, we pass the synthetic images through the CUT generator. Before, we train CUT on unpaired sets of synthetic and real images for 400 epochs, following the curriculum suggested by Park *et al.* [5]. For the training of CUT, we choose a batch size of 40—irrespective of the patch size. This is to keep the sources of variation between the results for different patch sizes limited. The deviation from the standard batch size of 1 for CUT has two main reasons: First, the training time can be substantially reduced while improving memory usage. Second, we argue that for the patch-based application of CUT it is beneficial to use more averaged gradients, especially as some of the randomly sampled patches may consist entirely of irrelevant background information. Besides, we do not horizontally flip images for data augmentation to only expose the network to images that could occur in reality. Further deviations from the defaults of the standard implementation are stated in the respective experiment descriptions for the two datasets considered. All training of CUT has been performed using NVIDIA A100-SXM4-40GB accelerator cards.

We mainly work with the YCB-Video dataset but demonstrate the broader applicability of our approach on the HomebrewedDB dataset as well.

B. Results on YCB-Video

Stillleben has been evaluated on the YCB-Video dataset [3], thus this is the best-suited dataset for a comparative evaluation. For the training of CUT, we use 10k images generated using Stillleben and 10k images from the training set of YCB-Video.

The first investigated aspect is the patch size, which we choose between 60^2 and 160^2 pixels as mentioned above. It is worth noting that internally, CUT works with patch sizes that are multiples of 4. Otherwise, patches are rescaled to have such size using bicubic interpolation. While it appears possible that this interpolation would introduce some beneficial or detrimental smoothing to the input images, we saw no consistent effect on the results. We evaluate the performance for the following patch sizes: 60^2, 70^2, 90^2, 100^2, 120^2, and 160^2 pixels. The results are shown in Fig. 5, alongside the results for RefineNet trained on purely synthetic and real YCB-Video images. It can be seen that especially at patch size 60^2, but to some degree still for 70^2, only insufficient information is conveyed for this dataset—compared to the larger patch sizes. 160^2 and 90^2 appear to yield the best results. Given the fact that the training time for CUT largely depends on the patch size (see Table I), 90^2 seems to be the best trade-off. Consequently, using CUT-refined synthetic images offers a significant benefit over using pure Stillleben images as the performance is close to what training on real data yields.

Apart from the general performance increase, CUT-refining synthetic images offers another benefit: Irrespective of the patch size, refining the images leads to a significantly narrower distribution of the IoU values, closer to real data, which yields the narrowest distribution. In contrast, the variance is rather high for training on synthetic images.

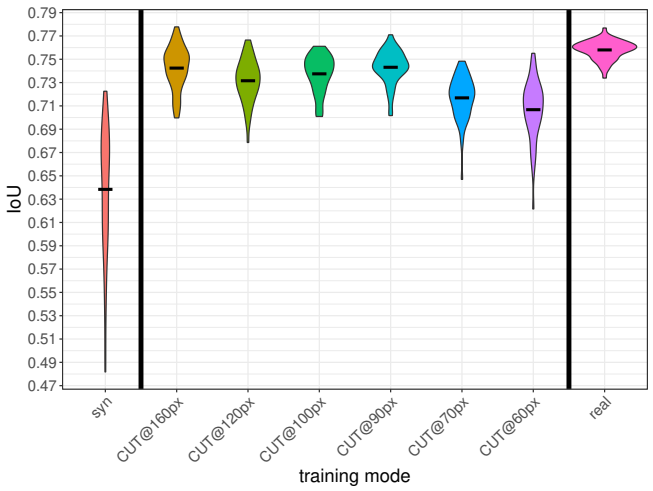

Fig. 5. Results on YCB-Video. The test IoU distribution over the last 50 training epochs for CUT-refined images for CUT patch sizes 160^2, 120^2, 100^2, 90^2, 70^2, and 60^2. The leftmost plot depicts the test results for training with synthetic images, the rightmost plot for training with real images. Training with CUT-refined synthetic images not only yields higher IoU values than pure synthetic images but also narrower distributions.

TABLE I
PATCH SIZE & TRAINING EPOCH TIME

patch size [px]	time [s]
160×160	181
120×120	110
100×100	82
90×90	66
70×70	54
60×60	43

C. Modifications

Synthetic images can imitate image noise, for instance as proposed by Foi *et al.* [16]. Noise from real cameras, however, exposes properties that are hard to model and is inherently random. It could be the case that at the synthetic-to-real domain adaptation task, it is hard for a GAN to add such noise to an image. To address this and even further increase the IoU, we tested noise injection to add noise within the translation process. Given the GAN's encoder-decoder architecture, we decided to inject it directly at the input of the decoder by adding N normally-distributed random feature maps to the M feature maps from the encoder, where $N \ll M$. To ease the noise integration and return to the previous number of feature maps, we add three convolution layers before the actual decoder part. Using a patch size of 160^2 pixels, we tested injecting 0, 4, 8, 16 and 32 random feature maps. The results can be seen in Fig. 6. While 8 feature maps appear to even have a detrimental effect across multiple runs of this experiment, the general impression is that injecting noise is not beneficial. This contradicts the hypothesis that additional randomness apart from the noise added by Stillleben's camera model helps the GAN to closer match the real image distribution. We hypothesize that the artificial image noise from Stillleben is sufficient to produce images that cannot too easily be distinguished from real images by the GAN discriminator based on the kind of noise.

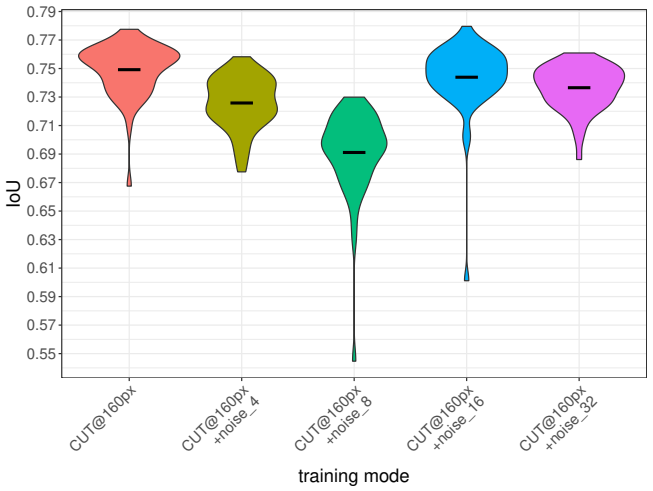

Fig. 6. Noise injection. The test set IoU distributions for 0, 4, 8, 16, and 32 injected random feature maps show no beneficial effect of injecting noise.

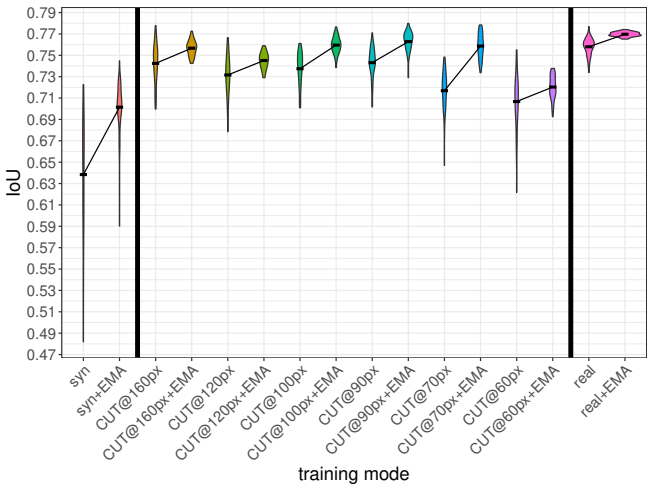

Fig. 7. Using exponential moving average (EMA) for the RefineNet parameters improves the performance and reduces the IoU variability.

Another change however is beneficial, but not directly related to CUT: Using an exponential moving average (EMA) of the RefineNet model parameters for the evaluation on the test set does significantly improve the performance and reduce the variability of the IoU (decay factor: 0.995). This is consistent over all patch sizes for CUT-refined images as well as synthetic and real images, as can be seen in Fig. 7. Note that we achieve more than 99% of the mean IoU for real training data, see Table II. We therefore use this modification in the following for the experiments with HomebrewedDB.

D. Results on HomebrewedDB

Encouraged by the promising results on the YCB-Video dataset, we also evaluate our approach on the HomebrewedDB [6] dataset. Both datasets consist of small objects on a table. However, the overall image appearance and the

TABLE II
RESULTS ON YCB-VIDEO (WITH EMA).

Training mode	mIoU (↑)	vs. Real (↑)	vs. Syn. (↑)
synthetic [3] +EMA	0.701	0.910	—
$\mathrm{CUT}_{160\mathrm{px}}$	0.757	0.983	+8.0%
$\mathrm{CUT}_{120\mathrm{px}}$	0.745	0.968	+6.3%
$\mathrm{CUT}_{100\mathrm{px}}$	0.760	0.987	+8.4%
$\mathrm{CUT}_{90\mathrm{px}}$	**0.763**	**0.991**	**+8.8%**
$\mathrm{CUT}_{70\mathrm{px}}$	0.759	0.986	+8.3%
$\mathrm{CUT}_{60\mathrm{px}}$	0.720	0.935	+2.7%
real [3] +EMA	0.770	1.000	+9.8%

presented arrangements differ by a lot. We use the HomebrewedDB data offered in the *BOP* challenge[1]. For both the offered test images (*BOP'19/20 test images (Primesense)*) and the validation images (*Validation images (Primesense)*), ground truth semantic annotations are included. No real training images are provided. However, 3D meshes for all objects are available, allowing us to use Stillleben.

We restrict our evaluation to the subset S of HomebrewedDB objects which are present in the test images. We generate synthetic scenes with objects from S and train CUT on validation images containing objects from S. The model trained on real data is trained analogously. In all cases, the official test set is used to evaluate the segmentation performance of the trained RefineNet models.

For generating the synthetic images using Stillleben, we make two slight modifications compared to the process proposed for YCB-Video by Schwarz and Behnke [3], based on the appearance of the real images: We remove the stickers randomly added to the objects and instead of rendering the objects on a textured table, a white table is used. Without any further adaptation to HomebrewedDB, we reach IoU values around 0.5 which are inferior to those achieved for YCB-Video. The achieved IoU for real images is slightly lower than for YCB-Video, see Fig. 8.

For the training of CUT, we use 10k synthetic images and all 1020 real validation images. Obviously, most real images are presented multiple times during each epoch. Due to our patch-based training, the shown part of the image however varies. Based on our results on YCB-Video, we restrict ourselves to patch sizes of 70^2 and 90^2 pixels.

With the same hyperparameter choices as for YCB-Video, we saw segmentation performance inferior to that of pure Stillleben. Looking at the produced images, the reason is a mode collapse: Most of the images are grey-textured, with the object shapes barely visible. This is consistent over multiple runs and patch sizes. A reason might be that for this dataset and the respective synthetic images produced by Stillleben, the loss weighting insufficiently ensures content preservation. Therefore, we propose to increase the weight of the PatchNCE while keeping it low enough to allow for meaningful changes to the appearance. The results for $\lambda_{\mathrm{NCE}} = 2,\ 5,\ 7$ can be seen in Fig. 8 for both patch

[1]https://bop.felk.cvut.cz/datasets/

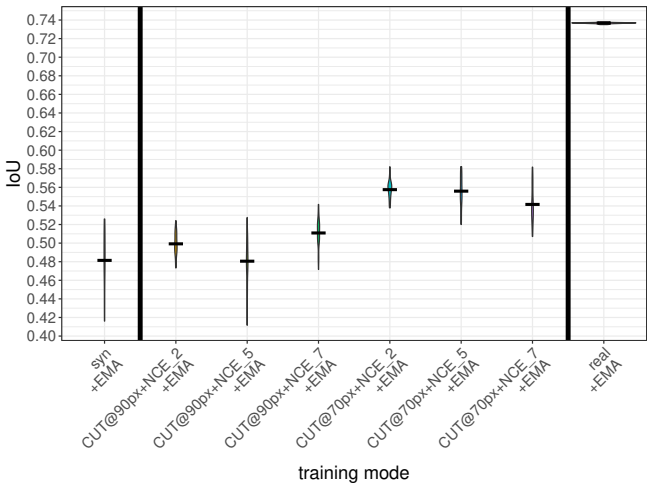

Fig. 8. Results on HomebrewedDB. We show the test IoU distribution over the last 50 training epochs for CUT-refined images for CUT patch size 90^2 and $\lambda_{\mathrm{NCE}} = 2,\ 5,\ 7$ as well as patch size 70^2 with the same values for λ_{NCE} (from left to right), complemented by the results for training on synthetic (left) and real images (right).

TABLE III
RESULTS ON HOMEBREWEDDB (WITH EMA).

Training mode	Mean IoU (↑)	vs. Real (↑)	vs. Syn. (↑)
synthetic	0.481	0.653	—
$\mathrm{CUT}_{90\mathrm{px},\lambda_{\mathrm{NCE}}=2}$	0.499	0.677	+3.7%
$\mathrm{CUT}_{90\mathrm{px},\lambda_{\mathrm{NCE}}=5}$	0.481	0.653	+0.0%
$\mathrm{CUT}_{90\mathrm{px},\lambda_{\mathrm{NCE}}=7}$	0.511	0.693	+6.2%
$\mathrm{CUT}_{70\mathrm{px},\lambda_{\mathrm{NCE}}=2}$	**0.558**	**0.757**	**+16.0%**
$\mathrm{CUT}_{70\mathrm{px},\lambda_{\mathrm{NCE}}=5}$	0.556	0.754	+15.6%
$\mathrm{CUT}_{70\mathrm{px},\lambda_{\mathrm{NCE}}=7}$	0.542	0.735	+12.7%
real	0.737	1.000	+53.2%

sizes considered. A modest increase of $\lambda = 2$ appears to be favorable, as does a patch size of 70^2—with regards to the IoU and it's variability as well. With these changes, we see a significant improvement over the results using raw Stillleben images, quantitatively larger than for YCB-Video (with EMA), see Table III. Not only the IoU is increased by CUT-refining the synthetic images, but also the IoU variance over the training epochs of RefineNet is reduced—consistent with what we see for YCB-Video.

Thus, our approach is applicable also beyond YCB-Video with only minor changes needed for a related dataset.

E. Combination with Real Training Data

Until now, we considered the case where we have no real training data and aim to enhance the usefulness of synthetic data. However, there might also be cases where we have real training data available but the achieved performance is not good enough. In their experimental setup, Schwarz and Behnke [3] have shown that it is beneficial to train RefineNet on synthetic and real data at the same time by randomly choosing the mini-batches from both datasets. One might wonder whether the use of CUT-refined images enhances this effect. The results for both YCB-Video and HomebrewedDB

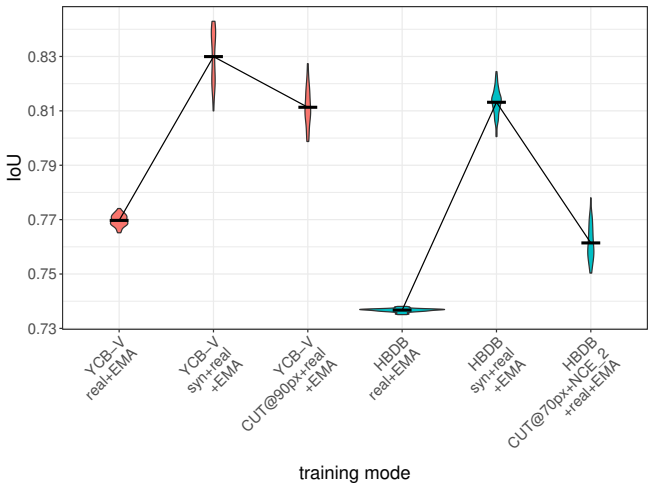

Fig. 9. Mixing synthetic and real data. We show the test IoU distribution over the last 50 training epochs for real training images, real and synthetic images mixed, as well as real and refined synthetic images mixed, for YCB-Video and HomebrewedDB, respectively.

(a) synthetic (b) refined

Fig. 10. Synthetic images and their CUT-refined versions.

are depicted in Fig. 9. While the achieved IoU on the respective test sets is still higher with refined images than without, the effect is less pronounced than is for synthetic data. From this, we hypothesize that using synthetic data has a regularizing influence on the training with real data, namely that the learned features are more domain-invariant. This effect is smaller for images refined towards more realism.

F. Analysis of Refinement Operations

We are mainly interested in achieving good segmentation performance to improve the value of synthetic training data for robotic applications. Still, it is worth analyzing what CUT is actually doing to the synthetic images. Figs. 10a) and b) show three synthetic images based on the YCB-Video objects and their CUT-refined versions, respectively.

While some refined images look more natural, some do not seem realistic to the human eye at all. Still, we achieve generalization to real data on the YCB-Video segmentation task that is close to what we get when training on real data. Hence, we hypothesize that—even if not for the human eye—refining the images aligns the synthetic and real image distributions more closely in the feature space of a CNN.

As we cannot investigate this in the high-dimensional feature space directly, we employ t-SNE embeddings [17] to project the data into two dimensions. Specifically, we use the YCB-Video keyframes, render corresponding Stillleben images using the pose annotations and refine them using a trained CUT generator. As we are more interested in what happens to the appearance of the objects in the images rather than in the backgrounds, we mask out the backgrounds using the segmentation masks provided by Stillleben. For the resulting set of triples, we calculate the feature maps of one extraction layer of RefineNet trained on real YCB-Video images. We apply adaptive average pooling with output size 1×1 to each feature map to obtain a vector of scalar values. Based on these vectors for all images, we calculate the t-SNE embeddings. The results vary based on the regarded layer. Embeddings based on features from an early extraction layer are depicted in Fig. 11. It can be seen that subsequent keyframes of the real YCB-Video video sequences are closely aligned. Besides, the synthetic images appear to form two clusters for which it was not possible to reliably determine their origin. We hypothesize that this split might be an artifact introduced by the tendency of t-SNE embeddings to form clusters, see [18]. The general impression is that CUT-refining the images both spreads the distribution and also aligns the distribution closer to the real image distribution. For some images, the refined synthetic images are embedded quite close to the corresponding real images, as indicated by the exemplary red line in Fig. 11. In later extraction layers of RefineNet, the effect is less clearly visible but still present, see Fig. 12. We saw similar effects for the early layers of AlexNet [19] trained on ImageNet[2], supporting the hypothesis that the domain adaptation actually does align the distributions more closely.

V. Discussion & Conclusion

We have presented a combined approach to generate training data from 3D object meshes using the synthesis pipeline Stillleben and unsupervised domain adaptation. We demonstrated the beneficial effect of our approach compared to purely synthetic data for a segmentation task on two robotics datasets, with only minor differences in the hyperparameters. For YCB-Video, the achieved segmentation performance is close to the performance with real training data. We explicitly remark that to apply our approach in new

[2]https://www.image-net.org/index.php

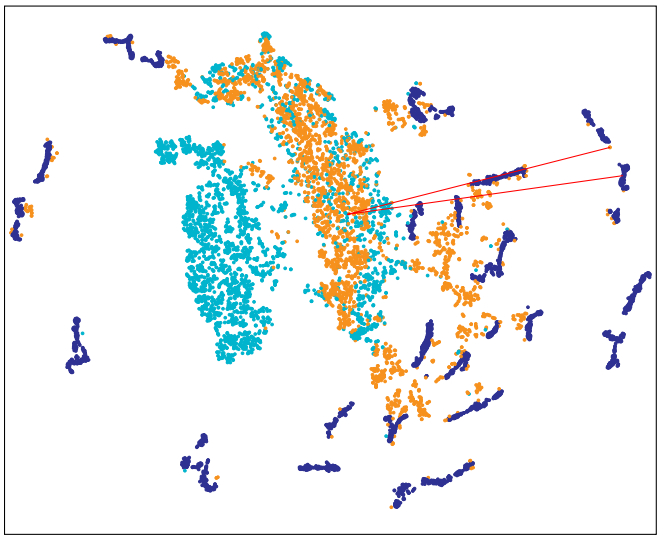

Fig. 11. t-SNE embeddings for an early extraction layer of RefineNet (turquoise: synthetic, orange: refined synthetic, blue: real images). The red line connects a corresponding synthetic-refined-real tuple.

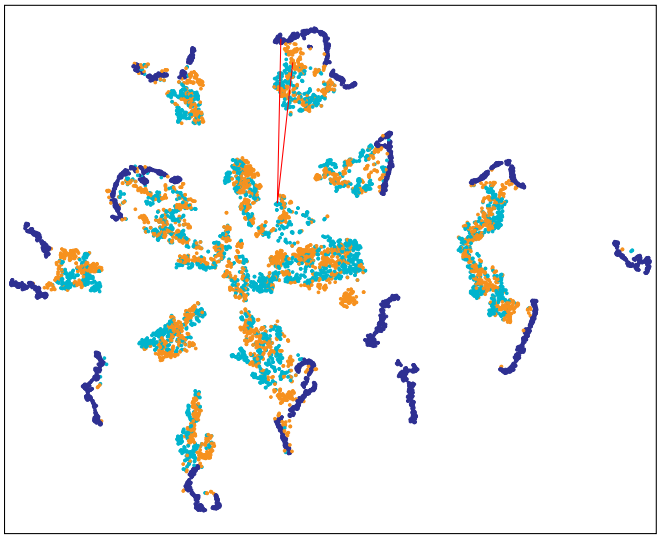

Fig. 12. t-SNE embeddings for a late extraction layer of RefineNet (turquoise: synthetic, orange: refined synthetic, blue: real images). The red line connects a corresponding synthetic-refined-real tuple.

situations, like robotic competitions, only object meshes and images from real cameras are required, thus no annotation is necessary. With state-of-the-art GPU hardware, obtaining the necessary refined training data for a new environment is possible in far less than a day. From that, we conclude that our approach has the potential to be applicable in real robotic setups without requiring any real annotations.

A limitation inherited from Stillleben is the dependence on high-quality object meshes. Additionally, the performance we achieve is close to what is possible on real data but does not yet match it and the variation of the performance over the segmentation training is notably higher than for real images. Another point worth mentioning is that the analysis of the t-SNE embeddings only partly explains the good results; more research into this direction would be beneficial, also to find further room for improvement.

Acknowledgement

This research has been funded by the Federal Ministry of Education and Research of Germany as part of the competence center for machine learning ML2R (01IS18038C).

References

[1] M. Schwarz, C. Lenz, G. M. García, S. Koo, A. S. Periyasamy, M. Schreiber, and S. Behnke, "Fast object learning and dual-arm coordination for cluttered stowing, picking, and packing," in *International Conference on Robotics and Automation (ICRA)*, 2018.

[2] D. Morrison, A. W. Tow, M. Mctaggart, R Smith, N. Kelly-Boxall, S. Wade-Mccue, J. Erskine, R. Grinover, A. Gurman, T Hunn, *et al.*, "Cartman: The low-cost cartesian manipulator that won the Amazon Robotics Challenge," in *International Conference on Robotics and Automation (ICRA)*, 2018, pp. 7757–7764.

[3] M. Schwarz and S. Behnke, "Stillleben: Realistic scene synthesis for deep learning in robotics," in *International Conference on Robotics and Automation (ICRA)*, 2020, pp. 10 502–10 508.

[4] 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.

[5] T. Park, A. A. Efros, R. Zhang, and J.-Y. Zhu, "Contrastive learning for unpaired image-to-image translation," in *European Conference on Computer Vision (ECCV)*, Springer, 2020, pp. 319–345.

[6] R. Kaskman, S. Zakharov, I. Shugurov, and S. Ilic, "HomebrewedDB: RGB-D dataset for 6D pose estimation of 3D objects," *International Conference on Computer Vision (ICCV) Workshops*, 2019.

[7] M. Wang and W. Deng, "Deep visual domain adaptation: A survey," *Neurocomputing*, vol. 312, pp. 135–153, 2018.

[8] G. J. Stein and N. Roy, "GeneSIS-RT: Generating synthetic images for training secondary real-world tasks," in *International Conference on Robotics and Automation (ICRA)*, 2018, pp. 7151–7158.

[9] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *International Conference on Computer Vision (ICCV)*, 2017.

[10] F. Mueller, F. Bernard, O. Sotnychenko, D. Mehta, S. Sridhar, D. Casas, and C. Theobalt, "GANerated hands for real-time 3D hand tracking from monocular RGB," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 49–59.

[11] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb, "Learning from simulated and unsupervised images through adversarial training," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2242–2251.

[12] K. Bousmalis, N. Silberman, D. Dohan, D. Erhan, and D. Krishnan, "Unsupervised pixel-level domain adaptation with generative adversarial networks," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 95–104.

[13] J. Hoffman, E. Tzeng, T. Park, J.-Y. Zhu, P. Isola, K. Saenko, A. Efros, and T. Darrell, "CyCADA: Cycle-consistent adversarial domain adaptation," in *International Conference on Machine Learning (ICML)*, PMLR, 2018, pp. 1989–1998.

[14] R. Gong, W. Li, Y. Chen, and L. V. Gool, "DLOW: Domain flow for adaptation and generalization," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[15] G. Lin, A. Milan, C. Shen, and I. Reid, "RefineNet: Multi-path refinement networks for high-resolution semantic segmentation," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[16] A. Foi, M. Trimeche, V. Katkovnik, and K. Egiazarian, "Practical Poissonian-Gaussian noise modeling and fitting for single-image raw-data," *IEEE Transactions on Image Processing*, vol. 17, no. 10, pp. 1737–1754, 2008.

[17] L. Van der Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, no. 11, 2008.

[18] M. Wattenberg, F. Viégas, and I. Johnson, "How to use t-SNE effectively," *Distill*, 2016.

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems (NIPS)*, vol. 25, 2012.