On Pre-trained Image Features and Synthetic Images for Deep Learning

Stefan Hinterstoisser[1(✉)], Vincent Lepetit[2(✉)], Paul Wohlhart[1(✉)], and Kurt Konolige[1(✉)]

[1] X, Mountain View 94043, USA
{hinterst,wohlhart,konolige}@google.com
[2] University of Bordeaux, 33405 Bordeaux, France
vincent.lepetit@u-bordeaux.fr

Abstract. Deep Learning methods usually require huge amounts of training data to perform at their full potential, and often require expensive manual labeling. Using synthetic images is therefore very attractive to train object detectors, as the labeling comes for free, and several approaches have been proposed to combine synthetic and real images for training. In this paper, we evaluate if 'freezing' the layers responsible for feature extraction to generic layers pre-trained on real images, and training only the remaining layers with plain OpenGL rendering may allow for training with synthetic images only. Our experiments with very recent deep architectures for object recognition (Faster-RCNN, R-FCN, Mask-RCNN) and image feature extractors (InceptionResnet and Resnet) show this simple approach performs surprisingly well.

1 Introduction

The capability of detecting objects in challenging environments is a key component for many computer vision and robotics task. Current leading object detectors—Faster-RCNNs [2], SSD [3], RFCN [4], Yolo9000 [5]—all rely on convolutional neural networks. However, to perform at their best, they require huge amounts of labeled training data, which is usually time consuming and expensive to create (Fig. 1).

Using synthetic images is therefore very attractive to train object detectors, as the labeling comes for free. Unfortunately, synthetic rendering pipelines are usually unable to reproduce the statistics produced by their real-world counterparts. This is often referred to as the 'domain gap' between synthetic and real data and the transfer from one to another usually results in deteriorated performance, as observed in [6] for example.

Several approaches have tried to overcome this domain gap. For instance, [7–9] use synthetic images in addition to real ones to boost performance. While

Electronic supplementary material The online version of this chapter (https://doi.org/10.1007/978-3-030-11009-3_42) contains supplementary material, which is available to authorized users.

L. Leal-Taixé and S. Roth (Eds.): ECCV 2018 Workshops, LNCS 11129, pp. 682–697, 2019.
https://doi.org/10.1007/978-3-030-11009-3_42

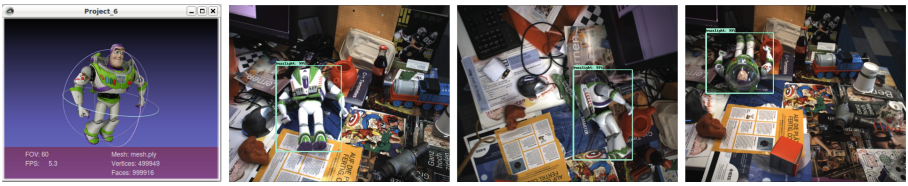


Fig. 1. We show that feature extractor layers from modern object detectors pre-trained on real images can be used on synthetic images to learn to detect objects in real images. The top-left image shows the CAD model we used to learn to detect the object in the three other images.

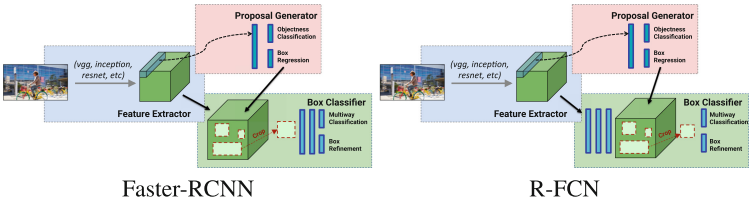


Fig. 2. The architectures of two recent object detectors with their feature extractors isolated as described in [1] (Figure taken from [1]).

this usually results in good performance, it is still dependent on real world labeled data. Transfer learning approaches are also possible [10–12], however they also require real images of the objects to detect. [13,14] create photo-realistic graphics renderings and [8,13–15] compose realistic scenes for improved performance. Unfortunately, these strategies are usually difficult to engineer, need domain specific expertise and require some additional data such as illumination information and scene labeling to create realistic scenes. [6] uses 'domain randomization' to narrow the gap. While this has shown very promising results, it has mainly been demonstrated to work with simple objects and scenarios. Other works [16,17] use Generative Adversarial Networks (GANs) to remove the domain gap, however, GANs are still very brittle and hard to train, and to the best of our knowledge they have not been used for detection tasks yet.

In this paper we consider a simple alternative solution. As shown by [1] and illustrated in Fig. 2, many of today's modern feature extractors can be split into a feature extractor and some remaining layers that depend on the meta-architecture of the detector. Our claim is twofold: (a) the pre-trained feature extractors are already rich enough and do not need to be retrained when considering new objects to detect; (b) when applied to an image synthetically generated using simple rendering techniques, the feature extractors work as a "projector" and output image features that are close to real image features.

Therefore, by freezing the weights of feature extractor pre-trained on real data and by only adapting the weights of the remaining layers during training, we are able to train state-of-the-art object detectors purely on synthetic data.

While using pre-trained layers for feature extraction and finetuning them on a different task is not new (for example, VGG [18] has been used extensively for this purpose, our contribution is to show that this approach also enables us to train on synthetic data only if the pre-trained weights of the feature extractor are frozen. Since we have not found any reference on this particular approach, we evaluated it and report the results here as we thought it could be very useful for the community. We also show that this observation is fairly general and we give both qualitative and quantitative experiments for different detectors—Faster-RCNN [2], RFCN [4] and Mask-RCNN [19]—and different feature extraction networks—InceptionResnet [20] and Resnet101 [21].

Furthermore, we show that different cameras have different image statistics that allow different levels of performance when re-trained on synthetic data. We will demonstrate that performance is significantly boosted for these cameras if this simple approach is applied.

In the remainder of the paper we first discuss related work, describe how we generate synthetic data, demonstrate the domain gap between synthetic and real data, and detail our experiments and conclusions.

2 Related Work

Mixing real and synthetic data to improve detection performance is a well established process. Many approaches such as [7,8,22], to mention only very recent ones, have shown the usefulness of adding synthetic data when real data is limited. In contrast to [7,8] which use real masked image patches, [22] uses 3D CAD models and a structure-preserving deformation pipeline to generate new synthetic models to prevent overfitting. However, while these approaches obtain better results compared to detectors trained on real data only, they still require real data.

In order to avoid expensive labeling in terms of time and money, some approaches learn object detectors purely from synthetic data. For instance, a whole line of work uses photo-realistic rendering [13,14] and complex scene composition [8,13–15] to achieve good results, and [23] stresses the need for photo-realistic rendering. Some approaches even use physics engines to enable realistic placing of objects [24]. This requires significant resources and highly elaborate pipelines that are difficult to engineer and need domain specific expertise [25]. Furthermore, additional effort is needed to collect environment information like illumination information [14] to produce photo-realistic scenes. For real scene composition, one also needs to parse real background images semantically in order to place the objects meaningful into the scene.

This usually needs manual post-processing or labeling which is both expensive and time consuming. While these graphics based rendering approaches already show some of the advantages of learning from synthetic data, they usually suffer from the domain gap between real and synthetic data.

To address this, a new line of work [7–9] moves away from graphics based renderings to composing real images. The underlying theme is to paste masked

patches of objects into real images, and thus reducing the dependence on graphics renderings. This approach has the advantage that the images of the objects are already in the right domain—the domain of real images—and thus, the domain gap between image compositions and real images is smaller than the one of graphics based rendering and real images. While this has shown quite some success, the amount of data is still restricted to the number of images taken from the object in the data gathering step and therefore does not allow to come up with new views of the object. Furthermore, it is not possible to generate new illumination settings or proper occlusions since shape and depth are usually not available. In addition, this approach is dependent on segmenting out the object from the background which is prone to segmentation errors when generating the object masks.

Recently, several approaches [16,17] tried to overcome the domain gap between real and synthetic data by using generative adversarial networks (GANs). This way they produced better results than training with real data. However, GANs are hard to train and up to now, they have mainly shown their usefulness on regression tasks and not on detection applications.

Yet another approach is to rely on transfer learning [10–12], to exploit a large amount of available data in a source domain, here the domain of synthetic images, to correctly classify data from the target domain, here the domain of real images, for which the amount of training data is limited. This is typically done by tighting two predictors together, one trained on the source domain, the other on the target domain or by training a single predictor on the two domains. This is a general approach as the source and target domains can be very different, compared to synthetic and real images, which are more related to each other. In this paper, we exploit this relation by applying the same feature extractor to the two domains. However, in contrast to [10–12] we do not need any real images of the objects of interest in our approach.

As mentioned in the introduction, finetuning pre-trained object detection networks [26] and freezing intermediate level layers during fine-tuning [27] is not new. However, to the best to our knowledge, no paper has shown that these two techniques when combined can help to bridge the domain gap between real and synthetic data and enable state-of-the-art object detectors to be trained only from synthetically rendered data with only little degradation compared to models trained on real data only. For instance, [28] discusses finetuning the hidden layers and is not about freezing layers. Also, it only tackles the classification part of RCNN, as the object proposal component of RCNN is not deep. In addition, its training dataset is not purely synthetic and contains real images, too. [7,9,22] also use fine-tuning but no freezing of layers. In addition, while [22] renders CAD models, [7,9] are only about composing images only.

3 Method

In this section, we will present our simple synthetic data generation pipeline and describe how we change existing state-of-the-art object detectors to enable them

to learn from synthetic data. In this context, we will focus on object instance detection. Throughout this paper, we will mainly consider Faster-RCNN [2] since it demonstrated the best detection performance among a whole family of object detectors as shown in [1]. However, in order to show the generability of our approach, we will also present additional quantitative and qualitative results of other detectors (RFCN [4] and Mask-RCNN [19]) in Sect. 4.7.

3.1 Synthetic Data Generation Pipeline

Similar to [7], we believe that while global consistency can be important, local appearance—so called patch-level realism—is also important. The term patch-level realism refers to the observation that the content of the bounding box framing the rendered object looks realistic.

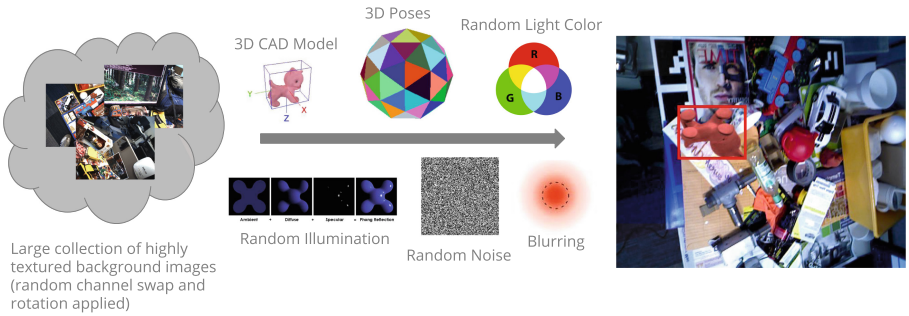


Fig. 3. Our synthetic data generation pipeline. For each generated 3D pose and object, we render the object over a randomly selected cluttered background image using OpenGL and the Phong illumination model [29]. We use randomly perturbed light color for rendering and add image noise to the rendering. Finally, we blur the object with a Gaussian filter. We also compute a tightly fitting bounding box using the object's CAD model and the corresponding pose. (Color figure online)

This principle is an important assumption for our synthetic data generation pipeline, shown in Fig. 3. For each object, we start by generating a large set of poses uniformly covering the pose space in which we want to be able to detect the corresponding object. As in [30], we generate rotations by recursively dividing an icosahedron, the largest convex regular polyhedron. We substitute each triangle into four almost equilateral triangles, and iterate several times. The vertices of the resulting polyhedron give us then the two out-of-plane rotation angles for the sampled pose with respect to the coordinate center. In addition to these two out-of-plane rotations, we also use equally sampled in-plane rotations. Furthermore, we sample the scale logarithmically to guarantee an approximate linear change in pixel coverage of the reprojected object between consecutive scale levels.

The object is rendered at a random location in a randomly selected background image using a uniform distribution. The selected background image is

part of a large collection of highly cluttered real background images taken with the camera of choice where the objects of interest are not included. To increase the variability of the background image set, we randomly swap the three background image channels and randomly flip and rotate the images (0°, 90°, 180° and 270°). We also tried to work without using real background images and experimented with backgrounds only exhibiting one randomly chosen color, however, that did not lead to good results.

We use plain OpenGL with simple Phong shading [29] for rendering where we allow small random perturbations of the ambient, the diffuse and the specular parameters. We also allow small random perturbations of the light color. We add random Gaussian noise to the rendered object and blur it with a Gaussian kernel, including its boundaries with the adjacent background image pixels to better integrate the rendering with the background. We also experimented with different strategies for integrating the rendered object in images as [7], however this did not result in significant performance improvements.

3.2 Freezing a Pre-trained Feature Extractor

As shown in [1] and illustrated in Fig. 2, many state-of-the-art object detectors including Faster-RCNN [2], Mask-RCNN [19], and R-FCN [4] can be decoupled as a 'meta-architecture' and a feature extractor such as VGG [18], Resnet [21], or InceptionResnet [20].

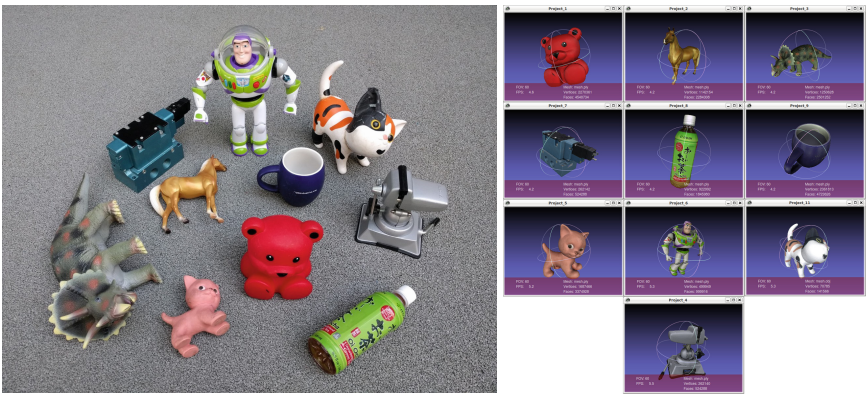
While the meta-architecture defines the different modules and how they work together, the feature extractor is a deep network cut at some selected intermediate convolutional level. The remaining part can be used as part of the multi-way classification+localization of the object detector. As discussed in the introduction, for the feature extractor, we use frozen weights pre-learned on real images, to enable training the remaining part of the architecture on synthetic images only.

In practice, we use the Google's public available OpenSource version [1] of Faster-RCNN and RFCN, and our own implementation of Mask-RCNN. The 'frozen' parts are taken according to [1], by training InceptionResnet and Resnet101 on a classification task on the ImageNet-CLs dataset. We freeze InceptionResnet (v2) after the repeated use of block17 and right before layer Mixed_7a and Resnet101 after block3. All other remaining parts of the networks are not 'frozen', meaning their weights are free to adapt when we train the detector on synthetic images.

We evaluate this approach in the next section.

4 Experiments

In this section, we first describe the dataset we created for these evaluations, made of synthetic and real images of 10 different objects. We also considered two different cameras, as the quality of the camera influences the recognition results as we will show. The rest of the section reports our experiments and the conclusions we draw from them.



(a) Real Objects (b) 3D CAD Models

Fig. 4. (a) The real objects used in our experiments and (b) their CAD models. We chose our objects carefully to represent different colors and 3D shapes and to cover different fields of applications (industrial objects, household objects, toys). (Color figure online)

4.1 Objects and 3D CAD Models

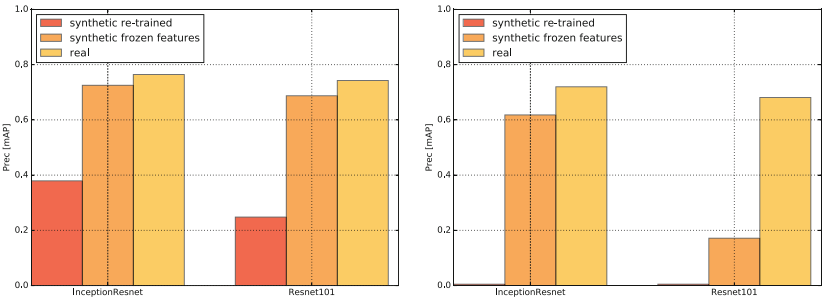
As shown in Fig. 4, we carefully selected the objects we used in our experiments: We tried to represent different colors, textures (homogeneous color versus highly textured), 3D shapes and material properties (reflective versus non-reflective). Except for the mug and the bottle, the 3D shapes of the objects we selected can look very different from different views. We also tried to consider objects from different application fields (industrial objects, household objects, toys). For each real object we have a textured 3D CAD model at hand which we generated using our in-house 3D scanner.

4.2 Cameras

We consider two cameras, an AsusXtionPROLive and a PtGreyBlackfly. For each camera, we generated a training dataset and an evaluation dataset. The training datasets consist of approximatively 20 K and the evaluation datasets of approximatively 1 K manually labeled real world images. Each sample image contains one of the 10 objects shown in Fig. 4 in challenging environments: heavy background clutter, illumination changes, etc. In addition, we made sure that each object is shown from various poses as this is very important for object instance detection. Furthermore, for each dataset all objects have the same amount of images.

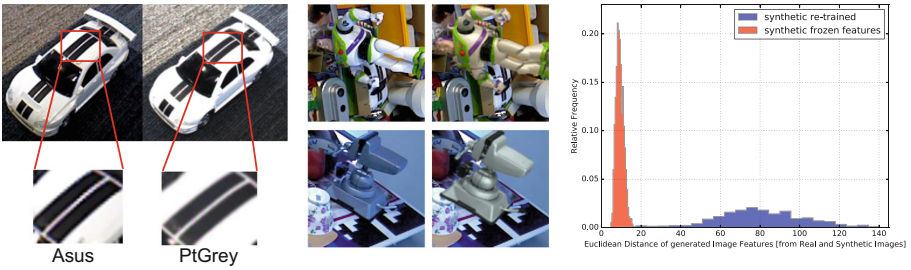
4.3 Freezing the Feature Extractor

Figure 5 shows that when Faster-RCNN is trained on synthetic images and tested on real images, it performs significantly worse than when trained on real data. By



(a) PtGreyBlackfly (b) AsusXtionPROLive

Fig. 5. The effect of freezing the pre-trained feature extractor, for two different cameras. Training the feature extractors on synthetic images performs poorly, and totally fails in the case of the AsusXtionPROLive camera. When using feature extractors pre-trained on real images without retraining them, the performances of detectors trained on synthetic data are almost as good as when training them on real data, except when ResNet101 is used with images from the AsusXtionPROLive camera.



(a) (b) (c)

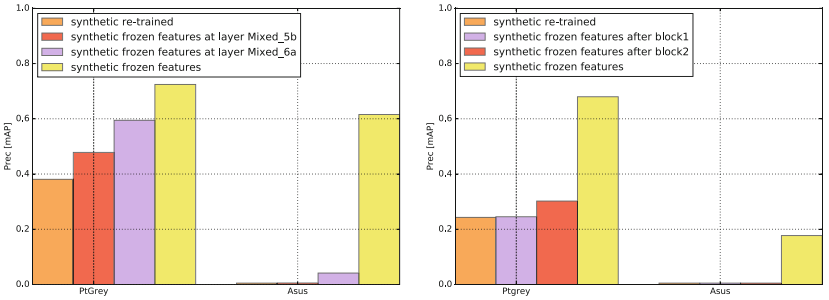
Fig. 6. (a) Debayering Artefacts of the AsusXtionPROLive camera (zoom for better view). (b) Two examples of pairs of a real image and a synthetic one for the same object under the same pose. (c) Distributions of the Euclidean distances between image features generated for the real images and the corresponding synthetic images. See text in Sect. 4.3 for details. (Color figure online)

contrast, when we freeze the feature extractor's weights during training to values pre-trained on real images, and only train the remaining parts of the detector, we get a significant performance boost. We even come close to detectors trained purely on real world data, as we typically obtained up to 95% of the performance when trained on synthetic data.

In general, we observe that our method exhibits better results for the PtGrey-Blackfly camera than for the AsusXtionPROLive camera. In contrast to the PtGrey camera, the Asus camera exhibits 'debayering artefacts' along edges that we do not simulate. We believe that this debayering artefact is the main reason for the differences between the two cameras (see Fig. 6(a)).

To get a better intuition why freezing the feature extractor gives significant better results than retraining it on synthetic data, we performed the following experiment: We created 1000 image pairs with different objects under various poses. Each image pair consists of one image that shows the real object and of another image where we superpose a rendering of the object's CAD model on top of the real image, under the same pose as the real object. Figure 6(b) shows two examples.

We then compared the distributions of the Euclidean distances between image features generated for the real images and the corresponding synthetic images. As we can see Fig. 6(c), the distribution is much more clustered around 0 when the features are computed using a frozen feature extractor pre-trained on real images (red) compared to the distribution obtained when the pre-trained feature extractor is finetuned on synthetic images (blue).



a) InceptionResnet [20] b) Resnet101 [21]

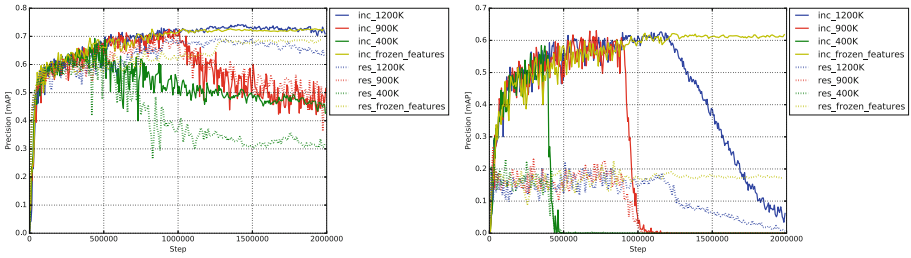
Fig. 7. We freeze features at different layers of InceptionResnet [20] and Resnet101 [21]. We can see that freezing the full feature extractor performs best (yellow). (Color figure online)

4.4 Freezing the Feature Extractor at Different Layers

We also performed experiments where we freeze the feature extractor at different intermediate layers i.e. layers lying between the input and the output layers of the feature extractor as specified in Sect. 3.2. As can be seen in Fig. 7, freezing the full feature extractor always performs best. For the AsusXtionPROLive camera, freezing the feature extractor on intermediate levels even results in a dramatic loss of performance.

4.5 On Finetuning the Feature Extractor

One may wonder if the domain shift between synthetic and real images still leads to decreased performance after the detector was trained for some time with the pre-trained feature extractor frozen. One could argue that all remaining detector



a) PtGreyBlackfly b) AsusXtionPROLive

Fig. 8. Finetuning the feature extractor after 400K, 900K and 1200K steps where the pre-trained feature extractor was frozen for the PtGreyBlackfly and the AsusXtion-PROLive cameras. We show results for the InceptionResnet [20] and Resnet101 [21] architectures.

weights have already started to converge and therefore, the domain shift is far less influential. As a result, the frozen feature extractor could be unfrozen to finetune its weights to adapt to the learning task.

However, as we show in Fig. 8, this is not true. Even after 1200K training steps where the feature extractor was frozen and the detection performance starts to plateau the detector's performance degrades significantly if the frozen feature extractor is unfrozen and its weights are finetuned. Table 1 gives the corresponding numbers.

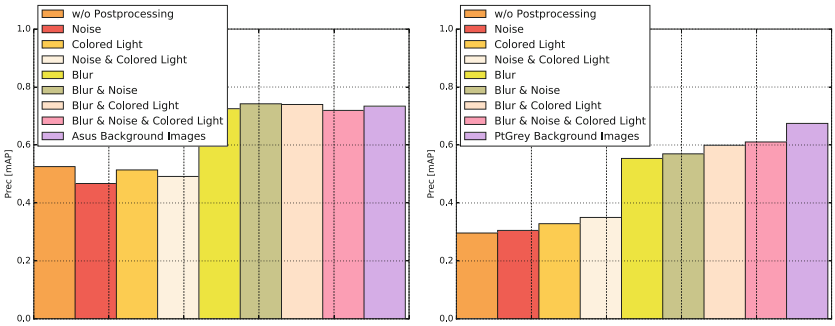
Table 1. Outcomes of all our experiments. We give numbers for InceptionResnet [20]/Resnet101 [21]. Except for the experiments with real data (last column), all experiments were performed on synthetic data only. We emphasized the best results trained on synthetic data.

		Synthetic	Frozen	400K	900K	1200K	Real
Asus	Prec [mAP]	.000/.000	**.617**/.171	.000/.000	.000/.000	.061/.006	.719/.681
	Prec [mAP@0.5]	.000/.000	**.948**/.385	.000/.000	.000/.000	.114/.016	.983/.988
	Prec [mAP@0.75]	.000/.000	**.733**/.130	.000/.000	.000/.000	.064/.004	.872/.844
	Acc [@100]	.000/.010	**.686**/.256	.000/.000	.000/.000	.079/.007	.772/.742
PtGrey	Prec [mAP]	.374/.243	**.725**/.687	.426/.317	.514/.485	.709/.626	.764/.742
	Prec [mAP@0.5]	.537/.410	**.971**/.966	.606/.491	.717/.685	.936/.912	.987/.987
	Prec [mAP@0.75]	.431/.239	**.886**/.844	.495/.355	.593/.564	.835/.756	.908/.916
	Acc [@100]	.461/.324	**.771**/.736	.483/.384	.577/.551	.768/.695	.808/.804

4.6 Ablation Experiments

In the following experiments, we investigated the influence of the single steps in the image generation pipeline. For all these experiments we used InceptionResnet [20] as feature extractor. The feature extractor itself was frozen. We found

out that blurring the rendered object and its adjacent image pixels gives a huge performance boost. Adding noise to the rendered object or enabling random light color did not give much improvement in performance and its influence depends on the camera used. As already mentioned, we also experimented with different blending strategies as in [7], that is using different blending options in the same dataset: no blending, Gaussian blurring and Poisson blending, however we could not find significant performance improvements.



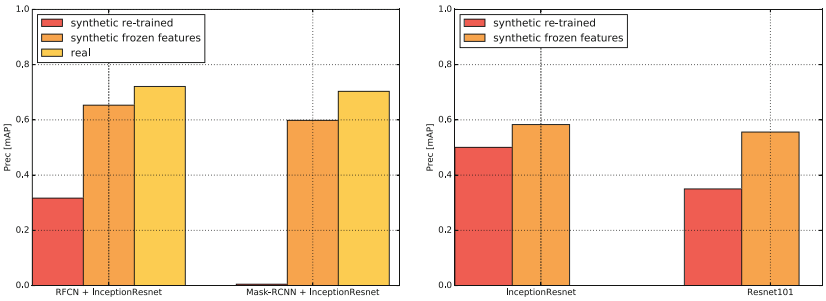
a) PtGreyBlackfly b) AsusXtionPROLive

Fig. 9. Influences of the different building blocks for synthetic rendering for the PtGrey-Blackfly and the AsusXtionPROLive cameras. Results were obtained with Inception-Resnet [20] as the feature extractor. Blurring is clearly a useful yet simple operation to apply to the synthetic images to improve the results.

We also investigated what happens if we use the internal camera parameter of our target camera but a background dataset taken with another camera. As we can see in Fig. 9, results seem to stay approximately the same for the PtGreyBlackfly camera and seem to improve for the AsusXtionPROLive camera. The later seems reasonable since the background images taken with the PtGreyBlackfly camera are more cluttered and are showing more background variety than the background images taken with the AsusXtionPROLive camera. These results suggest that the camera images can be taken from an arbitrary source and we only have to make sure that a high amount of background variety is provided.

4.7 RFCN, MASK-RCNN and the Dishware Dataset

To show the generality of our approach, we also performed several addition experiments. Figure 10(a) shows the results for RFCN [4] trained only on synthetic data with the feature extractor frozen and compares them with those using RFCN trained on real data and and those using RFCN re-trained on synthetic data. Freezing the feature extractor helps to unlock significant performance improvements also here.



a) Results of RFCN and Mask-RCNN b) Results on the Dishware dataset

Fig. 10. Left: Results using RFCN [4] on the PtGreyBlackfly dataset. Freezing the feature extractor boosts performance significantly on this method as well. We observe the same results if we train Mask-RCNN on the AsusXtionPROLive dataset. Right: We also performed experiments on the Dishware dataset using the PtGreyBlackfly camera. Since we have only real evaluation data and no real labeled training data we show the difference between training purley on synthetic data with non-frozen and frozen features. While one can observe a significant gap between the two approaches, the gap is not as large as in previous experiments. We believe that this is because the dataset contains mostly uniform and little textured objects and thus, is less prone to synthetic image statistics generated by rendering.

Figure 10(a) also shows quantitative results of Mask-RCNN [19] trained only on synthetic data with the feature extractor frozen. Similar to what we observed with Faster-RCNN and RFCN, freezing the feature extractor significantly boosts the performance when trained on synthetic data. Figure 13 shows that we are able to detect objects in highly cluttered environments under various poses and get reasonable masks. This result is especially important since it shows that exhaustive (manual) pixel labeling is made redundant by training from synthetic data.

We also show results of Faster-RCNN on another dataset depicted in Fig. 11 that we created with the PtGreyBlackfly camera. We call this dataset the Dishware dataset as it contains 9 dishware objects (*i.e.* plates, cups, bowls) and their corresponding 3D CAD models. For this dataset, in addition to the 3D CAD models of the nine objects, we also have real evaluation data, but no real training data at hand. Therefore, we only show the gap between training with re-trained and frozen features on synthetic training data. The evaluation dataset consists of approximatively 1K manually labeled real world images where objects are seen in different cluttered environments under various poses and severe illumination changes. Each object has the same number of evaluation images. As one can see in Fig. 10(b), freezing the features helps to significantly increase performance. While the gap between these two approaches is significant, it is less than what we observed on our first dataset (see Fig. 5). We believe that this is because the dataset contains mostly uniform or little textured objects and thus, is less prone to synthetic image statistics generated by rendering.

(a) Real Dishware Objects (b) 3D CAD Dishware Models

Fig. 11. (a) The real objects of our second dataset (dishware dataset) used in Fig. 10 and (b) their CAD models.

Fig. 12. Results of Faster-RCNN trained on synthetic images only with the feature extractor frozen. The objects are detected in highly cluttered scenes and many different instances are available in one image. Note that the different objects are seen in different arbitrary poses.

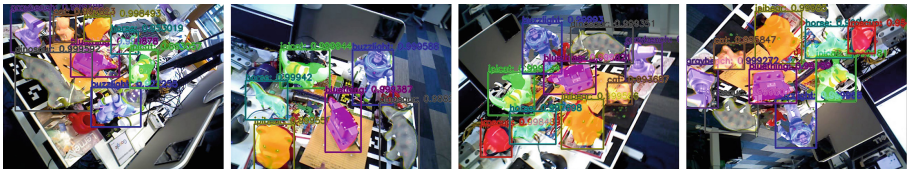


Fig. 13. Results of Mask-RCNN [19] trained on synthetic images only with the feature extractor frozen. The images were taken with the AsusXtionPROLive camera in a highly cluttered environment under various poses.

Fig. 14. Objects with similar shapes and colors detected in challenging environments. The detector was trained on synthetic images only. (Color figure online)

4.8 Qualitative Results

Figure 12 shows some qualitative results on images exhibiting several of the 10 objects we considered in various poses with heavy background clutter and illumination changes. We use Faster-RCNN [2] with the InceptionResnet [20] as feature extractor and trained the rest of the network on synthetic images only. Figure 13 shows results of Mask-RCNN [19] trained on synthetic images only. Figure 14 shows some other objects trained with the method presented in this paper.

5 Conclusion

We have shown that by freezing a pre-trained feature extractor we are able to train state-of-the-art object detectors on synthetic data only. The results are close to approaches trained on real data only. While we have demonstrated that object detectors re-trained on synthetic data lead to poor performances and that images from different cameras lead to different results, freezing the feature extractor always gives a huge performance boost.

Our experiments suggest that simple rendering is sufficient to achieve good performances and that complicated scene composition does not seem necessary. Training from rendered 3D CAD models allows us to detect objects from all possible viewpoints which makes the need for a real data generation and expensive manual labeling pipeline redundant.

Acknowledgments. The authors thank Google's VALE team for tremendous support using the Google Object Detection API, especially Jonathan Huang, Alireza Fathi, Vivek Rathod, and Chen Sun. In addition, we thank Kevin Murphy, Vincent Vanhoucke, and Alexander Toshev for valuable discussions and feedback.

References

1. Huang, J., et al.: Speed and accuracy trade-offs for modern convolutional object detectors. In: Conference on Computer Vision and Pattern Recognition (2017)
2. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: towards real-time object detection with region proposal networks. In: Advances in Neural Information Processing Systems (2015)
3. Liu, W., et al.: SSD: single shot multibox detector. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9905, pp. 21–37. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46448-0_2
4. Dai, J., Li, Y., He, K., Sun, J.: R-FCN: object detection via region-based fully convolutional networks. In: Advances in Neural Information Processing Systems (2016)
5. Redmon, J., Farhadi, A.: YOLO9000: better, faster, stronger. In: Conference on Computer Vision and Pattern Recognition (2017)
6. Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., Abbeel, P.: Domain randomization for transferring deep neural networks from simulation to the real world. In: International Conference on Intelligent Robots and Systems (2017)

7. Dwibedi, D., Misra, I., Hebert, M.: Cut, paste and learn: surprisingly easy synthesis for instance detection. arXiv Preprint (2017)
8. Georgakis, G., Mousavian, A., Berg, A.C., Kosecka, J.: Synthesizing training data for object detection in indoor scenes. In: Robotics: Science and Systems Conference (2017)
9. Rad, M., Lepetit, V.: BB8: a scalable, accurate, robust to partial occlusion method for predicting the 3D poses of challenging objects without using depth. In: International Conference on Computer Vision (2017)
10. Rozantsev, A., Salzmann, M., Fua, P.: Beyond sharing weights for deep domain adaptation. In: Conference on Computer Vision and Pattern Recognition (2017)
11. Bousmalis, K., Trigeorgis, G., Silberman, N., Krishnan, D., Erhan, D.: Domain separation networks. In: Advances in Neural Information Processing Systems (2016)
12. Ganin, Y., et al.: Domain-adversarial training of neural networks. J. Mach. Learn. Res. (2016)
13. Gupta, A., Vedaldi, A., Zisserman, A.: Synthetic data for text localisation in natural images. In: Conference on Computer Vision and Pattern Recognition (2016)
14. Alhaija, H.A., Mustikovela, S.K., Mescheder, L., Geiger, A., Rother, C.: Augmented reality meets deep learning for car instance segmentation in urban scenes. In: British Machine Vision Conference (2017)
15. Varol, G., et al.: Learning from synthetic humans. In: Conference on Computer Vision and Pattern Recognition (2017)
16. Shrivastava, A., Pfister, T., Tuzel, O., Susskind, J., Wang, W., Webb, R.: Learning from simulated and unsupervised images through adversarial training. In: Conference on Computer Vision and Pattern Recognition (2017)
17. Bousmalis, K., Silberman, N., Dohan, D., Erhan, D., Krishnan, D.: Unsupervised pixel-level domain adaptation with generative adversarial networks. In: Conference on Computer Vision and Pattern Recognition (2017)
18. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: International Conference for Learning Representations (2015)
19. He, K., Gkioxari, G., Dollár, P., Girshick, R.: Mask R-CNN. arXiv Preprint (2017)
20. Szegedy, C., Ioffe, S., Vanhoucke, V., Alemi, A.A.: Inception-V4, inception-resnet and the impact of residual connections on learning. In: American Association for Artificial Intelligence Conference (2017)
21. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Conference on Computer Vision and Pattern Recognition (2016)
22. Su, H., Qi, C.R., Li, Y., Guibas, L.J.: Render for CNN: viewpoint estimation in images using CNNs trained with rendered 3D model views. In: ICCV (2015)
23. Movshovitz-attias, Y., Kanade, T., Sheikh, Y.: How useful is photo-realistic rendering for visual learning? In: European Conference on Computer Vision (2016)
24. Mitash, C., Bekris, K.E., Boularias, A.: A self-supervised learning system for object detection using physics simulation and multi-view pose estimation. In: International Conference on Intelligent Robots and Systems (2017)
25. Richter, S.R., Vineet, V., Roth, S., Koltun, V.: Playing for data: ground truth from computer games. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9906, pp. 102–118. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46475-6_7
26. Girshick, R.B., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: Conference on Computer Vision and Pattern Recognition (2014)

27. Ouyang, W., Wang, X., Zhang, C., Yang, X.: Factors in finetuning deep model for object detection with long-tail distribution. In: Conference on Computer Vision and Pattern Recognition (2016)
28. Peng, X., Sun, B., Ali, K., Saenko, K.: Learning deep object detectors from 3D models. In: International Conference on Computer Vision (2015)
29. Phong, B.T.: Illumination for computer generated pictures. Commun. ACM **18**, 311–317 (1975)
30. Hinterstoisser, S., et al.: Model based training, detection and pose estimation of texture-less 3D objects in heavily cluttered scenes. In: Lee, K.M., Matsushita, Y., Rehg, J.M., Hu, Z. (eds.) ACCV 2012. LNCS, vol. 7724, pp. 548–562. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37331-2_42