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

Article · October 2017

CITATIONS
5

READS
214

4 authors, including:

Stefan Hinterstoisser
Google Inc.
19 PUBLICATIONS **982** CITATIONS
SEE PROFILE

Vincent Lepetit
Université Bordeaux 1
199 PUBLICATIONS **11,389** CITATIONS
SEE PROFILE

Paul Wohlhart
Graz University of Technology
29 PUBLICATIONS **1,573** CITATIONS
SEE PROFILE

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

Stefan Hinterstoisser
X
hinterst@google.com

Vincent Lepetit
Universite de Bordeaux
vincent.lepetit@u-bordeaux.fr

Paul Wohlhart
X
wohlhart@google.com

Kurt Konolige
X
konolige@google.com

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 show that a simple trick is sufficient to train very effectively modern object detectors with synthetic images only. In order to train them we initialize the part responsible for feature extraction with generic layers pre-trained on real data. We show that that 'freezing' these layers and train only the remaining layers with plain OpenGL rendering performs surprisingly well.

1. Introduction

The capability of detecting objects and object instances in challenging environments is key a component for many computer vision and robotics task. Current leading object detectors exploit convolutional neural networks [17, 11, 3, 16]. Despite their superior performance w.r.t. hand crafted traditional approaches they still suffer from the huge amount of labeled training data which is usually time consuming and expensive to get.

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 domain gap between synthetic and real data and the transfer from one to another usually results in deteriorated performance as also observed by others [22].

Several approaches have tried to overcome this domain gap. For instance, [15, 4, 5] mix real and synthetic data to boost performance. While this usually results in good performance, it is still dependent on real world labeled data.

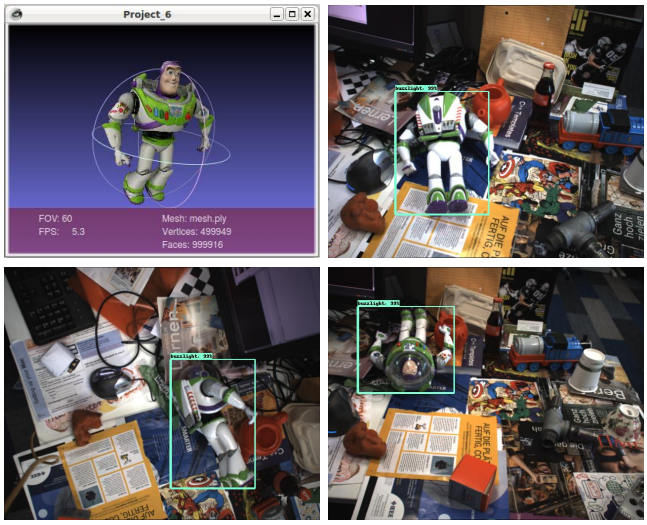
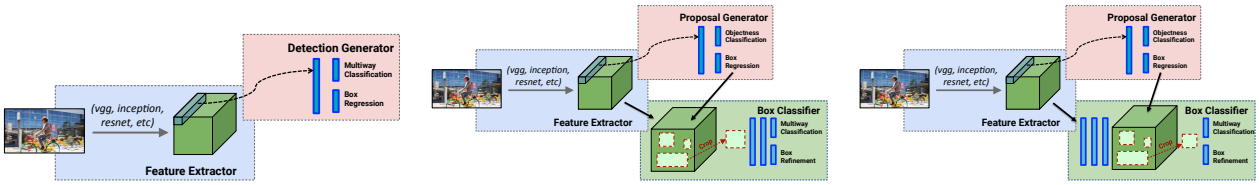


Figure 1. We show that feature extractor layers from modern object detectors pre-trained on real images can be used on synthetic images to learn to detect objects in real images. The top-left image shows the CAD model we used to learn to detect the object in the three other images.

[6, 1] create photo-realistic graphics renderings and [6, 1, 5, 23] compose realistic scenes which both shows to improve performance. Unfortunately, these strategies are usually difficult to engineer, need domain specific expertise and require some additional data (e.g. illumination information and scene labeling) to create realistic scenes.

In contrast, [22] 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.

Some other approaches [19, 2] use Generative Adversarial Networks (GANs) to close the domain gap, however, GANs are still very brittle and hard to train, and to our



a) SSD b) Faster-RCNN c) R-FCN

Figure 2. The architectures of three recent object detectors with their feature extractors isolated as described in [10].

knowledge they haven't been used for detection tasks yet.

In this paper we consider a simple alternative solution. As shown by [10] 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. 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.

We also show in this paper that this observation is fairly general and we will give qualitative and quantitative experiments for different detectors (i.e. Faster-RCNN [17], RFCN [3] and Mask-RCNN [7]) and different feature extraction networks (i.e. InceptionResnet [21] and Resnet101 [8]).

Furthermore, we will show that different cameras have different image statistics that allow different levels of performance when naively trained with synthetic data. We will demonstrate that performance is significantly boosted for these cameras if our simple trick is applied.

In the remainder of the paper we first talk about related work, show how we generate synthetic data, demonstrate the domain gap between synthetic and real data and show how to boost object detection by freezing the pre-trained feature extractor during training. Finally, we show qualitative and quantitative experiments.

2. Related Work

Mixing real and synthetic data in order to improve detection performance is a well established process. Many approaches such as [4, 5], to mention only very recent ones, have shown the usefulness of adding synthetic data when real data is limited. However, while results outperform detectors trained on real data only, these approaches still need real data and therefore are dependent on expensive data labeling.

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 [1, 6] and complex scene composition [6, 1, 5, 23] in order to achieve good results, and [13] stresses the need for photo-realistic rendering. Some approaches even use physics engines to enable realistic placing of objects [12]. This requires significant resources and highly elaborate pipelines that are difficult to engineer and need domain specific expertise [18]. Furthermore, additional effort is needed to collect environment information like illumination information [1] 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 [4, 5, 15] 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 i.e. 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 [19, 2] 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 showed their usefulness on regression tasks and not

on detection applications.

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 rely on Faster-RCNN [17] since it demonstrated best detection performance among a whole family of object detectors as shown in [10]. However, in order to show the generability of our approach, we will also present additional quantitative and qualitative results of other detectors (i.e. RFCN [3] and Mask-RCNN [7]) at the end of this paper (see Sec. 4.4).

3.1. Synthetic Data Generation Pipeline

In this section, we detail our synthetic data generation pipeline, shown in Fig. 3. Similar to [4], we believe that while global consistency can be important, local appearance – so called patch-level realism – can bring us a long way. The term patch-level realism refers to the observation that the content of the bounding box framing the rendered object looks realistic. This principle is an important assumption for our synthetic data generation pipeline.

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 [9], 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 scale logarithmically to guarantee an approximate linear change in pixel coverage of the reprojected object between consecutive scale levels. Translations parallel to the camera plane are chosen randomly guaranteeing a uniform distribution of object appearances in the image. Having the camera's internal camera parameters and a pose allows us to transform the object and project it into a randomly selected background image using OpenGL.

The selected background image is part of a large collection of highly cluttered real background images taken with the camera of choice where we made sure that the objects of interest are not included.

In order to increase the variability of the background image set, we randomly swap the three background image channels. In addition, we randomly flip and rotate the image (0°, 90°, 180° and 270°). We use simple Phong shading [14] 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 in order to better integrate the rendered object with its background.

In the course of this work, we also experimented with different blending strategies as [4], however this did not result in significant performance improvements. Since we know the transformation and the internal camera parameters, we can compute a tight 2D bounding box around the object which serves as label for training.

	AsusXtionPROLive		PtGreyBlackfly	
	synthetic	real	synthetic	real
Prec [mAP]	.000/.000	.719/.681	.374/.243	.764/.742
Acc [@100]	.000/.010	.772/.742	.461/.324	.808/.804

Table 1. Instance Detection performance (mAP) of Faster-RCNN [17] naively trained on synthetic and evaluated on real data for two different feature extractor networks (InceptionResnet [21] and Resnet101 [8]). To test the influence of the camera, we created two synthetic datasets as described in Sec. 3.1: one with a PtGreyBlackfly camera and one with the AsusXtionPROLive . Each dataset uses the internal camera parameters and the background images specific to the camera used. Details to the train and test datasets can be found in Section 4. Training on synthetic data and evaluating on real data results in low detection performance in case of the PtGreyBlackfly camera and the total absence of any reasonable detection result in case of the AsusXtionPROLive camera.

3.2. Naive Training on Synthetic Data

Tab. 1 shows the effect of the domain shift between real and synthetic data: Faster-RCNN trained on synthetic data (generated as described in Sec. 3.1) and tested on real data performs significantly worse than when trained on real data. This observation is independent of the camera and of the feature extraction network used. In the remainder of the work this strategy is referred to as *"naive training"*. Details about the train and test datasets can be found in Section 4.

3.3. Enabling Training from Synthetic Data

As shown in [10] and illustrated in Fig. 2, most state-of-the-art object detectors including SSD [11], Faster-RCNN [17], and R-FCN [3] can be decoupled as a 'meta-architecture' and a feature extractor such as VGG [20], Resnet [8], or InceptionResnet [21].

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 of the object detector. As shown in [10], this classification network can be pretrained on real

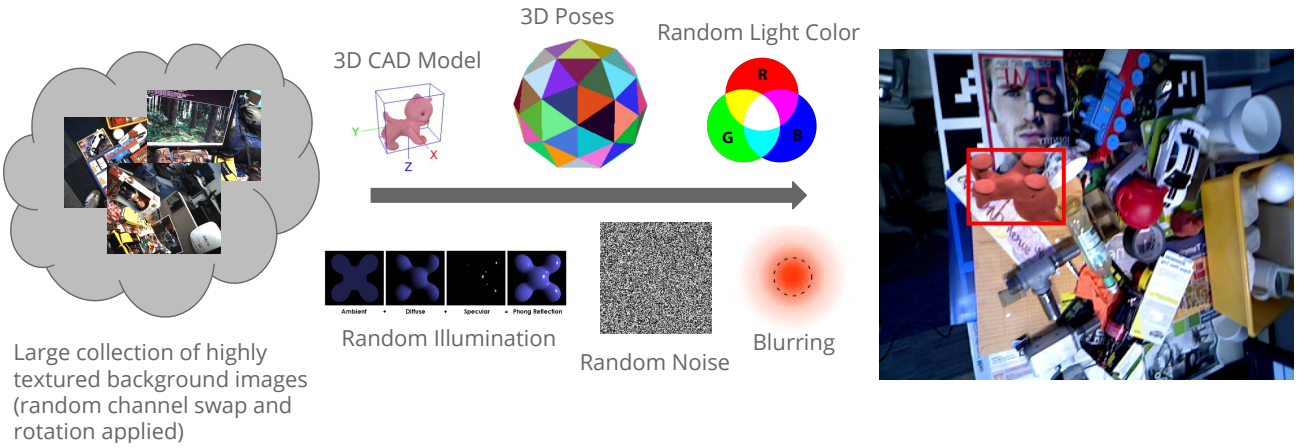


Figure 3. Our synthetic data generation pipeline. For each generated pose and object, we project this object on a randomly selected cluttered background image using OpenGL and the Phong illumination model [14]. We use randomly perturbed light color for rendering and add noise to the rendering. Finally, 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 3D pose.

world data—in our case on ImageNet-CLS—and the trained weights can then initialize the corresponding parts in the object detector.

This novel view leads to two claims founding the base of this work: 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.

From a) follows that the weights of the feature extractor don't need further adaption to give reasonable good performance, while b) ensures that the remaining layers of the detector are fed with features close to real image features which leads to performances similar to the one expected with real features.

If we combine these two assumptions, there is only one simple thing left to enable training from synthetic data: since the pre-trained feature extractor can only produce an approximate projection from the manifold of synthetic onto the manifold of real images features, a (small) fraction of the generated features is of synthetic nature. Therefore, we need to freeze the feature extractor's weights during training and only train the remaining parts of the detector in order to avoid to adapt the feature extractor to the synthetic domain. Thus, we can keep up the real world expressiveness of the feature extractor and continue to serve quasi real image features to the remaining layers of the detector.

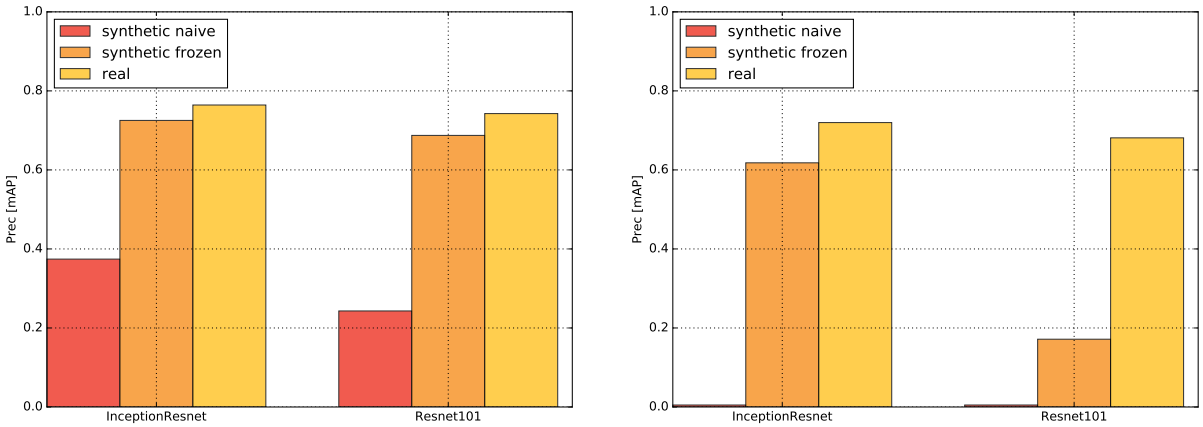
This allows to train the object detector purely on synthetic data and, as shown in Fig. 4, this gives a significant performance boost compared to when the feature extractor is also trained on synthetic images. We even come close to detectors trained purely on real world data, as we typically obtained up to 95% of the performance when trained on synthetic data.

4. Additional Experiments

Before we continue showing additional experiments we give details of our real world training and evaluation datasets. For each camera (AsusXtionPROLive and PtGrey-Blackfly) we generated one training and one evaluation dataset. The training datasets consist of approx 20K and the evaluation datasets of approx 1K manually labeled real world images. Each sample image contains one of the 10 objects shown in Fig. 5 in front of difficult environments (i.e. 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. We use Google's object detection API [10] for our experiments.

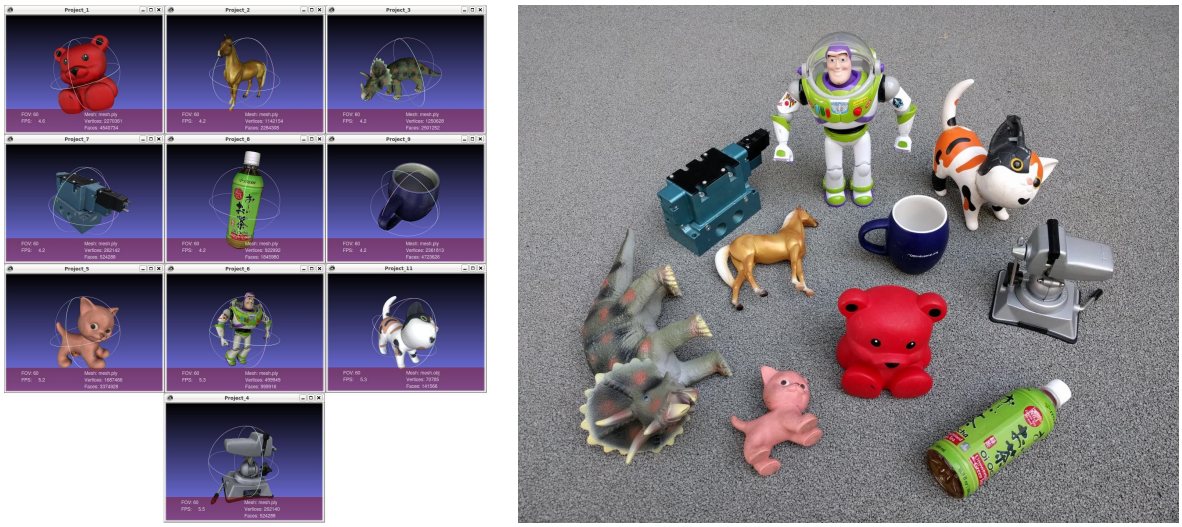
4.1. Objects and 3D CAD models

The objects we deal with in this paper are carefully selected for object instance detection: we try to represent different colors, 3D shapes, textures (i.e. homogeneous color vs. highly textured) and material properties (i.e. reflective vs. non-reflective). Except the mug and the bottle, the 3D shapes of the objects we selected show high variance across views i.e. they look very different from different aspects. We also try to represent objects from different application fields (toys, industrial objects, household objects). For each real object we have a textured 3D Cad model at hand which we generated using our in-house 3D scanner. The objects and the corresponding 3D Cad models are shown in Fig. 5.



a) PtGreyBlackfly b) AsusXtionPROLive

Figure 4. We show that freezing the pre-trained feature extractor helps to significantly boost performance of detectors purely trained on synthetic data. Thus, performance is close to detectors trained purely on real data. In case of the AsusXtionPROLive camera, freezing the pre-trained feature extractor even unlocks training on synthetic data only. InceptionResnet [21] in general seems to perform better than Resnet101 [8] on synthetic data.



a) 3D Cad Models b) Real Objects

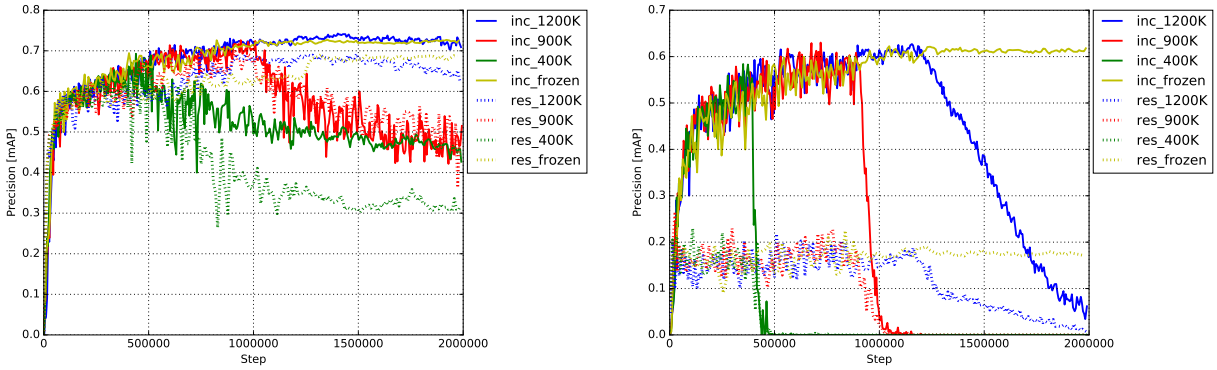
Figure 5. We show a) the object's 3D Cad Models and the b) the real object instances. We chose our objects carefully to represent different colors and 3D shapes and to cover different fields of applications (toys, industrial objects, household objects).

4.2. On Finetuning the Feature Extractor

One of the remaining questions is if the domain shift between synthetic and real data 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 detector weights have already started to converge and therefore, the domain shift does not matter any more. As a result the frozen feature extractor could be unfrozen in order to allow its weights to adapt to the learning task. However, as we show in Fig. 6, this is not true. Even 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. Table 2 gives the corresponding numbers.

4.3. Influence of Individual Steps in the Data Generation Pipeline

In the following experiment, we investigated the influence of the single steps in the data generation pipeline. For all these experiments we used InceptionResnet [21] as fea-



a) PtGreyBlackfly b) AsusXtionPROLive

Figure 6. Unfreezing the feature extractor after 400K, 900K and 1200K steps for the PtGreyBlackfly and the AsusXtionPROLive cameras. We show results for the InceptionResnet [21] and Resnet101 [8] architectures.

		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

Table 2. We show the outcome of all our experiments. We give numbers for InceptionResnet [21] / Resnet101 [8]. Except the experiments with real data (last column), all other experiments were performed purely on synthetic data. We emphasized the best results trained on synthetic data.

ture 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 imager used. We also experimented with different blending strategies as in [4] (i.e. using different blending options in the same dataset: No blending, Gaussian Blurring and Poisson blending), however we could not find significant performance improvements.

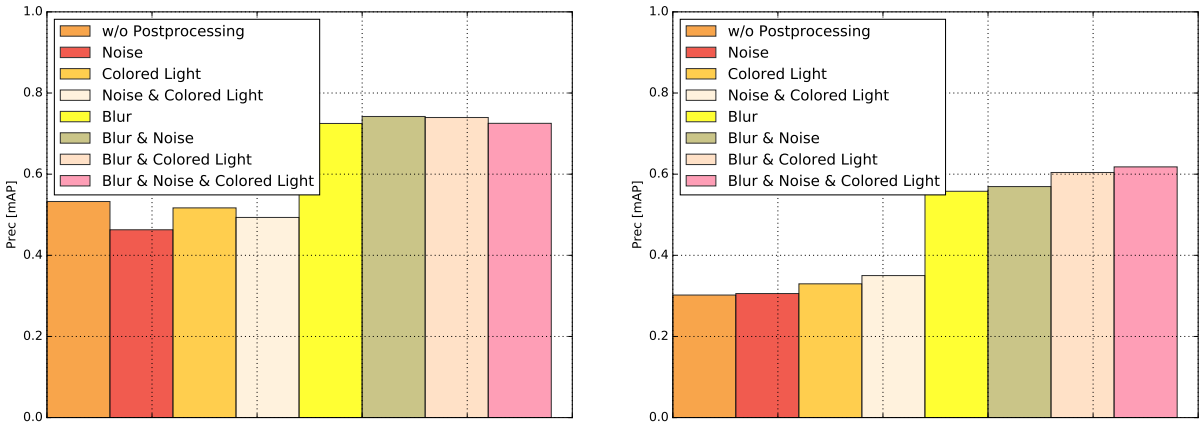
4.4. RFCN and MASK-RCNN

In order to show the generality of our approach, we performed some addition experiments. In Tab. 8, we show RFCN [3] trained purely on synthetic data with the feature extractor frozen and compare it to RFCN trained on real data and to RFCN naively trained on synthetic data. As we can see, freezing the feature extractor helps to unlock significant performance improvements.

In Fig. 11, we exhibit qualitative results of Mask-RCNN trained purely on synthetic data with the feature extractor frozen. As we can see 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. Training Mask-RCNN naively on synthetic data results in very bad performance similar as it is the case for Faster-RCNN and RFCN.

4.5. Qualitative Results

In Fig. 9 we show some of the evaluation images with the detected bounding boxes and classification result. As feature extractor we use a InceptionResnet [21] and we purely trained on synthetic data generated as explained in Sec. 3.1. It shows the 10 objects we trained on (see Fig. 4.1) in various poses with heavy background clutter and illumination changes. In Fig. 10, we show some other exemplary images with multiple objects in the image under heavy background clutter. In Fig. 11, we exhibit results on Mask-RCNN trained purely on synthetic images.



a) PtGreyBlackfly b) AsusXtionPROLive

Figure 7. Influences of the different building blocks for synthetic rendering. Results are obtained with InceptionResnet [21] as the feature extractor.

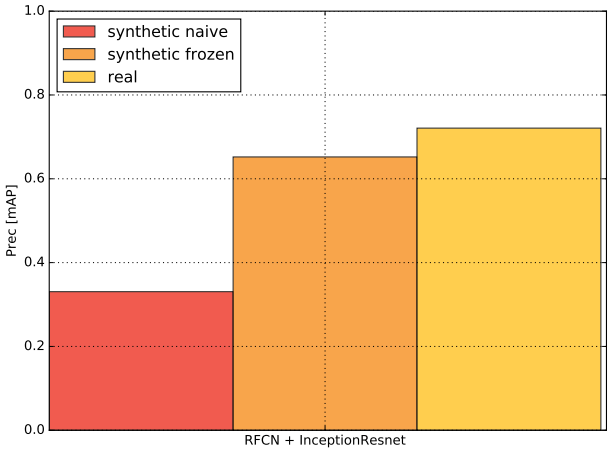


Figure 8. Our simply trick also enables other detectors to train from synthetic data: here we show results of RFCN [3]. We see that freezing the feature extractor boosts performance significantly.

5. Conclusion

In this paper we have shown that by freezing a pretrained feature extractor we are able to train state-of-the-art object detectors purely on synthetic data. The results are close to approaches trained on real data only. While we have demonstrated that object detectors trained naively 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.

The results of our experiments suggest that simple OpenGL 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, Konstantinos Bousmalis and Alexander Toshev for valuable discussions and feedback.

References

[1] H. A. Alhaija, S. K. Mustikovela, L. Mescheder, A. Geiger, and C. Rother. Augmented reality meets deep learning for car instance segmentation in urban scenes. In *British Machine Vision Conference*, 2017. 1, 2

[2] 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*, 2017. 1, 2

[3] J. Dai, Y. Li, K. He, and J. Sun. R-FCN: object detection via region-based fully convolutional networks. In *Advances in Neural Information Processing Systems*, 2016. 1, 2, 3, 6, 7

[4] D. Dwibedi, I. Misra, and M. Hebert. Cut, paste and learn: Surprisingly easy synthesis for instance detection. In *ArXiv*, 2017. 1, 2, 3, 6

[5] G. Georgakis, A. Mousavian, A. C. Berg, and J. Kosecka. Synthesizing training data for object detection in indoor scenes. In *Robotics: Science and Systems Conference*, 2017. 1, 2

[6] A. Gupta, A. Vedaldi, and A. Zisserman. Synthetic data for text localisation in natural images. In *Conference on Computer Vision and Pattern Recognition*, 2016. 1, 2

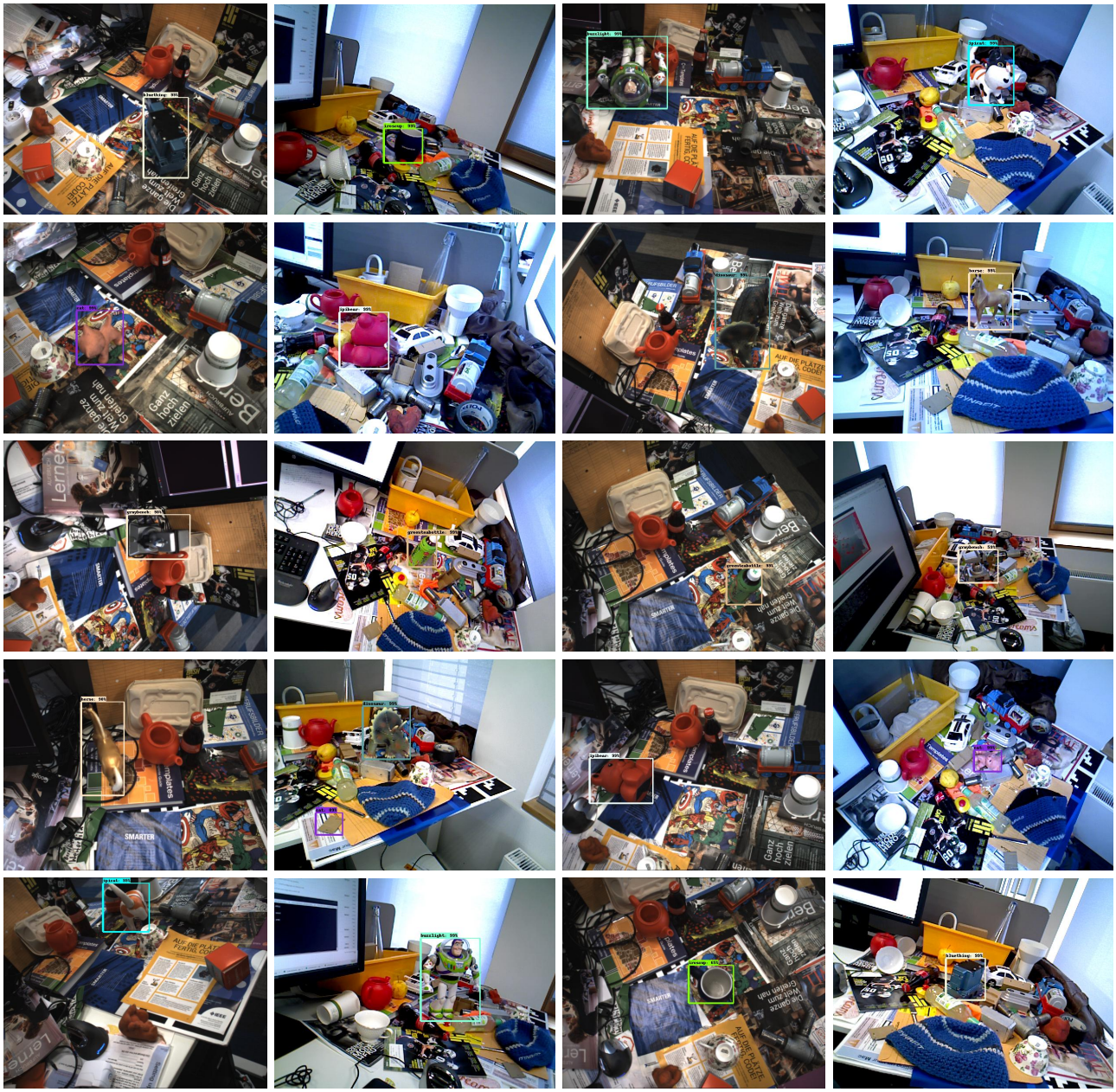


Figure 9. We show results of Faster-RCNN trained purely on synthetic data with the feature extractor frozen in highly cluttered scenes. Note that the different objects are seen in different arbitrary poses. The images are part of the evaluation datasets.

[7] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask R-CNN. In *arXiv preprint arXiv:1703.06870*, 2017. 2, 3, 9

[8] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition*, 2016. 2, 3, 5, 6

[9] 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*, 2012. 3

[10] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy. Speed and accuracy trade-offs for modern convolutional object detectors. In *Conference on Computer Vision and Pattern Recognition*, 2017. 2, 3, 4

[11] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg. SSD: single shot multibox detector. In *European Conference on Computer Vision*, 2016. 1, 3

[12] C. Mitash, K. E. Bekris, and A. Boularias. A self-supervised

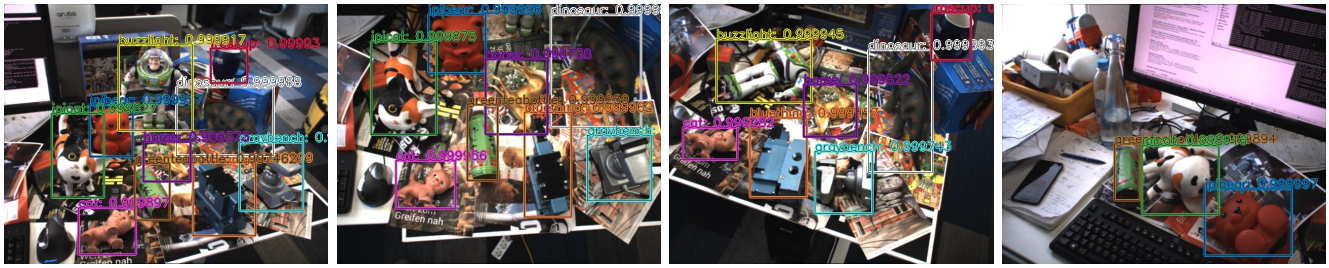


Figure 10. We show results of Faster-RCNN trained purely on synthetic data with the feature extractor frozen. The objects are detected in highly cluttered scenes and many different instances are available in one image. Note that the different objects are seen in different arbitrary poses.

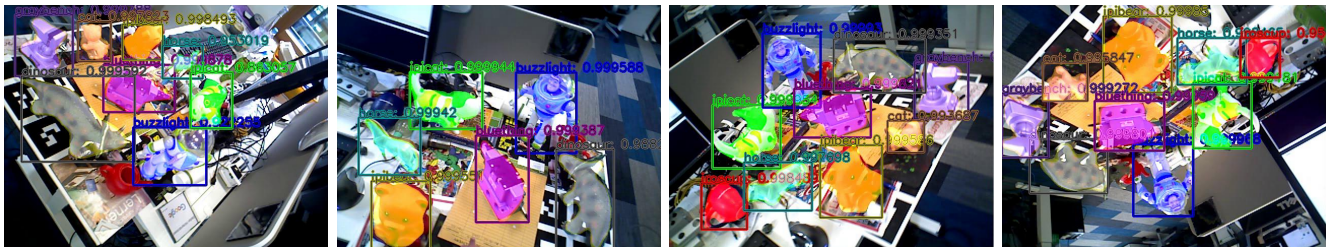


Figure 11. Our simply trick also enables other detectors to train from synthetic data: here we show results of Mask-RCNN [7] trained on synthetic data with the feature extractor frozen. The images were taken with the AsusXtionPROLive camera in a highly cluttered environment under various poses.

learning system for object detection using physics simulation and multi-view pose estimation. In *International Conference on Intelligent Robots and Systems*, 2017. 2

[13] Y. Movshovitz-Attias, T. Kanade, and Y. Sheikh. How useful is photo-realistic rendering for visual learning? In *GMDL 2016 (in conjunction with ECCV 2016)*, 2016. 2

[14] B. T. Phong. Illumination for computer generated pictures. In *Communications of the ACM*, 1975. 3, 4

[15] 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 *International Conference on Computer Vision*, 2017. 1, 2

[16] J. Redmon and A. Farhadi. Yolo9000: Better, faster, stronger. In *Conference on Computer Vision and Pattern Recognition*, 2017. 1

[17] 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*. 2015. 1, 2, 3

[18] S. R. Richter, V. Vineet, S. Roth, and V. Koltun. Playing for data: Ground truth from computer games. In *European Conference on Computer Vision*, 2016. 2

[19] 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*, 2017. 1, 2

[20] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference for Learning Representations*, 2015. 3

[21] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *American Association for Artificial Intelligence Conference*, 2017. 2, 3, 5, 6, 7

[22] 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 *International Conference on Intelligent Robots and Systems*, 2017. 1

[23] G. Varol, J. Romero, X. Martin, N. Mahmood, M. J. Black, I. Laptev, and C. Schmid. Learning from synthetic humans. In *Conference on Computer Vision and Pattern Recognition*, 2017. 1, 2