On Rendering Synthetic Images for Training an Object Detector

Artem Rozantsev (artem.rozantsev@epfl.ch)

Vincent Lepetit (vincent.lepetit@epfl.ch)

Pascal Fua (pascal.fua@epfl.ch)

School of Computer and Communication Sciences Swiss Federal Institute of Technology, Lausanne (EPFL)

Technical Report

June 16, 2014

Abstract

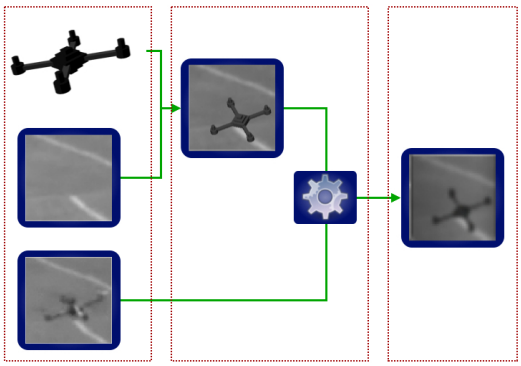
We propose a novel approach to synthesizing images that are effective for training object detectors. Starting from a small set of real images, our algorithm estimates the rendering parameters required to synthesize similar images given a coarse 3D model of the target object. These parameters can then be reused to generate an unlimited number of training images of the object of interest in arbitrary 3D poses, which can then be used to increase classification performances.

A key insight of our approach is that the synthetically generated images should be similar to real images, not in terms of image quality, but rather in terms of features used during the classifier training. We demonstrate the benefits of using such synthetically generated images in the context of drone detection, where limited amount of training data is available.

1 Introduction

It has recently been shown that when enough training data is available, even relatively simple statistical approaches can address image classification problems [14] very effectively. In the commercial world, this is a key ingredient of high performing face detection software deployed by companies such as Apple and Google. However, there are real-world scenarios in which the required training data is hard to obtain in sufficiently large quantities. For example, our work is motivated by the emerging need for Unmanned Aerial Vehicles (UAVs), or *drones*, to see and avoid each other as they become increasingly numerous and autonomous in the sky. In this application, training videos are rare and do not cover the full range of possible shapes, poses, and lighting conditions under which they can be seen.

Our goal therefore is to supplement a small number of available real training samples, with an arbitrary large dataset of synthetic ones in order to improve the detection accuracy of a final classifier. Synthetic data has long been used for Optical Character Recognition purposes [15, 25]. More recently, it has been spectacularly successful for 3D body pose estimation purposes in conjunction with the data produced by depth camera [22]. However, 2D images of characters and depth-images do not usually include lighting, motion blur and other artifacts that affect images from ordinary video-sequences, making representative synthetic image sets even more necessary. This issue was partly addressed for human detection and pose estimation purposes in [19, 20] by providing ways to modify existing images to simulate some of these effects. However, [19] does not model



(a) (b) (c)

Figure 1: Synthetic data generation pipeline. (a) Input to the system includes a simple model of the object of interest, an image of this object and a background image (it should be the same background as the image of the object has). (b) Overlaying the model on the background yields a synthetic image. This image is then processed to maximize its similarity to a real image from the perspective of the detector. (c) The resulting synthetic image can be used for training the detector.

image-acquisition artifacts while [20] involves considerable amounts of manual interaction, which is less desirable.

Furthermore, to the best of our knowledge none of these approaches offers a principled way to choose the image synthesis parameters to match the behavior of real-world cameras in the presence of noise. The relevant parameters are typically tuned by hand, which quickly becomes unmanageable when the rendering pipeline is complex. To overcome this limitation, we therefore introduce a fully automated and generic method to estimate these parameters from a small set of available real images to maximize the performance of a classifier trained using the resulting synthetic images.

To this end, we start from a small set of real seed images containing a target object and corresponding background images without it, such as the ones depicted by Fig. 1(a). Given a very coarse 3D model of the object of interest, such as that of the drone of Fig. 1(a), we estimate the 3D pose of the object, overlaid onto the background image, and then post-process the resulting composite image so that it is as similar as possible to the real one. This is achieved by automated selection of the post-processing parameters to maximize a similarity between the two images. Once these parameters are found, we can then change the position

and the orientation of the object in the image to generate arbitrary large synthetic datasets with realistic imaging artifacts.

A key ingredient of our approach is the similarity function used to measure the difference between real and composite images. An obvious candidate would be the pixel-wise Euclidean distance. However, our goal is not generating pleasantly looking synthetic data, but rather synthetic images that are effective for training purposes. In our experiments we show that the similarity between images *should depend on the image features* used by the detection method. We demonstrate this for three broadly used methods representative of the state-of-the-art: The Deformable Part Model (DPM) method [6], an AdaBoost-based detector [8], and a Convolutional Neural Network (CNN) [21].

In summary, our contribution is a novel and fully automated approach to generating synthetic training image databases that increases detection performance, irrespective of the specific classifier used. We will demonstrate this in the context of our chosen field of application, that is, drone detection, which is challenging because drones tend to be small, hard to see and come in many different shapes.

In the remainder of this paper, we first discuss the effects we want to model in our synthetic images. We then describe and compare the different similarity functions to quantify the similarity between synthetic and real images. Finally, we demonstrate the power of our approach on aircrafts of very different shapes and flying in various environments and lighting conditions.

2 Related Work

Given the prevalence of Machine Learning based algorithms, capturing and annotating training images has become a major issue, and sometimes a severe bottleneck when such images are hard to acquire. In such cases, using Computer Graphics techniques to generate them is a very attractive alternative.

For example, Optical Character Recognition systems have long been trained using samples created by applying various deformations and adding image noise to actual samples [15, 25]. Similarly, synthetically generated image patches have been successfully used in [16, 4] for feature detection. Note, however, that neither characters nor patches exhibit the full complexity of natural images and are therefore easier to synthesize. In [22], this approach was used on complete depth images generated from 3D models of people to train classifiers to recover human 3D pose from the output of a Kinect camera. This has been remarkably successful, in large part because it provides a way to create arbitrarily large training dataset.

However, depth images also lack many of the imaging artefacts present in ordinary images, such as motion blur or lighting effects, which make it difficult to use such an approach for video imagery.

This was attempted in [19] by generating images of pedestrians in various poses and environments to train a pedestrian detector. The results are encouraging but the method does not take complex imaging artefacts into account. More recently, an approach to creating more realistic synthetic images by extracting people's silhouettes from real images, and superimposing them over various backgrounds was proposed [20]. However, it is very specific to pedestrian detection and requires a considerable amount of manual annotation.

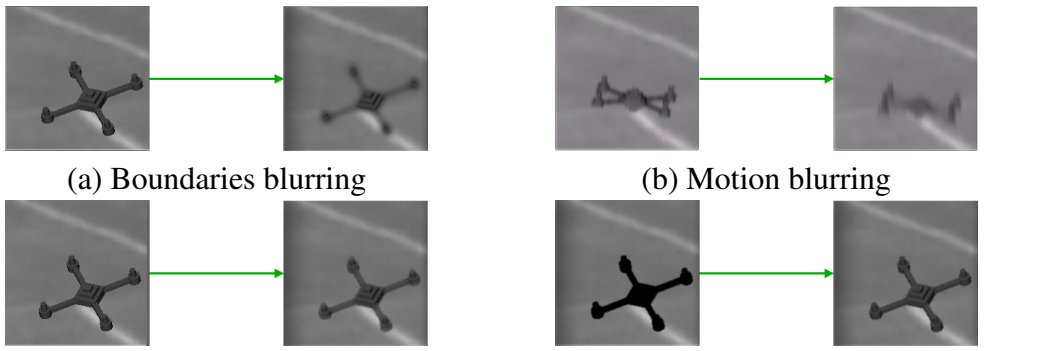
Of course, generic image synthesis techniques have been used in computer vision for many other purposes, such as face detection [7], optimizing camera tracking algorithms [9], evaluation of the algorithms [10, 3, 23, 18, 2, 11], gesture recognition and pose estimation [1, 24], or rendering virtual objects that merge well with real images [13]. Some of these approaches simply project the 3D model of the object of interest on an arbitrary background image. Others add post-processing on similarly generated synthetic images in order to make them look realistic. However, to the best of our knowledge none of them estimate neither how realistic the resulting images are, nor how suitable they are for the application itself.

In this work, we will use some of the same approaches to synthesizing realistic images. This being said, that visual realism is not our end goal, but rather the classification performance improvement. As such our algorithm, unlike the others, automatically optimizes the rendering parameters solely for this purpose.

3 Generating Synthetic Images

As illustrated by Fig. 1, while our pipeline is simple, it depends on many parameters that would be hard to choose by hand. We use basic coarse CAD models, such as that of Fig. 1(a), which roughly captures the target object geometry. We assume that we are given a small set of real images featuring the target object and a corresponding set of background images without it. For each real image, we compute 5 *pose* parameters, that include 3 orientations $(\alpha^p, \beta^p, \gamma^p)$ and 2 translations(t_x^p, t_y^p), which, lets us project the 3D model at the desired location. Note that as we use multi-scale detector, we do not need to vary the scale of the object.

As shown in Fig. 2, we then post-process the synthetic image to maximize its similarity to the real image. This involves:



(a) Boundaries blurring (b) Motion blurring

(c) Random noise (d) Diffuse coefficient of material variation

Figure 2: The four post-processing effects we use for increasing the similarity between the real and synthetic images.

- *Object boundary blurring* **(BB)**. This introduces artefacts due to the discretization of the image sensor, which produces a mixture of the intensities of the background and the target object along its boundaries. To simulate this effect we apply Gaussian blurring along the object boundaries after the object image has been overlaid on the background image. This is controlled by the standard deviation σ^s of the Gaussian kernel used for smoothing.

- *Motion blurring* **(MB)**. This mimics the blurring effect that affects on fast moving objects if the shutter time of the camera is too long. To simulate this effect we use anisotropic Gaussian blurring applied to the pixels of the object in the direction of its motion. The parameters are the two standard deviations σ_u^m and σ_v^m of the Gaussian kernel and the angle α^m of the motion.

- *Random noise* **(RN)**. This emulates the shot noise added to the image by the camera. To simulate this effect we simply add independent Gaussian noise to the pixel intensities. Note this is limited to the image pixels that correspond to the inserted object, as the background images are real ones and already contain similar noise. This is controlled by the standard deviation σ^n of the Gaussian distribution used to generate the noise.

- *Material properties* **(MP)**. We also vary the material properties, by changing the weight w^d of the diffuse reflection. This allows us not only to vary the color of the object, but also to introduce some diffuse lighting effects.

While we do not take specularities into account, this would be a very natural extension to our approach.

We refer to these synthetic data generation parameters as *capture* parameters

$$\Theta = [\underbrace{\alpha^p, \beta^p, \gamma^p, t_x^p, t_y^p}_{\text{pose}}, \underbrace{\sigma^s, \sigma_u^m, \sigma_v^m, \alpha^m, \sigma^n, w^d}_{\text{capture}}]^\top \tag{1}$$

These parameters are challenging to tune, because they are correlated. This is the case for the object pose and the direction of motion blur, as well as for the amount of boundary blurring and motion blurring. Thus our goal is to estimate Θ parameters for every seed real image that we use for synthetic data generation. Given the background images and the corresponding Θ parameters, we retain the capture parameters and randomize the pose ones to generate arbitrary large numbers of synthetic images that will be realistic enough to be used for training the object detector. We explain below how we recover these Θ parameters.

4 Optimizing the Rendering Parameters

In order to optimize the pose and capture parameters in Θ, we rely on a small set of real images of the target object, together with the corresponding images of the background without the target.

Starting from a background image on which we render the CAD model of the target object, we optimize the rendering parameters to reproduce the corresponding real image. This optimization is performed on each image independently, because the same capture parameters do not necessarily apply to all of them. More formally, we consider the set of pairs of real images $\{(X_i, B_i)\}_{i=0}^N$, where $X_i \in \chi$ is the i^{th} image of the object and $B_i \in \chi$ is the background image for X_i. Let $d : \chi \times \chi \rightarrow \mathbf{R}^+$ be a similarity function, which we use to compare two images, and which we will define explicitly in Sect. 4.1. Lastly, let $S(\Theta, B_i) \in \chi$ represent the synthetically rendered image by applying the synthetic data generation process with parameters Θ to the Background B_i

To determine which set of parameters Θ best matches for the real image X_i, we look for

$$\Theta^{(i)} = \underset{\Theta}{\text{argmin}}\, d(X_i, S(\Theta, B_i)) \,, \tag{2}$$

by Simulated Annealing [12]. In practice, we initialize the pose parameters by manually providing the object center, which could be avoided with a more sophis-

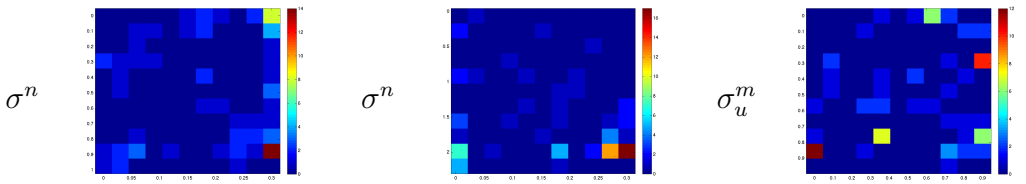


Figure 3: Each histogram depicts the joint distribution of different capture parameters. (best seen in color)

ticated optimization algorithm. Capture parameters are initialized randomly. This optimisation takes a few seconds on each of our 40×40 images.

The capture parameters in Θ depend on viewing conditions, such as lighting and weather conditions, which is why we perform the optimization in each image independently. Fig. 3 describes their distributions across images. Note that these distributions are absolutely not Gaussian and that it would therefore be non-trivial to describe them analytically. Full set of distributions between different pairs of parameters is presented in the supplementary materials.

4.1 Image Similarity Measures

The resulting parameters depend critically on the similarity function $d(\cdot, \cdot)$ used to evaluate how close the two images are to each other. The simplest is the Euclidean distance between the intensity values of corresponding pixels

$$d_{\text{Eucl}}(X_{\text{re}}, X_{\text{sy}}) = \sqrt{\sum_{v=1}^{H} \sum_{u=1}^{W} \left(X_{\text{re}}(u,v) - X_{\text{sy}}(u,v)\right)^2}\,, \tag{3}$$

where X_{re} and X_{sy} are the real and synthetic images respectively, and W and H denote the images dimensions.

However, our goal is to generate synthetic images that are more effective to train a detection method, and we will see that using this distance is not the best possible choice.

More specifically, we evaluated our approach in conjunction with three commonly used object detectors—DPM [6], an AdaBoost-based detector [8], and a CNN [21]—and we therefore introduce three different similarity functions, each one based on the image features used by one of these methods. Our experiments will show that our approach to image generation works best when relying on the distance function corresponding to the detection method.

Since DPM relies on Histograms-of-Gradients (HoG) [5], the first similarity function we consider the distance between the HoG vectors [5] computed for the two images as the similarity function

$$d_{\text{HoG}}(X_{\text{re}}, X_{\text{sy}}) = \sqrt{\sum_{i=1}^{L} \left(\text{HoG}_i(X_{\text{re}}) - \text{HoG}_i(X_{\text{sy}})\right)^2}, \tag{4}$$

where $\text{HoG}_i(X)$ is the i^{th} coordinate of the HoG vector computed for image X.

We also consider a detector based on AdaBoost with the weak learners based on the image gradients proposed in [17]. We write

$$h_{R,o,\tau}(X) = \begin{cases} 1, & \text{if } E(X, R, o) > \tau, \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

These weak learners are parametrized by a region R, an orientation o, and a threshold τ. $E(X, R, e)$ is the normalized image gradient energy over region R in X and in orientation o. We therefore introduce the additional function:

$$d^H_{\text{WL}}(X_{\text{re}}, X_{\text{sy}}) = \sqrt{\sum_{i=1}^{L} \alpha_i \left(h_i(X_{\text{re}}) - h_i(X_{\text{sy}})\right)^2}, \tag{6}$$

where L is the number of weak learners h_i with their corresponding weights α_i. We tried two different methods to build such a set:

- $d^R_{\text{WL}}(X_{\text{re}}, X_{\text{sy}})$ will denote the previous similarity function when random weak learners, each with a weight $\alpha = 1$, are used;
- $d^L_{\text{WL}}(X_{\text{re}}, X_{\text{sy}})$ will denote the previous similarity function when a set of weak learners and their weights selected by AdaBoost on the seed real images is used.

The third detection method we consider is a Convolutional Neural Network (CNN) which, contrary to the two previous method, does not rely on hard-coded image features but learns them instead. We therefore first train a CNN on the real seed images only and consider the distance

$$d_{\text{CNN}}(X_{\text{re}}, X_{\text{sy}}) = \sqrt{\sum_{n=2}^{N} \sum_{i=1}^{L_n} \left(\text{CNN}^n_i(X_{\text{re}}) - \text{CNN}^n_i(X_{\text{sy}})\right)^2}, \tag{7}$$

where $\text{CNN}^n_i(X)$ is the value of the i^{th} neuron of the n^{th} layer of the Convolutional Neural Network; N is the number of layers in the CNN; L_n is the number of neurons of the n^{th} layer of CNN.

In Fig. 4, we show synthetic images with the corresponding real seed images. Each image was obtained by finding the rendering parameters that minimize one of the five similarity functions introduced above.

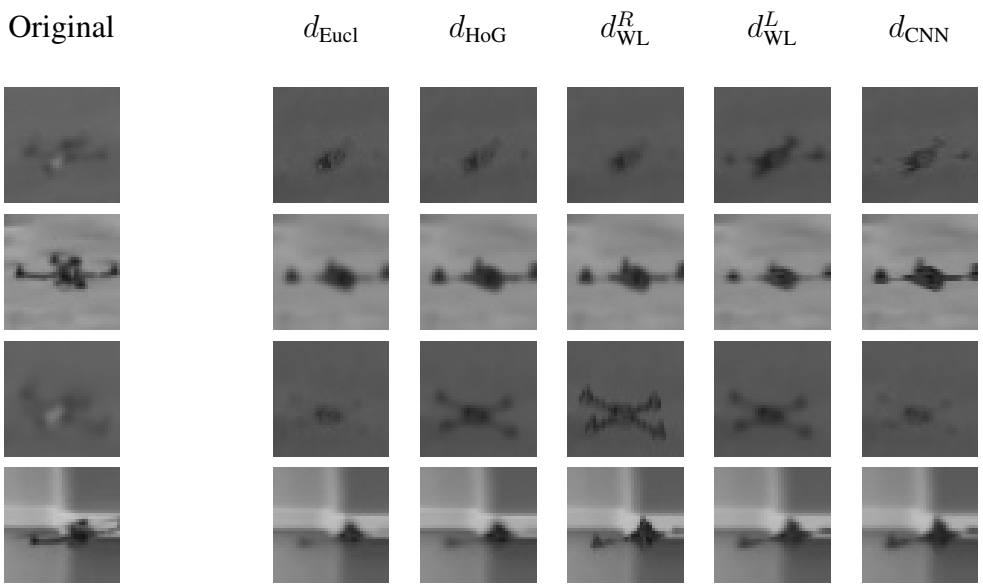


Figure 4: Samples of real images with corresponding synthetic ones. The Θ parameters for the synthetic images were optimised using different image similarity functions.

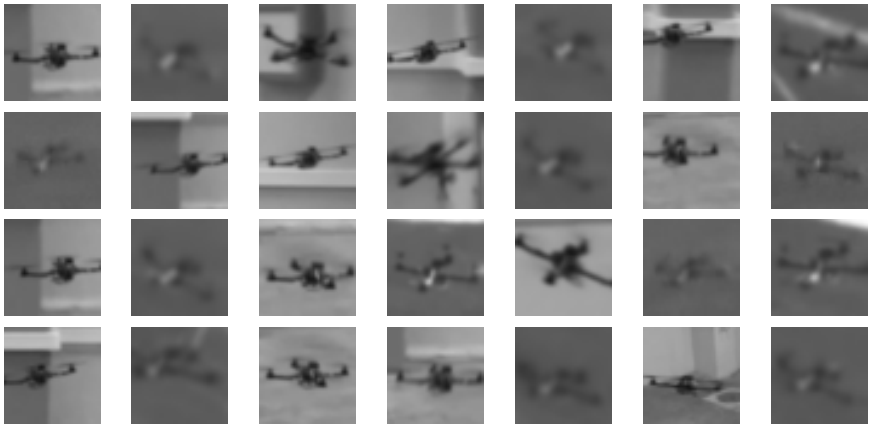


Figure 5: Sample real images from the dataset we collected for evaluation of our method.

5 Results

In this section, we evaluate our synthetic data generation method for UAV detection. We first present our dataset. We then evaluate how appropriate the different

similarity functions of Section 4.1 are for our purpose. We also study the relative influence of the different effects included into our rendering pipeline.

Finally, to demonstrate its applicability in a wider context, we apply our approach to creating synthetic training data for aircrafts of very different shapes and that are flying in various environments and lighting conditions.

5.1 Experimental Setup and Dataset

We created a dataset of 2 000 images of UAVs in various environments under different lighting conditions. Fig. 5 depicts some of the images. The images were captured from one UAV filming another one while the UAVs were both in motion.

To obtain the background images required to render the composite images, we first aligned consecutive frames by computing the homographies between the frames, and kept the median intensity at each location of the aligned images.

The training and testing videos were acquired in different environments and feature different backgrounds. The CAD model of the UAV used for rendering only coarsely outlines the main geometrical structure of the real object, as illustrated in Fig. 1(a). Negative training and testing samples were obtained by randomly sampling the backgrounds of the training images. For detection, we use a sliding window approach that applies the detector at every spatial location and at different scales of the whole image. Non maximum suppression is then applied to the response image scale-space.

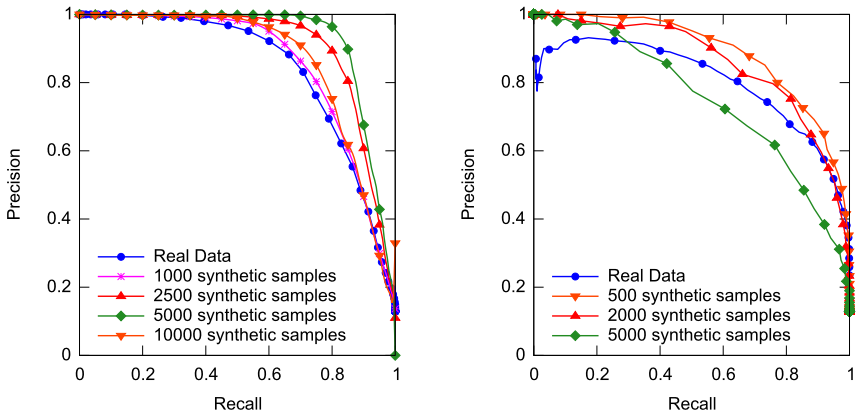
The classification methods in the experiments are trained with a combination of real and synthetic data and tested on the real data only.

5.2 Influence of the Number of Synthetic and Real Images

To evaluate how much we can improve the performances using synthetic images generated with our approach, we trained each of the classification methods we consider with different numbers of synthetic samples in addition to the training real samples. The synthetic samples were generated using the parameters obtained with the corresponding similarity functions.

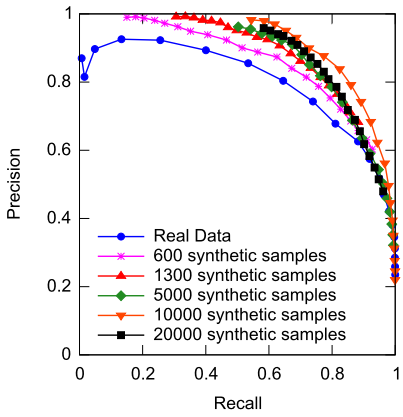
Fig. 6(a) compares the performances of these classifiers when varying the number of synthetic samples. It can be seen that using the synthetic image significantly improves performance over using the real images alone. However, this is only true up to a point when there are too many synthetic images.

We also evaluated the influence of the number of seed real images on the final performances, by decreasing the number of real images used to optimize the

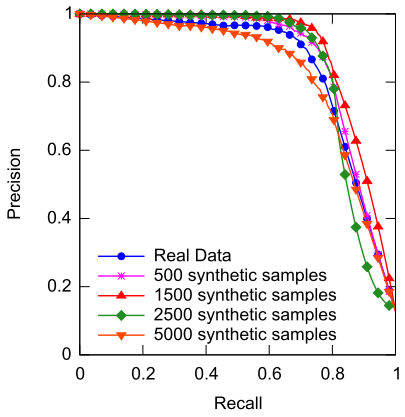


DPM and $d_{\text{HoG}}(.,.)$

AdaBoost and $d^R_{\text{WL}}(.,.)$



AdaBoost and $d^L_{\text{WL}}(.,.)$



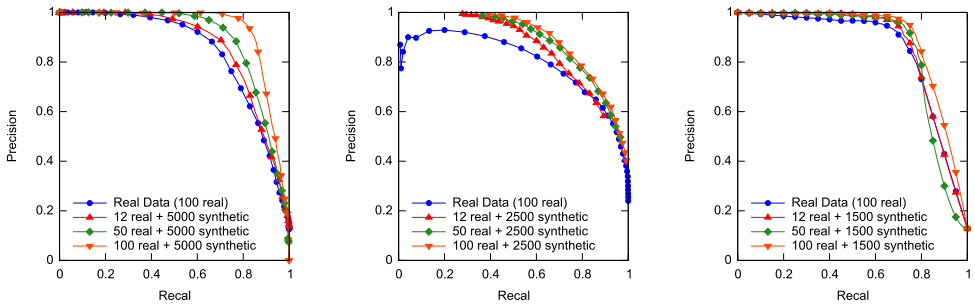
CNN and $d_{\text{CNN}}(.,.)$

Figure 6: We varied the number of synthetic images used for training, for three classification methods, using their corresponding similarity measures. Using both real and synthetic data for training phase increases performances compared to real data used alone. However using too much synthetic data may also hurt. (best seen in color)

rendering parameters. Fig. 7 shows the results for the AdaBoost classifier. Using as few as 12 real samples is enough to generate synthetic samples that allows us to outperform a classifier trained with about 8 times more real images. Unsurprisingly, increasing the number of seed real images results in an improvement of the final performances.

	Using real images only	Similarity measure: $d_{\text{Eucl}}(.,.)$	$d_{\text{HoG}}(.,.)$	$d^R_{\text{WL}}(.,.)$	$d^L_{\text{WL}}(.,.)$	$d_{\text{CNN}}(.,.)$
Classification method:		Average precision:				
DPM	0.84	0.78	**0.93**	0.70	0.72	0.67
AdaBoost	0.80	0.72	0.85	0.89	**0.92**	0.75
CNN	0.85	0.84	0.84	0.84	0.86	**0.89**

Table 1: Comparison of average precisions for each classification method, when the optimal number of synthetic images is used. Each classification method performs best with the corresponding similarity function.



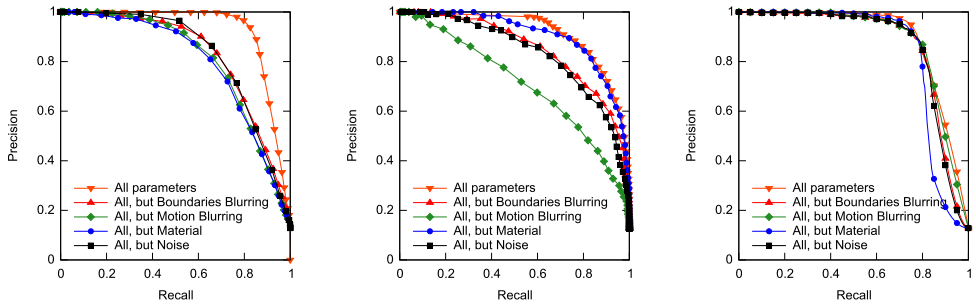
DPM and $d_{\text{HoG}}(.,.)$ AdaBoost and $d^L_{\text{WL}}(.,.)$ CNN and $d_{\text{CNN}}(.,.)$

Figure 7: Performances of the DPM, AdaBoost and CNN classifiers for different numbers of seed real images. Using as few as 12 real samples is enough to generate synthetic samples that allow us to outperform a classifier trained with 100 images. For every classifier synthetic images were generated using the corresponding similarity measure.

5.3 Optimal Performance

In this section, for each classification method, we use the optimal numbers of synthetic samples as discussed in the previous section. For comparison purposes, we also estimated the optimal numbers of synthetic samples when using the Euclidean distance $d_{\text{Eucl}}(.,.)$ as similarity measure.

Table 1 confirms that each classification method performs best with the corresponding distance function, and that using the Euclidean distance tends to be a



DPM and $d_{\text{HoG}}(.,.)$ AdaBoost and $d^{L}_{\text{WL}}(.,.)$ CNN and $d_{\text{CNN}}(.,.)$

	Synthetic data generation effects:				
	All	no Boundary Blurring	no Motion Blur	no Random Noise	no Material Properties variation
Classification method:			Average precision:		
DPM	**0.93**	0.83	0.84	0.83	0.80
AdaBoost	**0.92**	0.85	0.71	0.75	0.91
CNN	**0.89**	0.88	**0.89**	0.88	0.85

Figure 8: Evaluation of the synthetic data generation effects. We fixed one capture parameter in Θ and then optimized the other parameters using the best similarity measure for each classification method. Each effect has clearly a positive influence of the quality of the synthetic data, however the impact is different for every classification method. (best seen in color)

bad idea. The best performances are obtained with DPM trained with both real and synthetic images, even though CNN was better than DPM when no synthetic images were used. This highlights the advantage of using the synthetic images generated using our method.

5.4 Relative Importance of the Various Rendering Effects

To check whether correctly setting all the capture parameters is truly important, we performed the following set of evaluations. We fixed one capture parameter in Θ and then optimized the other parameters using the best similarity measure for each classification method. We then used the resulting Θ's to generate the synthetic images, trained the corresponding classifier on these images and evaluated this classifier on the test real images. We repeated the experiments for each of the capture parameters.

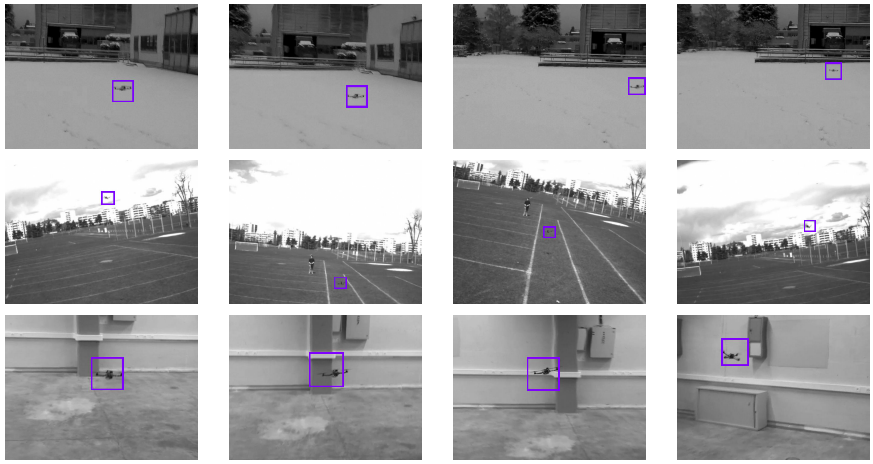


Figure 9: Examples of detections made by the AdaBoost classifier trained on both real and synthetic samples on different real UAV video sequences.

Figure 10: The three CAD models we used for our seed dataset. They are freely available on the internet.

The results are shown in Fig. 8. Each effect has clearly a positive influence on the final performances.

5.5 Detecting Multiple Kinds of Aircrafts

We trained the AdaBoost classifier on both real and synthetic data and tested it on real video sequences that have different backgrounds comparing to the ones that were used for the synthetic data generation. Sample detections are shown in Fig. 9.

Additionally we generated synthetic data using the models of three types of different fixed-wing aircrafts and tested them on different real video sequences. The models that we used are shown on Fig. 10.

We use 100 real images of 3 types of aircrafts with their respective background images. These images were collected by manually annotating different video sequences where the aircrafts fly in different weather conditions and appear

Figure 11: Sample seed real images of the aircrafts.

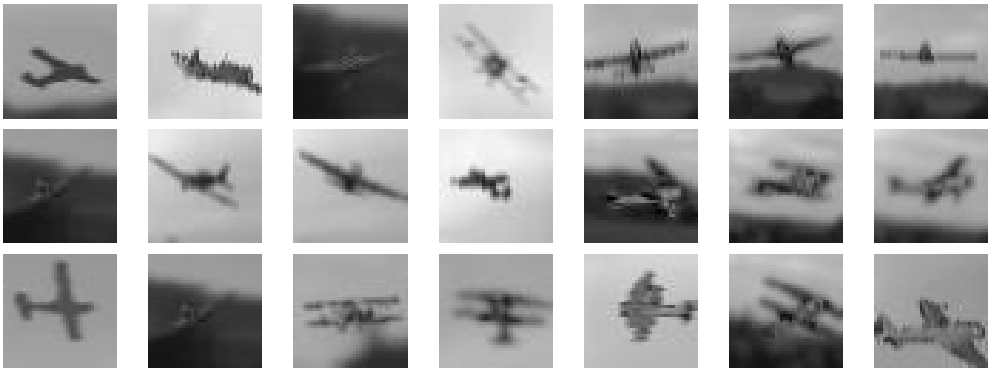


Figure 12: Synthetic samples of the aircrafts.

at different angles. Sample images from this dataset are shown on Fig. 11.

For the detection of the aircrafts, we used the AdaBoost classifier trained using real and synthetic images generated based on the $d^L_{\text{WL}}(.,.)$ similarity function. We generated 10,000 synthetic samples to supplement the real images and used them as a training dataset. Sample synthetic images are depicted by Fig. 12.

The testing dataset consists of 8 video sequences, one of which contains 5,000 frames, while the others are made of 500 frames. These sequences show different types of aircrafts flying in different environments and weather conditions. Using the detector trained on both real and synthetic images we could achieve about 80% detection accuracy while the detection accuracy was about 67% when using only real images. Sample detections are shown in Fig. 13.

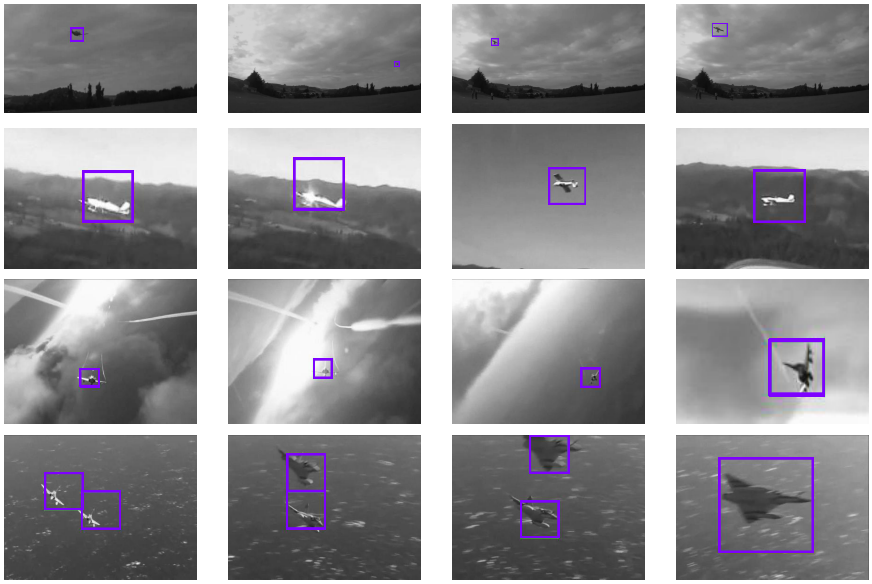


Figure 13: Examples of detections made by the classifier trained on both real and synthetic samples on various aircraft and drone video sequences.

6 Conclusion

We showed that by properly optimizing the parameters of a very simple rendering pipeline, we can generate synthetic images that significantly improve the performances of an image classifier when used for training. We believe our parameter optimization scheme is a powerful tool to manage the large numbers of parameters a more complex rendering pipeline could have. It therefore opens new doors towards the use of sophisticated Computer Graphics and post-processing effects to generate images even closer to real ones, and to relax the cumbersome need for large numbers of real images to train Computer Vision methods.

References

[1] V. Athitsos and S. Sclaroff. Estimating 3D Hand Pose from a Cluttered Image. In *Conference on Computer Vision and Pattern Recognition*, pages 432–439, June 2003.

[2] S. Baker, D. Scharstein, J.P. Lewis, S. Roth, M.J. Black, and R. Szeliski. A Database and Evaluation Methodology for Optical Flow. *International Journal of Computer Vision*, 92:1–31, 2011.

[3] J. L. Barron, D. J. Fleet, and S. S. Beauchemin. Performance of Optical Flow Techniques. *International Journal of Computer Vision*, 12:43–77, 1994.

[4] D. Cireşan, A. Giusti, L.M. Gambardella, and J. Schmidhuber. Deep Neural Networks Segment Neuronal Membranes in Electron Microscopy Images. In *Advances in Neural Information Processing Systems*, 2012.

[5] N. Dalal and B. Triggs. Histograms of Oriented Gradients for Human Detection. In *Conference on Computer Vision and Pattern Recognition*, 2005.

[6] P. Felzenszwalb, D. Mcallester, and D. Ramanan. A Discriminatively Trained, Multiscale, Deformable Part Model. In *Conference on Computer Vision and Pattern Recognition*, June 2008.

[7] F. Fleuret and D. Geman. Coarse-To-Fine Visual Selection. *International Journal of Computer Vision*, 41(1):85–107, January 2001.

[8] Y. Freund and R.E. Schapire. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. In *European Conference on Computational Learning Theory*, pages 23–37, 1995.

[9] A. Handa, R.A. Newcombe, A. Angeli, and A.J. Davison. Real-Time Camera Tracking: When is High Frame-Rate Best? In *European Conference on Computer Vision*, pages 222–235, 2012.

[10] B.K.P. Horn and B.G. Schunck. Determining Optical Flow. *Artificial Intelligence*, 17:185–204, 1981.

[11] B. Kaneva, A. Torralba, and W.T. Freeman. Evaluation of Image Features Using a Photorealistic Virtual World. In *International Conference on Computer Vision*, 2011.

[12] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.

[13] G. Klein and D. W. Murray. Simulating Low-Cost Cameras for Augmented Reality Compositing. *IEEE Transactions on Visualization and Computer Graphics*, 16(3):369–380, 2010.

[14] Q. Le, M. Ranzato, R. Monga, M. Devin, K. Chen, G. Corrado, J. Dean, and A. Ng. Building High-Level Features Using Large Scale Unsupervised Learning. In *International Conference on Machine Learning*, 2012.

[15] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 1998.

[16] V. Lepetit and P. Fua. Keypoint Recognition Using Randomized Trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(9):1465–1479, September 2006.

[17] K. Levi and Y. Weiss. Learning Object Detection from a Small Number of Examples: the Importance of Good Features. In *Conference on Computer Vision and Pattern Recognition*, 2004.

[18] J. Liebelt and C. Schmid. Multi-View Object Class Detection with a 3D Geometric Model. In *Conference on Computer Vision and Pattern Recognition*, 2010.

[19] J. Marin, D. Vázquez, D. Geronimo, and A. M. Lopez. Learning Appearance in Virtual Scenarios for Pedestrian Detection. In *Conference on Computer Vision and Pattern Recognition*, pages 137–144, 2010.

[20] L. Pishchulin, A. Jain, A. Mykhaylo, T. Thormaehlen, and B. Schiele. Articulated People Detection and Pose Estimation: Reshaping the Future. In *Conference on Computer Vision and Pattern Recognition*, 2012.

[21] T. Serre, L. Wolf, S. Bileschi, M. Riesenhuber, and T. Poggio. Robust Object Recognition with Cortex-Like Mechanisms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2007.

[22] J. Shotton, R. Girshick, A. Fitzgibbon, T. Sharp, M. Cook, M. Finocchio, R. Moore, P. Kohli, A. Criminisi, A. Kipman, and A. Blake. Efficient Human Pose Estimation from Single Depth Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 99, 2012.

[23] M. Stark, M. Goesele, and B. Schiele. Back to the future: Learning shape models from 3d cad data. In *British Machine Vision Conference*, pages 106.1–106.11, 2010.

[24] L. Taycher, G. Shakhnarovich, D. Demirdjian, and T. Darrell. Conditional Random People: Tracking Humans with CRFs and Grid Filters. In *Conference on Computer Vision and Pattern Recognition*, 2006.

[25] T. Varga and H. Bunke. Generation of Synthetic Training Data for an Hmm-Based Handwriting Recognition System. In *7th Int. Conference on Document Analysis and Recognition*, pages 618–622, 2003.