RENDERGAN: GENERATING REALISTIC LABELED DATA

Leon Sixt, Benjamin Wild, & Tim Landgraf
Fachbereich Mathematik und Informatik
Freie Universität Berlin
Arnimalle 7 Berlin, Germany
{leon.sixt, benjamin.wild, tim.landgraf}@fu-berlin.de

arXiv:1611.01331v2 [cs.NE] 7 Nov 2016

ABSTRACT

Deep Convolutional Neuronal Networks (DCNN) are showing remarkable performance on many computer vision tasks. Due to their large parameter space, they require many labeled samples when trained in a supervised setting. The costs of annotating data manually can render the usage of DCNNs infeasible. We present a novel framework called RenderGAN that can generate large amounts of realistic, labeled images by combining a 3D model and the Generative Adversarial Network framework. In our approach, image augmentations (e.g. lighting, background, and detail) are learned from unlabeled data such that the generated images are strikingly realistic while preserving the labels known from the 3D model. We apply the RenderGAN framework to generate images of barcode-like markers that are attached to honeybees. A DCNN is trained on this data only. It performs better on a test set of real data than an equal DCNN trained on the limited amounts of real data available.

1 INTRODUCTION

When an image is taken from a real world scene, many factors determine the final appearance: background, lighting, object shape, position and orientation of the object, the noise of the camera sensor, and more. In computer vision, high-level information such as class, shape, or pose is reconstructed from raw image data. Most real-world applications require the reconstruction to be invariant to noise, background, and lighting changes.

In recent years, deep convolutional neural networks (DCNN) advanced to the state of the art in many computer vision tasks (Krizhevsky et al., 2012; He et al., 2015; Razavian et al., 2014). More training data usually increases the performance of DCNNs. While image data is mostly abundant, labels for supervised training must often be created manually – a time-consuming and tedious activity. The costs of labeling are considerable for tasks with complex annotations, e.g. human joint angles, camera viewpoint or image segmentation.

A common approach to deal with limited amount of labels is data augmentation (Goodfellow et al., 2016, Chapter 7.4). Translation, noise, and other deformations can often be applied without changing the labels, thereby effectively increasing the number of training samples and reducing overfitting.

DCNNs learn a hierarchy of features – many of which are applicable to related domains (Yosinski et al., 2014). Therefore, a common technique is to pre-train a model on a larger dataset such as ImageNet (Deng et al., 2009) and then fine-tune its parameters to the task at hand (Girshick et al., 2014; Long et al., 2015; Razavian et al., 2014). This technique only works in cases where a large enough related dataset exists. Furthermore, labeling enough data for fine-tuning might still be costly.

If a basic model of the data exists (for example, a 3D model of the human body), it can be used to generate labeled data. Su et al. (2015) used 3D-CAD models from online repositories to generate large amounts of training images for the viewpoint estimation task on the PASCAL 3D+ dataset (Xiang et al., 2014). Richter et al. (2016) and Ros et al. (2016) used 3D game engines to collect labeled data for image segmentation. However, the explicit modeling of the image acquisition

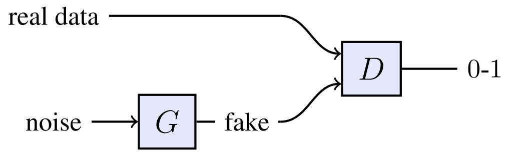


Figure 1: Topology of a GAN. The discriminator network D is trained to distinguish between "fake" and real data. The generator network G receives a random vector as input. G is optimized to maximize the chance of the discriminator making a mistake.

physics (scene lighting, reflections, lense distortions, sensor noise, etc.) is cumbersome and might still not be able to fully reproduce the particularities of the imaging process such as unstructured background or object specific noise. Training a DCNN on generated data that misses certain features will result in overfitting and poor performance on the real data. Therefore, previous work relied partially on real data, e.g. by using pre-trained models or mixing generated and real data for training.

Generative Adversarial Networks (GAN) (see Fig. 1) can learn to generate high-quality samples from unlabeled data (Goodfellow et al., 2014). For example, Denton et al. (2015) synthesized images with a GAN on the CIFAR dataset (Krizhevsky, 2009), which were hard for humans to distinguish from real images. However, no information can be inferred about the generated samples. The representation of GANs is too entangled to be useful for collecting labels. Springenberg (2015) explored GANs in a semi-supervised setting. Chen et al. (2016) use an information theory approach to disentangle the representation. They successfully learned to associate certain dimensions of the representation of high-level concepts such as camera viewpoint or hair style. However, explicitly controlling the relationship between the latent space and generated samples is an open problem. For generating labeled samples, it is necessary to exactly know the value of the parameters, e.g. the exact orientation of the camera or object position. Any error will result in decreased performance of the supervised algorithm trained with the generated data.

We solved this problem with the RenderGAN framework by embedding a 3D model into the generator network of a GAN. Samples from the 3D model are rendered realistic by adding missing factors (e.g. blurriness, lighting, background, or details). We constrain the augmentation of the images such that the high-level information represented by the 3D model is preserved. The RenderGAN framework allows us to generate images of which the labels are known from the 3D model, and that also look strikingly real due to the GAN framework. The training procedure of the RenderGAN framework does not require any labels. We can generate high-quality, labeled data with a simple 3D model and a large amount of unlabeled data. Our approach reduces the costs of labeling considerably.

The RenderGAN framework was developed to solve the scarcity of labels in the BeesBook project (Wario et al., 2015) in which we analyze the social behavior of honeybees. A barcode-like marker is attached to the honeybees' backs for identification (see Fig. 3). Annotating this data is tedious, and therefore only a limited amount of labeled data exists. A basic 3D model of the tag (see the upper row of Fig. 2) captures the position, orientation, and bit configuration. The RenderGAN learned lighting, background, and details from unlabeled data. We evaluate the RenderGAN by comparing DCNNs trained on either the generated data or on the limited amount of labeled data available. The DCNN trained on the generated data performs better on the real test set.

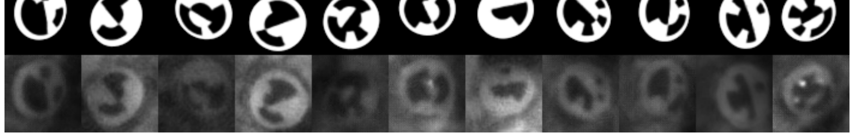
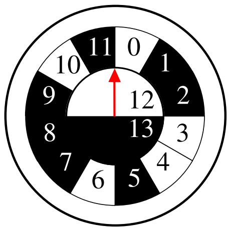
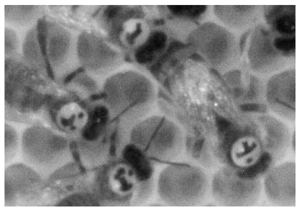


Figure 2: Images generated with the RenderGAN and the corresponding images from the 3D model.



(a) Tag structure



(b) Tagged bees in the hive

Figure 3: **(a)** The tag represents a unique binary code (cell 0 to 11) and encodes the orientation with the semicircles 12 and 13. The red arrow points towards the head of the bee. This tag encodes the id `100110100010`. **(b)** Cutout from a high-resolution image. The bee on the right is the queen.

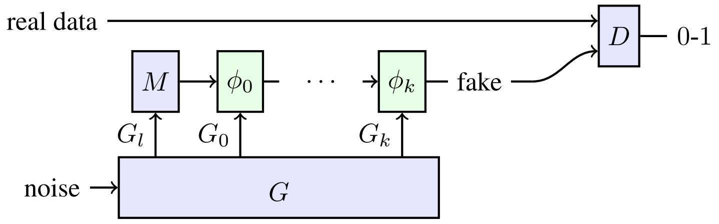


Figure 4: The generator G cannot directly produce samples. Instead, G has to use a 3D model M to generate a simple image, which is then modified by the parameters G_i through the augmentation functions ϕ_i to match the real data.

2 RenderGAN

Most supervised learning tasks can be modeled as a regression problem, i.e. approximating a function $\hat{f} : \mathbb{R}^n \mapsto L$ that maps from data space $\mathbb{R}$ to label space L. We consider $\hat{f}$ to be the best available function on this particular task. Analog to ground truth data, one could call $\hat{f}$ the ground truth function.

In the RenderGAN framework, we aim to solve the inverse problem to this regression task: generate data given the labels. This is achieved by embedding a simple 3D model into the generator of a GAN. The samples generated by the simple model must be representative of the given labels but may lack many factors of the real data such as background or lightning. Through a cascade of augmentation functions, the generator can adapt the images from the 3D model to match the real data.

Let's start by formalizing image augmentation as a function $\phi(x, d)$, which changes the image x based on the augmentation parameter d (a tensor of any rank). The augmentation must preserve the labels of the image x. Therefore, it must hold for all images x and all augmentations parameters d:

$$\hat{f}\left(\phi(x, d)\right) = \hat{f}(x) \tag{1}$$

The augmentation function must furthermore be differentiable w.r.t. x and d as the gradient will be back-propagated through ϕ to the generator. Image augmentations such as lighting, surrounding, and noise do preserve the labels and fit this definition. We will provide appropriate definitions of ϕ for the mentioned augmentations in the following chapter.

If appropriate augmentation functions are found that can model the missing factors and are differentiable, we can use the GAN framework to find parameters that result in realistic output images. Multiple augmentation functions can be combined to perform a more complex augmentation. In the simplest form, the augmentations are applied sequentially. Generally, the augmentation functions can form a DAG. Here, we consider only the sequential case. Suppose we have k augmentation functions and k corresponding outputs G_i from the generator. G can also predict the parameters of the 3D model M – denoted by $G_l(z)$. This requires the 3D model

M to be differentiable, which can be achieved by emulating the 3D model with a neural network (Dosovitskiy et al., 2015). Alternatively, if the label distribution is known, one could sample the 3D model parameters directly. The resulting generator $g(z)$ can be written as (see Fig. 4 for a visual interpretation):

$$g(z) = \phi_0(\phi_1(\dots\phi_k(M(G_l(z)), G_k(z))\dots, G_1(z)), G_0(z)) \tag{2}$$

As any differentiable function approximator can be employed in the GAN framework, the theoretical properties still hold. The training is carried out as in the conventional GAN framework. In a real application, the augmentation functions might restrict the generator from converging to the data distribution.

If the training converges, we can collect generated realistic data with $g(z)$ and the high-level information captured in the 3D model with $G_l(z)$. We can now train a supervised learning algorithm on the labeled generated data $(G_l\,(z)\,, g\,(z))$ and solve the regression task of approximating $\hat{f}$ without depending on manual labels.

3 Application to the BeesBook project

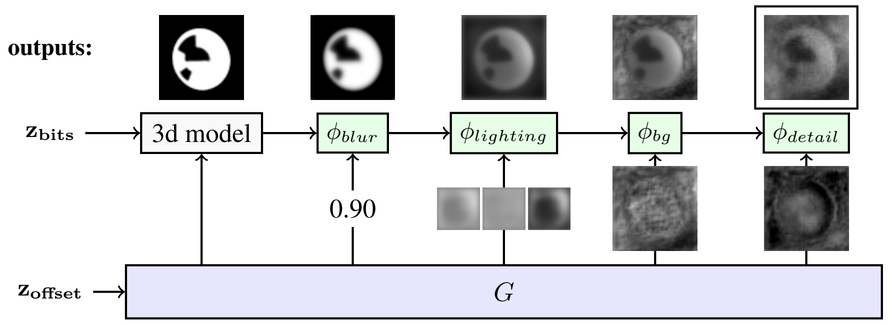


Figure 5: Augmentation functions of the RenderGAN in the BeesBook project. The arrows from G to the augmentation functions ϕ show the inputs to the augmentation functions. The generator provides the position and orientations to the 3d model, whereas the bits are sampled uniformly. On top, the output of each stage is shown. The output of ϕ_{detail} is forwarded to the discriminator.

In the BeesBook project, we aim to understand the complex social behavior of honey bees. For this analysis, we rely on the trajectories of honeybees. For identification, a tag with a binary code is attached to the back of the bees. Four high-resolution cameras record the beehive over multiple weeks.

The orientation, position, and bits of the tags are needed to build the trajectories of the bees over time. Decoding this information is not trivial: the bees are oriented in any direction in space. The tag might be partially occluded. Moreover, spotlights on the tag can sometimes even puzzle humans. A previously used computer vision pipeline did not have the performance to track the bees reliable. Although we assumed DCNN would perform well on this task, the high costs of manual annotations made it infeasible to acquire enough labels for training. We therefore wanted to synthesize images in such a way that the parameters of the tag are known and which are realistic enough to train a neural network on them from scratch.

The 3D model of the tag takes the position, orientation, and id of a tag as parameters. Although the 3D model represents the given parameters well, the generated images lack many important factors: lighting, blurriness, background, and details (see Fig. 2). The 3D model furthermore provides a background segmentation mask and a depth map of the tag.

The binary ids are sampled uniformly during training. As the distribution of the position and orientation of the tag are unknown, the generator learns to predict them. The discriminator loss must be backpropagated through the 3D model to the generator. Therefore, we train a neural network to emulate the 3D model. Its output is indistinguishable from the images of the 3D model. The weights of the 3D network are fixed during the RenderGAN training.

We apply different augmentation functions that account for blurriness, lighting, background, and details. In many places of the generator, we will have to restrict the range of values, which is done with the clip layer that we will introduce later. The output of the 3D model and of each augmentation function is of shape $(64, 64)$ and in the range $[-1, 1]$. In Fig. 5, the structure of the generator is shown.

Blurriness: The 3D model produces hard edges, but the images of the real tags show a broad range of blurriness. The generator produces a scalar $\alpha \in [0, 1]$ per image that controls the blurriness.

$$\phi_{blur}(x, \alpha) = (1 - \alpha)\,(x - b_\sigma\,(x)) + b_\sigma(x) \tag{3}$$

where $b_\sigma(x) = x * k_\sigma$ denotes convolving the image x with a Gaussian kernel k_σ of scale σ. The implementation of the blur function is inspired by Laplacian pyramids (Burt & Adelson, 1983). As required for augmentation functions, the labels are preserved, because we limit the maximum amount of blurriness by picking $\sigma = 2$. ϕ_{blur} is also differentiable w.r.t the inputs α and x.

Lighting of the tag: The images from the 3D model are black and white. In real images, tags exhibit different shades of gray. We model the lighting by a smooth scaling and shifting of the pixel intensities. The generator provides three outputs for the lighting: scaling of black parts s_b, scaling of white parts s_w and a shift t. All outputs have the same dimensions as the image x. An important invariant is that the black bits of the tag must stay darker than the white bits. Otherwise, a bit could flip, and the label would change. By restricting the scaling s_w and s_b to be between 0.10 and 1, we ensure that this invariant holds. The lighting should cause smooth changes in the image. Hence, Gaussian blur $b(x)$ is applied to s_b, s_w, and t.

$$\phi_{lighting}(x, s_w, s_b, t) = x \cdot b(s_w) \cdot W(x) + x \cdot b(s_b) \cdot (1 - W(x)) + b(t) \tag{4}$$

The segmentation mask $W(x)$ is one for white parts and zero for the black part of the image. As the intensity of the input is distributed around -1 and 1, we can use thresholding to differentiate between black and white parts.

Background: The background augmentation can change the background pixels arbitrarily.

$$\phi_{bg}(x, d) = x \cdot (1 - B(x)) + d \cdot B(x) \tag{5}$$

Here, B denotes a segmentation mask which is one for pixels belonging to the background and zero elsewhere. The mask is provided by the 3D model. As ϕ_{bg} can only change background pixels, the labels remain unchanged.

Details: In this stage, the generator can add small details to the whole image including the tag. The output of the generator d is passed through a high-pass filter to ensure that the added details are small enough not to flip a bit. Furthermore, d is restricted to be in $[-2, 2]$ to make sure the generator cannot avoid the highpass filter by producing huge values. With the range $[-2, 2]$, the generator has the possibility to change black pixels from to white, which is needed to model spotlights.

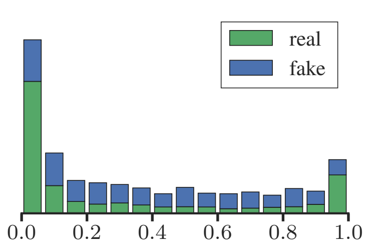
$$\phi_{detail}(x, d) = x + \text{highpass}(d) \tag{6}$$

The high-pass is implemented by taking the difference between the image and a blurred version of the image ($\sigma = 3.5$). As the spotlights on the tags are only a little smaller than the bits, we increase its slope after the cutoff frequency by repeating the high-pass filter three times.

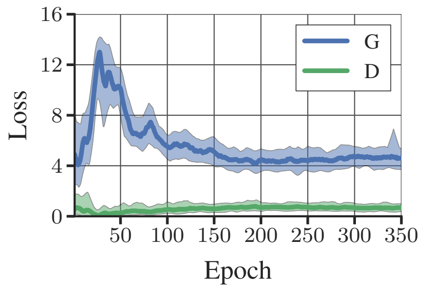
The image augmentations are applied in the order as listed above: $\phi_{detail} \circ \phi_{background} \circ \phi_{lighting} \circ \phi_{blur}$. Please note that there exist parameters to the augmentation functions that could change the labels. As long as it is guaranteed that such augmentations will result in unrealistic looking images, the generator network will learn to avoid them. For example, while the detail augmentation could be used to add high-frequency noise to obscure the tag, this artifact would be easily detected by the discriminator.

Architecture of the generator: The generator network has to produce outputs for each augmentation function. We will outline only the most important parts. See our code available online for all the details of the networks[1]. The generator starts with a small network consisting of dense layers, which predicts the parameters for the 3D model (position, orientations). The output of another dense layer is reshaped and used as starting block for a chain of convolution

[1]`https://github.com/berleon/deepdecoder`



(a)



(b)

Figure 6: **(a)** Histogram of the discriminator scores on fake and real samples. **(b)** Losses of the generator (G) and discriminator (D).

and upsampling layers. We found it advantageous to merge a depth map of the 3D model into the generator as especially the lighting depends on the orientation of the tag in space. The input to the blur augmentation is predicted by reducing an intermediate convolutional feature map to a single scalar. An additional network is branched off to predict the input to the lighting augmentation. For the background generation, the output of the lighting network is merged back into the main generator network together with the actual image from the 3D model.

For the discriminator architecture, we mostly rely on the architecture given by Radford et al. (2015), but doubled the number of convolutional layers and added a final dense layer. This change improved the quality of the generated images.

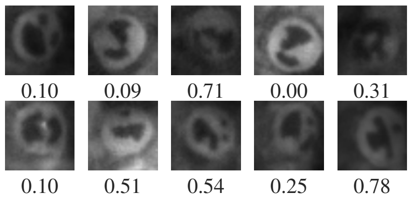
Clip layer: At multiple times in the generator, we have to restrict the output to a fixed range, usually $[-1; 1]$. Functions like $\tanh$ or $\mathrm{sigmoid}$ did not work for our use-case. As they squash all numbers to a fixed interval, the gradient vanishes if the output comes near to the bounds. We are using a combination of clipping and activity regularization to keep the output in a given interval $[a, b]$. If the input x is out of bounds, it is clipped and a regularization loss r depending on the distance between x and the appropriate bound is added.

$$r(x) = \begin{cases} \alpha||x - a||_1 & \text{if } x < a \\ 0 & \text{if } a \leq x \leq b \\ \alpha||x - b||_1 & \text{if } x > b \end{cases} \tag{7}$$

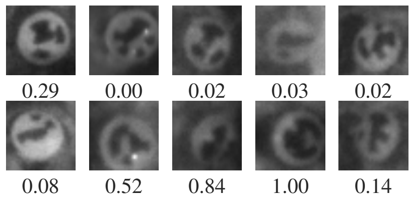
$$f(x) = \min(\max(a, x), b) \tag{8}$$

With the scalar α, the weight of the loss can be adapted. For us $\alpha = 15$ worked well. If α is picked too small, the regularization loss might not be big enough to move the output of the previous layer towards the interval $[a, b]$. During training, we observe that the loss decreases to a small value but never vanishes. If we replace the clip layers in the generator with tanh activations, the training does no longer converge.

Training: We train generator and discriminator as in the normal GAN setting. We use 2.4M unlabeled images of tags to train the RenderGAN. We use Adam (Kingma & Ba, 2014) as an optimizer with a starting learning rate of 0.0002, which we reduce in epoch 200, 250, and 300 by a



(a) Generated



(b) Real

Figure 7: Discriminator scores for generated and real images.

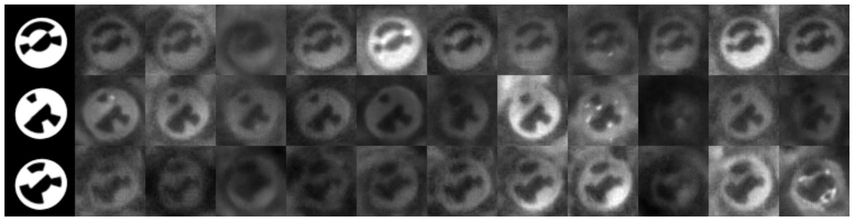


Figure 8: Random points in the z-space given the tag parameters.

factor of 0.25. In Fig. 6b the training loss of the GAN is shown. The GAN does not converge to the point where the discriminator can no longer separate generated from real samples. The augmentation functions might restrict the generator too much such that it cannot model certain properties. Nevertheless, it is hard for a human to distinguish the generated from real images. In some cases, the generator creates artifacts that destroy the images. As these bad images are scored unrealistic by the discriminator without exception, we can discard them for the training of the supervised algorithm. In Fig. 8, we show random points in the latent space, while fixing the tag parameters. The generator indeed learned to model the various lighting conditions, noise intensities, and backgrounds.

4 Results

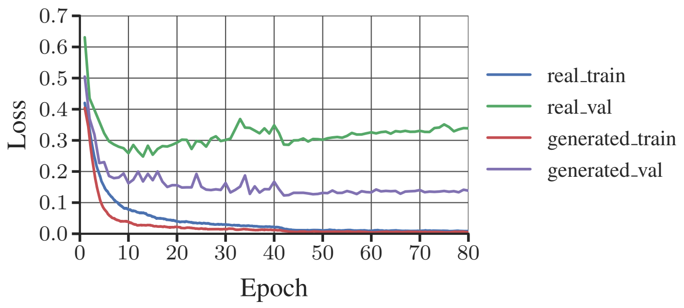


Figure 9: Train and validation losses from the training with the generated data (*generated_train* / *generated_val*) and from the training with the real data (*real_train* / *real_val*).

We compare the performance of a DCNN trained on generated and real data. The generated training set contains 5 million tags created with the RenderGAN framework. At this amount one training sample is only used twice during training. An epoch consists of 1000 batches á 128 samples. The labels of the real data are extracted from ground truth data that was originally collected to evaluate bee trajectories. This ground truth data contains the path and id of each bee over multiple consecutive frames. Data from five different time spans was annotated – in total 66K tags. As the data is correlated in time (same ids, similar lighting conditions), we assign the data from one time span completely to either the train or test set. The data from three time spans forms the train set (40K). The test set (26K) contains data from the remaining two time spans. The ground truth data lacks the orientation of the tags, which is therefore omitted for this evaluation.

We use early stopping to select the best weights of the networks. For the training with the generated data, we use the real train set as the validation set. When training on real data, the test set is also used for validation. We could alternatively reduce the training set further and form an extra validation set, but this would harm the performance of the DCNN trained on the real data.

Due to the smaller size of the real training set, we augment it with random translation, rotation, shear transformation, histogram scaling, and noise. The generated data is not augmented. We use the 34-layer ResNet architecture (He et al., 2015) but start with 16 feature maps instead of 64. The training loss of both networks is plotted in Fig. 9.

Table 1: Comparison of the mean Hamming distance (MHD).

	DCNN on generated data	DCNN on real data	CV pipeline
MHD	0.42	1.09	1.08

The DCNNs are evaluated on the mean Hamming distance (MHD) i.e. the expected value of bits decoded wrong. Trained humans can decode tags with a MHD of around 0.23. Bee tags can be partially occluded, e.g. when a bee sticks its head into the comb.

In the evaluation, we also include the previously used computer vision pipeline. It used a manual feature extraction, for example, a modified Hough transformation to find ellipses. Its MHD is only a very rough estimate given that the computer vision pipeline was evaluated and fine-tuned on the same data set.

In Table 1, we present the results of this evaluation. The best performing model is the DCNN trained on the generated data. If we use predictions from this DCNN instead of the computer vision pipeline, the accuracy of the tracking improves from 55% of the ids assigned correctly to 96%. At this quality, it is possible to analyze the social behavior of the honeybees reliably.

5 Discussion

We proposed a novel extension to the GAN framework that is capable of rendering samples from a basic 3D model more realistic. Compared to computer graphics pipelines, the RenderGAN can learn complex effects from unlabeled data that would be otherwise hard to model with explicit rules. Contrary to conventional GANs, the generator provides explicit information about the synthesized images, which can be used as labels for a supervised algorithm.

We showed an application of the RenderGAN framework to the BeesBook project, in which the generator adds blurriness, lighting, background, details to images from a basic 3D model. The generated data looks strikingly real and includes fine details such as spotlights or specific noise.

Our method is an improvement over previous work that applied 3D models to produce training samples for DCNNs (Su et al., 2015; Richter et al., 2016; Ros et al., 2016). Whereas previous work relied on real data for training using pre-trained models or mixing real and generated data, we were able to train a DCNN from scratch with generated data that performed well when tested on real data.

While some work is required to adapt the RenderGAN to a specific domain, once set up, arbitrary amounts of labeled data can be acquired cheaply, even if the data distribution changes. For example, if the tag design changes to include more bits, small adaptions to the 3D model's source code and eventually the hyperparameters of the augmentation functions would be sufficient. However, if we had labeled the data by hand, then we would have to annotate data again.

A disadvantage of the RenderGAN framework is that the augmentation functions must be carefully crafted for the application at hand to ensure that high-level information is preserved. Furthermore, a suitable 3D model must be available.

6 Future Work

For future work, it would be interesting to see the RenderGAN framework used on other tasks where basic 3D models exist e.g. human faces, pose estimation, or viewpoint prediction. In this context, one could come up with different augmentation functions e.g. colorization, affine transformations, or diffeomorphism. The RenderGAN could be especially valuable to domains where pre-trained models are not available or when the annotations are very complex. Another direction of future work might be to extend the RenderGAN framework to other fields. For example, in speech synthesis, one could use an existing software as a basic model and improve the realism of the output with a similar approach as in the RenderGAN framework.

REFERENCES

Peter Burt and Edward Adelson. The laplacian pyramid as a compact image code. *IEEE Transactions on communications*, 31(4):532–540, 1983.

Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. *arXiv preprint arXiv:1606.03657*, 2016.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255. IEEE, 2009.

Emily L Denton, Soumith Chintala, Rob Fergus, et al. Deep generative image models using a laplacian pyramid of adversarial networks. In *Advances in neural information processing systems*, pp. 1486–1494, 2015.

Alexey Dosovitskiy, Jost Tobias Springenberg, and Thomas Brox. Learning to generate chairs with convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1538–1546, 2015.

Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587, 2014.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pp. 2672–2680, 2014.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016. URL `http://www.deeplearningbook.org`.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images, 2009. ISSN 1098-6596.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully Convolutional Networks for Semantic Segmentation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3431–3440, 2015. ISSN 10636919. doi: 10.1109/CVPR.2015.7298965.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, 2015.

Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 806–813, 2014.

Stephan R Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. In *European Conference on Computer Vision*, pp. 102–118. Springer, 2016.

German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio M Lopez. The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3234–3243, 2016.

Jost Tobias Springenberg. Unsupervised and semi-supervised learning with categorical generative adversarial networks. *arXiv preprint arXiv:1511.06390*, 2015.

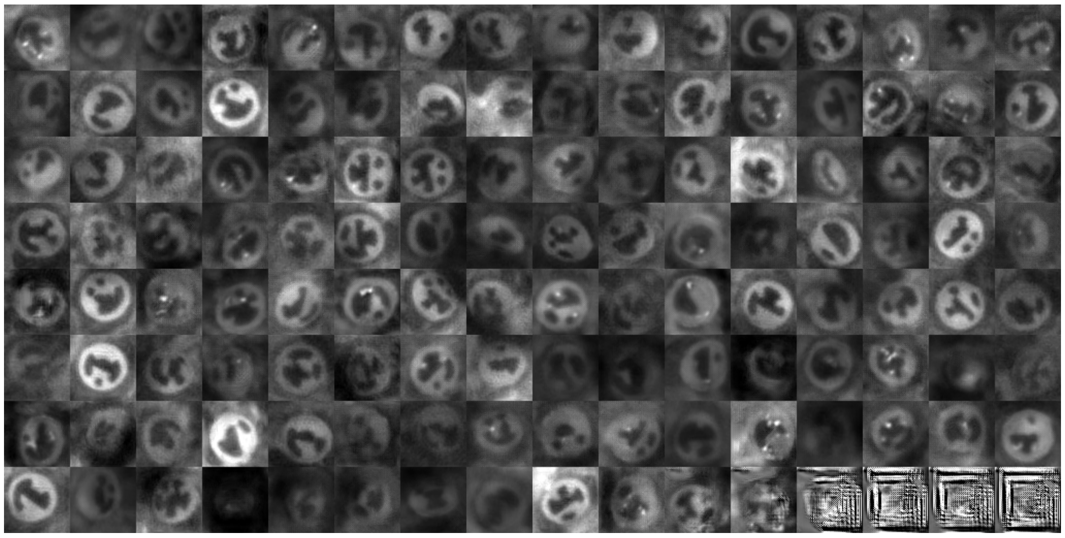
Hao Su, Charles R Qi, Yangyan Li, and Leonidas J Guibas. Render for cnn: Viewpoint estimation in images using cnns trained with rendered 3d model views. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2686–2694, 2015.

Fernando Wario, Benjamin Wild, Margaret Jane Couvillon, Raúl Rojas, and Tim Landgraf. Automatic methods for long-term tracking and the detection and decoding of communication dances in honeybees. *Frontiers in Ecology and Evolution*, 3:103, 2015.

Yu Xiang, Roozbeh Mottaghi, and Silvio Savarese. Beyond pascal: A benchmark for 3d object detection in the wild. In *IEEE Winter Conference on Applications of Computer Vision*, pp. 75–82. IEEE, 2014.

Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pp. 3320–3328, 2014.

7 Appendix


(a) Generated images

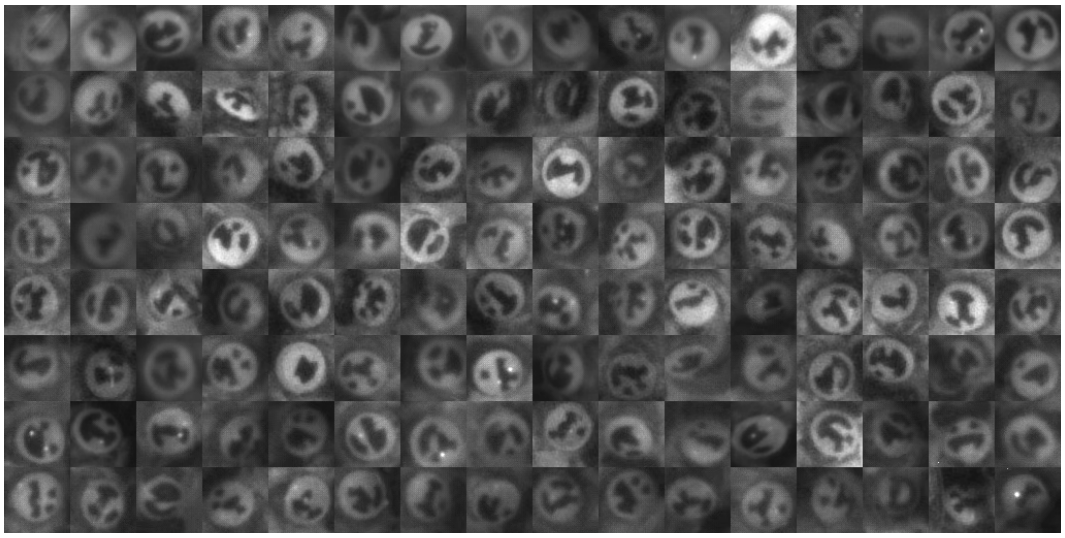

(b) Real images

Figure 10: Continuum visualization on the basis of the discriminator score: Most realistc scored samples top left corner to least realistc bottom right corner. Images with artifacts are scored unrealistic.

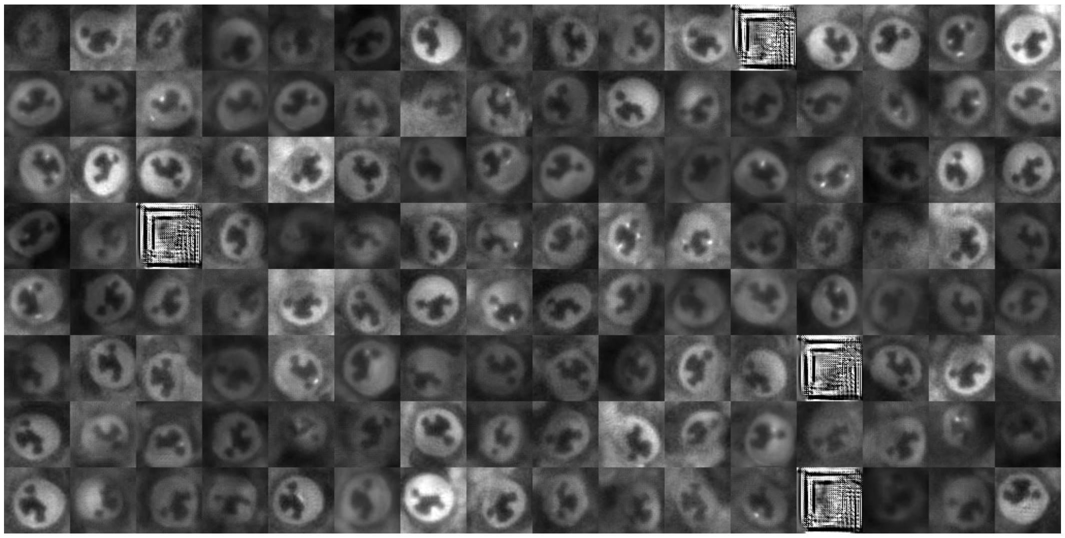


Figure 11: Images generated with the generator given a bit configuration

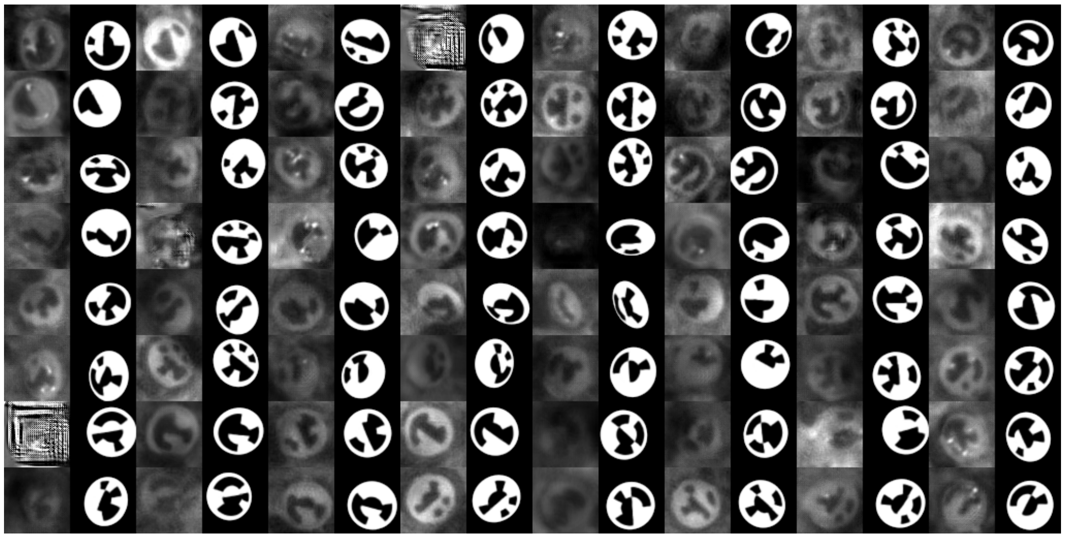


Figure 12: Correspondence of generated images and 3D model