DeceptionNet: Network-Driven Domain Randomization (Supplement)

Sergey Zakharov [*,†], Wadim Kehl [✣], and Slobodan Ilic [*,†]

[*] Technical University of Munich ✣ Toyota Research Institute † Siemens Corporate Technology

sergey.zakharov@tum.de, wadim.kehl@tri.global, slobodan.ilic@siemens.com

1. Network Architecture

Let C_k^B be a convolutional block composed of the following layers: 3×3 convolution with k filters, BatchNorm (BN), and ReLU activation function. Similarly, let UP_B be a decoding block made of: 2-factor upsampling transposed convolution, BatchNorm (BN), and ReLU.

DeceptionNet D: Using the defined nomenclature, the encoding part of the DeceptionNet can be described as: $C_{64}^B - MP - C_{128}^B - C_{128}^B - MP - C_{128}^B - C_{128}^B$; and its decoding part as: $UP_{64} - C_{64}^B - C_{64}^B - UP_{64} - C_{64}^B - C_{64}^B - M_B$. Where M_B is defined by the specific module type, and MP stands for a 2-factor max-pooling layer. Encoding blocks have skip connections concatenating the channels with the opposite decoding blocks. The visual representation of the DeceptionNet's architecture is depicted in Fig. 4.

Task Network T: In both cases, i.e., for MNIST classification and Cropped LineMOD classification and pose estimation, T follows a simple LeNet-like architecture. As for MNIST (see Fig. 1), the final layer outputs the 10D vector, whereas for Cropped LineMOD (see Fig. 3) there is a 11D classification output as well as 4D quaternion output.

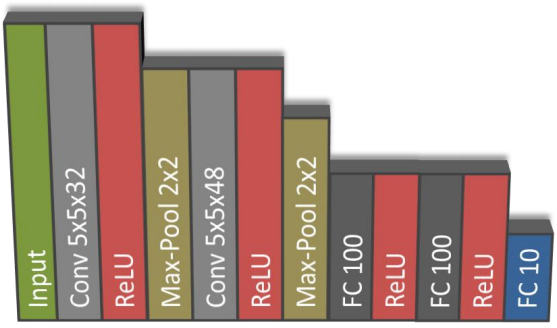

Figure 1: **MNIST Classifier:** Simple LeNet-like architecture, where 2 convolutional layers followed by ReLUs and max-poolings are finalized by 3 fully-connected layers.

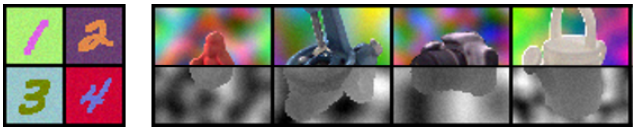

Figure 2: **Unguided samples:** We provide a sample of unguided augmentations for MNIST and LineMOD.

2. Unguided Randomization: BG Filling

One of the modalities we have compared our results with is unguided randomization that applies augmentations during the data preprocessing step. While using the same modules and constraints as our deception network, its perturbations are conditioned on random values instead of latent codes from the input.

Since our DeceptionNet is capable of generating very complex backgrounds, we have also used complex noise types for unguided randomization to make the comparison more fair (see Fig. 2). Apart from a uniform white noise, two additional noise types were used: Perlin noise [2] and cellular noise [3]. Sample frequencies were sampled from the uniform distribution $[0.0001, 0.1]$. Both noise types were generated using the open source FastNoise library [1].

Figure 3: **Cropped LineMOD Task Network:** Simple LeNet-like architecture followed by a dropout layer with a 50% rate and outputting both a class and pose vector.

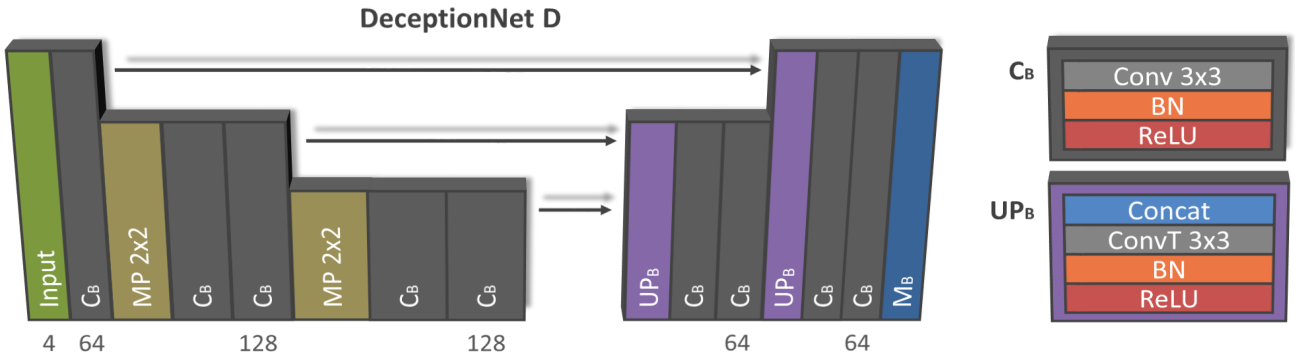

Figure 4: **DeceptionNet architecture**. Our network features a typical encoder-decoder architecture. The encoder part consists of 2 consecutive downsamplings followed by a sequence of convolutional blocks C_B. The decoder part shares a similar architecture for all presented augmentation modules. Arrows show the skip connections between blocks.

3. Additional Qualitative Results

In this section, we present additional output examples of the deception networks for Synthetic Cropped LineMOD and SYNTHIA test cases.

The LineMOD deception network uses all of the deception modules presented in the paper, whereas the SYNTHIA deception network uses three modules: light (L), elastic distortions (DS), and foreground noise (N). The sample outputs from each of the above-mentioned modules are shown in Fig. 6. Moreover, Fig. 5 demonstrates the output of the deception network during the training process. One can see that the output becomes increasingly more sophisticated for recognition by the task network.

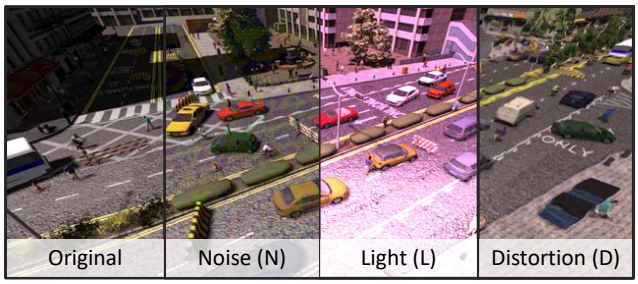

Figure 6: **Deception modules' transformations:** Augmentations applied for the SYNTHIA $\rightarrow$ Cityscapes scenario.

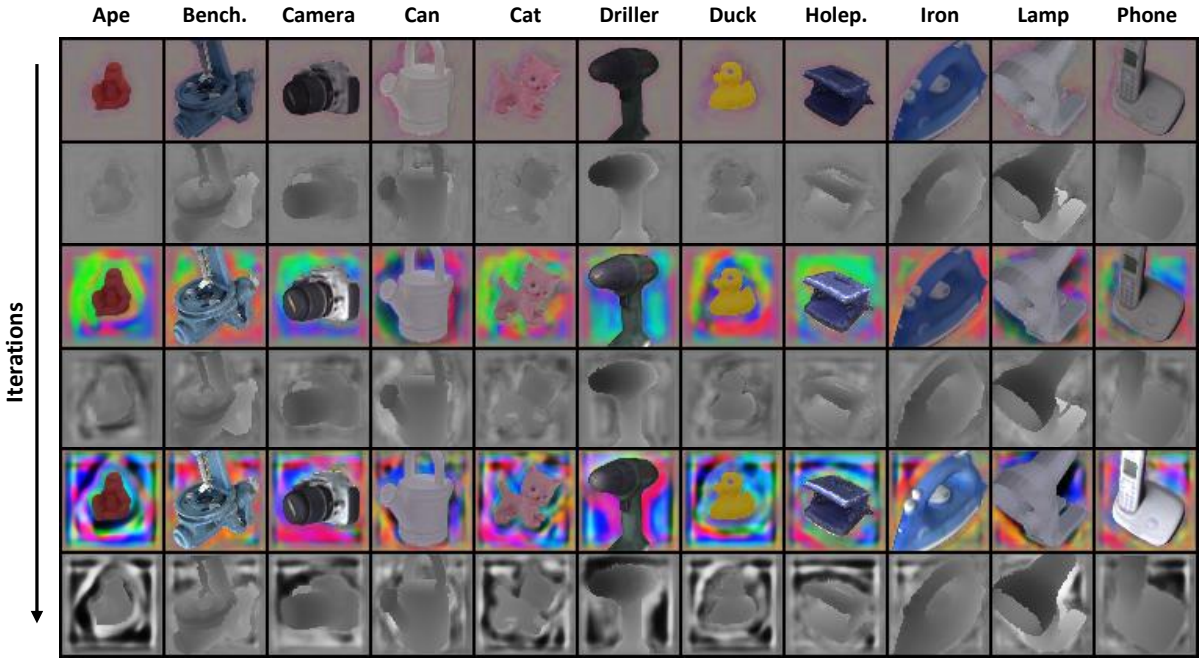

Figure 5: **Deceptive images $\mathbf{x}^{\mathrm{d}}$ over consecutive iterations:** The output becomes increasingly more complex for T.

References

[1] Jordan Peck. Fastnoise library. `https://github.com/Auburns/FastNoise`, 2016.

[2] Ken Perlin. Improving noise. In *ACM Transactions on Graphics (TOG)*, 2002.

[3] Steven Worley. A cellular texture basis function. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 291–294. ACM, 1996.