Supplementary Material: Task-Aware Synthetic Data Generation

A. Implementation Details

Our pipeline is implemented in python using the PyTorch deep-learning library. In the following subsections, we furnish relevant empirical details for our experiments on the different datasets in the manuscript.

A.1. Experiments on AffNIST data

In our experiments on the AffNIST benchmark, our synthesizer network generates affine transformations which are applied to MNIST images. These transformed images are then used to augment the MNIST training set and the performance of a handwritten digit classifier network trained on the augmented dataset is compared to an equivalent classifier trained on MNIST + AffNIST images. In the rest of the section, we give details about our experimental settings.

Data Pre-processing. For the AffNIST benchmark, our synthesizer network uses foreground masks from the MNIST training dataset. We pad the original 28×28 MNIST images by 6 black pixels on all sides to enlarge them to a 40×40 resolution. This is done to ensure that images generated by our synthesizer have the same size as the AffNIST images which also have a spatial resolution of 40×40 pixels.

Architecture of the Synthesizer Network. The synthesizer network takes as input a foreground image from the MNIST dataset and outputs a $6-$dimensional vector representing the affine parameters, namely the (i) angle of rotation, (ii) translation along the X$-$axis, (iii) translation along the Y$-$axis, (iv) shear, (v) scale along the X$-$axis, and (vi) scale along the Y$-$axis. We clamp these parameters to the same range used to generate AffNIST dataset. These parameters are used to define an affine transformation matrix which is fed to a Spatial Transformer module [1] alongside the foreground mask. The Spatial Transformer module applies the transformation to the foreground mask and returns the synthesized image. We attach the pytorch model dump for the synthesizer below.

```
Features(
  (BackBone): Sequential(
    (0): C2d(1, 10, ksz=(5, 5), st=(1, 1))
    (1): MaxPool2d(ksz=(3, 3), st=(2, 2))
    (2): BN2d(10, eps=1e-05, mmntm=0.1)
    (3): ReLU(inplace)
    (4): Dropout2d(p=0.5)
  )
  (FgBranch): Sequential(
    (0): C2d(10, 20, ksz=(3, 3), st=(1, 1))
    (1): ReLU(inplace)
    (2): BN2d(20, eps=1e-05, mmntm=0.1)
    (3): Dropout2d(p=0.5)
    (4): C2d(20, 20, ksz=(3, 3), st=(1, 1))
    (5): ReLU(inplace)
    (6): BN2d(20, eps=1e-05, mmntm=0.1)
    (7): Dropout2d(p=0.5)
  )
)
RegressionFC(
  (features): Sequential(
    (0): C2d(40, 20, ksz=(3, 3), st=(1, 1))
    (1): ReLU(inplace)
    (2): BN2d(20, eps=1e-05, mmntm=0.1)
    (3): Dropout2d(p=0.5)
    (4): C2d(20, 20, ksz=(3, 3), st=(1, 1))
    (5): ReLU(inplace)
    (6): BN2d(20, eps=1e-05, mmntm=0.1)
    (7): Dropout2d(p=0.5)
  )
  (regressor): Sequential(
    (0): Linear(in_f=1620, out_f=50, bias=True)
    (1): ReLU(inplace)
    (2): BatchNorm1d(50, eps=1e-05, mmntm=0.1)
    (3): Dropout(p=0.5)
    (4): Linear(in_f=50, out_f=20, bias=True)
    (5): ReLU(inplace)
    (6): BatchNorm1d(20, eps=1e-05, mmntm=0.1)
    (7): Dropout(p=0.5)
    (8): Linear(in_f=20, out_f=6, bias=True)
  )
)
```

The acronyms used in the pytorch model dump are described in Table. 1.

Architecture of the Target Classifier. For our experiments in Figure. 7, we use a target model with two convolutional layers followed by a dropout layer, and two linear layers. The pytorch model dump is attached below.

```
MNISTClassifier(
  (0): C2d(1, 10, ksz=(5, 5), st=(1, 1))
  (1): ReLU(inplace)
  (2): C2d(10, 20, ksz=(5, 5), st=(1, 1))
  (3): ReLU(inplace)
  (4): Dropout2d(p=0.5)
```

Acronym	Meaning
C2d	Conv2d
BN2d	BatchNorm2d
ksz	kernel_size
st	stride
pdng	padding
LReLU	LeakyReLU
neg_slp	negative_slope
InstNrm2D	InstanceNorm2d
mmntm	momentum
in_f	in_features
out_f	out_features

Table 1: Acronyms used within PyTorch model dumps.

```
  (5): Linear(in_f=980, out_f=50, bias=True)
  (6): ReLU(inplace)
  (7): Linear(in_f=50, out_f=10, bias=True)
)
```

For our experiments in Table. 1, we use the following architecture from [6], where Swish denotes the swish activation function from [3].

```
MNISTClassifierDeep(
  (0): C2d(1, 64, ksz=(5, 5), st=(2, 2))
  (1): BN2d(64, eps=1e-05, mmntm=0.1)
  (2): Swish(inplace)
  (3): C2d(64, 64, ksz=(5, 5), st=(2, 2))
  (4): BN2d(64, eps=1e-05, mmntm=0.1)
  (5): Swish(inplace)
  (6): C2d(64, 64, ksz=(5, 5), st=(2, 2))
  (7): BN2d(64, eps=1e-05, mmntm=0.1)
  (8): Swish(inplace)
  (9): C2d(64, 64, ksz=(5, 5), st=(2, 2))
  (10): BN2d(64, eps=1e-05, mmntm=0.1)
  (11): Swish(inplace)
  (12): Linear(in_f=576, out_f=50, bias=True)
  (13): Swish(inplace)
  (14): Linear(in_f=50, out_f=10, bias=True)
)
```

Training the Synthesizer Network. We use the ADAM optimizer with a batch-size of 1024 and a fixed learning rate of 10^{-3}. Xavier initialization is used with a gain of 0.4 to initialize the network weights. The synthesizer is trained in lock-step with the target model: we alternately update the synthesizer and target models. The weights of the target model are fixed during the synthesizer training. The synthesizer is trained until we find 500 hard examples per class. A synthesized image is said to be a hard example if $p - p^* > 0.05$ where $p*$ is the probability of it belonging to the ground truth class, and p is the maximum probability over the other classes estimated by the target model. We maintain a cache of hard examples seen during the synthesizer training and use this cache for training dataset augmentation. We observed from our experiments that the number of epochs required to generated 500 images per class increases over sleep cycles as the target model became stronger.

Training the Target Network. The original training dataset consists of MNIST training images. After each phase of synthesizer network training we augment the training dataset with all images from the cache of hard examples and train the target network for 30 epochs. The augmented dataset is used to fine-tune the classifier network. For fine-tuning, we use SGD optimizer with a batch-size of 64, 10^{-2} learning rate and a momentum of 0.5. For our experiments in Table. 1, we reduce the learning rate to 10^{-3} after 100 iterations of training the synthesizer and target networks.

A.2. Experiments on Pascal VOC data

In our experiments on the Pascal VOC person detection benchmark, we additionally employ a natural versus synthetic image discriminator to encourage the synthetic images to appear realistic. We use the SSD$-$300 pipeline from [2] as the target model.

Data Pre-processing. We resize all synthetic images to 300×300 pixels to be consistent with the SSD$-$300 training protocol. Background images for generating synthetic data are drawn from the COCO dataset, and foreground masks come from the VOC 2007 and 2012 trainval datasets. VOC datasets contain instance segmentation masks for a subset of trainval images. We use the ground truth segmentation masks and bounding box annotations to recover additional instance segmentation masks. We noticed that for about 10% images the segmentation and bounding box detections do not agree, therefore we visually inspected the instance masks generated and filtered out erroneous ones. Figure 1 shows some images from the VOC dataset where the segmentation and bounding box annotations disagree, resulting in erroneous instance masks.

The foreground segmentations are centered and normalized such that $\max(\text{height}, \text{width})$ of the segmentation bounding-box occupies at least 0.7 of the corresponding image dimension. We randomly pair background images with foreground instances while training the synthesizer network and during synthetic data generation.

Architecture of the Synthesizer Network. The synthesizer architecture is similar to the one used for AffNIST experiments with two enhancements: (i) we use the fully convolutional part of the the VGG$-$16 [4] network as the backbone, and (ii) we have an additional mid-level feature extraction subnetwork for the background image. This is described in the following model dump.

```
Features(
  (BackBone): Sequential(
    (0): C2d(3, 64, ksz=(3, 3), pdng=(1, 1))
    (1): BN2d(64, eps=1e-05, mmntm=0.1)
```

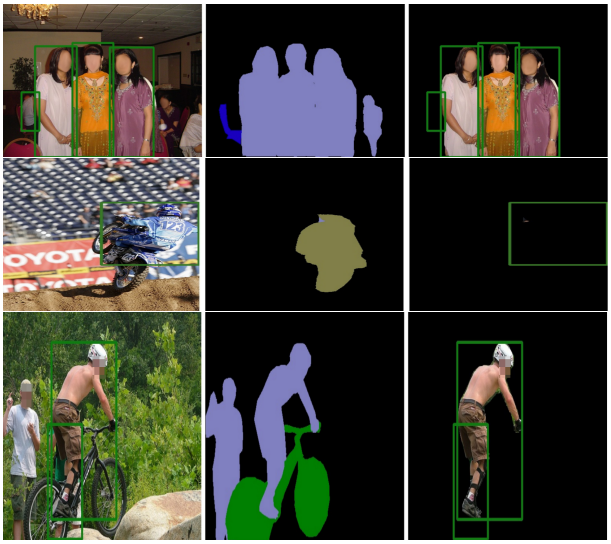

Figure 1: Disagreements between segmentation and detection annotations in the VOC dataset. Column 1 shows ground truth bounding boxes overlaid on images. Column 2 shows segmentation annotations. Column 3 shows generated instance masks: each green box corresponds to one instance.

```
    (2): ReLU(inplace)
    (3): C2d(64, 64, ksz=(3, 3), pdng=(1, 1))
    (4): BN2d(64, eps=1e-05, mmntm=0.1)
    (5): ReLU(inplace)
    (6): MaxPool2d(ksz=2, st=2)
    (7): C2d(64, 128, ksz=(3, 3), pdng=(1, 1))
    (8): BN2d(128, eps=1e-05, mmntm=0.1)
    (9): ReLU(inplace)
    (10): C2d(128, 128, ksz=(3, 3), pdng=(1, 1))
    (11): BN2d(128, eps=1e-05, mmntm=0.1)
    (12): ReLU(inplace)
    (13): MaxPool2d(ksz=2, st=2)
    (14): C2d(128, 256, ksz=(3, 3), pdng=(1, 1))
    (15): BN2d(256, eps=1e-05, mmntm=0.1)
    (16): ReLU(inplace)
    (17): C2d(256, 256, ksz=(3, 3), pdng=(1, 1))
    (18): BN2d(256, eps=1e-05, mmntm=0.1)
    (19): ReLU(inplace)
    (20): C2d(256, 256, ksz=(3, 3), pdng=(1, 1))
    (21): BN2d(256, eps=1e-05, mmntm=0.1)
    (22): ReLU(inplace)
    (23): MaxPool2d(ksz=2, st=2)
    (24): C2d(256, 512, ksz=(3, 3), pdng=(1, 1))
    (25): BN2d(512, eps=1e-05, mmntm=0.1)
    (26): ReLU(inplace)
    (27): C2d(512, 512, ksz=(3, 3), pdng=(1, 1))
    (28): BN2d(512, eps=1e-05, mmntm=0.1)
    (29): ReLU(inplace)
    (30): C2d(512, 512, ksz=(3, 3), pdng=(1, 1))
    (31): BN2d(512, eps=1e-05, mmntm=0.1)
    (32): ReLU(inplace)
    (33): MaxPool2d(ksz=2, st=2)
  )
  (FgBranch): Sequential(
    (0): C2d(512, 256, ksz=(3, 3), st=(1, 1))
    (1): BN2d(256, eps=1e-05, mmntm=0.1)
    (2): ReLU(inplace)
    (3): C2d(256, 256, ksz=(3, 3), st=(1, 1))
    (4): BN2d(256, eps=1e-05, mmntm=0.1)
    (5): ReLU(inplace)
    (6): C2d(256, 20, ksz=(3, 3), st=(1, 1))
  )
  (BgBranch): Sequential(
    (0): C2d(512, 256, ksz=(3, 3), st=(1, 1))
    (1): BN2d(256, eps=1e-05, mmntm=0.1)
    (2): ReLU(inplace)
    (3): C2d(256, 256, ksz=(3, 3), st=(1, 1))
    (4): BN2d(256, eps=1e-05, mmntm=0.1)
    (5): ReLU(inplace)
    (6): C2d(256, 20, ksz=(3, 3), st=(1, 1))
  )
)
RegressionFC(
  (features): Sequential(
    (0): C2d(40, 64, ksz=(5, 5), pdng=(2, 2))
    (1): ReLU(inplace)
    (2): BN2d(64, eps=1e-05, mmntm=0.1)
    (3): C2d(64, 64, ksz=(5, 5), pdng=(2, 2))
    (4): ReLU(inplace)
    (5): BN2d(64, eps=1e-05, mmntm=0.1)
  )
  (regressor): Sequential(
    (0): Linear(in_f=64, out_f=128, bias=True)
    (1): ReLU(inplace)
    (2): BatchNorm1d(128, eps=1e-05, mmntm=0.1)
    (3): Linear(in_f=128, out_f=128, bias=True)
    (4): ReLU(inplace)
    (5): BatchNorm1d(128, eps=1e-05, mmntm=0.1)
    (6): Linear(in_f=128, out_f=6, bias=True)
  )
)
```

Architecture of the Discriminator. Our discriminator is based on the discriminator architecture from [5].

```
Discriminator(
  (0): Sequential(
    (0): C2d(3, 64, ksz=(4, 4), st=(2, 2), pdng
    =(2, 2))
    (1): LReLU(neg_slp=0.2, inplace)
  )
  (1): Sequential(
    (0): C2d(64, 128, ksz=(4, 4), st=(2, 2), pdng
    =(2, 2))
    (1): InstNrm2D(128, eps=1e-05, mmntm=0.1,
    affine=False, track_running_stats=False)
    (2): LReLU(neg_slp=0.2, inplace)
  )
  (2): Sequential(
    (0): C2d(128, 256, ksz=(4, 4), st=(2, 2),
    pdng=(2, 2))
    (1): InstNrm2D(256, eps=1e-05, mmntm=0.1,
    affine=False, track_running_stats=False)
    (2): LReLU(neg_slp=0.2, inplace)
  )
  (3): Sequential(
    (0): C2d(256, 512, ksz=(4, 4), st=(1, 1),
    pdng=(2, 2))
    (1): InstNrm2D(512, eps=1e-05, mmntm=0.1,
    affine=False, track_running_stats=False)
    (2): LReLU(neg_slp=0.2, inplace)
  )
  (4): Sequential(
```

```
    (0): C2d(512, 1, ksz=(4, 4), st=(1, 1), pdng
    =(2, 2))
  )
  (5): AvgPool2d(ksz=3, st=2, pdng=[1, 1])
)
```

Training the Synthesizer, Discriminator and Target Networks. The synthesizer is trained in lock-step with the discriminator and the target models: the discriminator and target model weights are fixed and the synthesizer is trained for 1000 batches. The synthesizer is then used to generate synthetic images: we do a forward pass on 100 batches of randomly paired foreground and background images and pick composite images which have a target estimated probability of less than 0.5. These hard examples are added to the training dataset. The weights of the synthesizer network are then fixed and the discriminator and target models are trained for 3 epochs over the training set. This cycle is repeated 5 times.

The VGG-16 backbone of the synthesizer is initialized with an ImageNet pretrained model. The synthesizer is trained using the ADAM optimizer with a batch-size of 16 and a fixed learning rate of $1e-4$ is used. The weights of the discriminator are randomly initialized to a Normal distribution with a variance of 0.02. The discriminator is trained using the ADAM optimizer with a batch size of 16 and a fixed learning rate of $1e-4$ is used. We initialize the SSD model with the final weights from [2]. The SSD model is trained using the SGD optimizer, a fixed learning rate of $1e-5$, momentum of 0.9 and a weight decay of 0.0005.

Custom Spatial Transformer. The image resizing during data pre-processing step and the spatial-transformer module both involve bilinear interpolation which introduces quantization artefacts near the segmentations mask edges. To deal with these artefacts we customize the Spatial Transformer implementation. More specifically, we remove these artefacts by subtracting a scalar($\mu = 1 - 10^{-7}$) from the segmentation masks and applying ReLU non-linearity. The result is normalized back to a binary image. Gaussian blur is applied to the resulting mask for smooth transition near edges before applying alpha blending.

A.3. Experiments on GMU Kitchen data

Our experiments on the GMU data use the same synthesizer architecture and training strategy as described in Section. A.2 with one change: we do not use the discriminator in these experiments. We did not see noticeable improvements in performance with the addition of the discriminator in these experiments.

B. Qualitative Results

Qualitative results follow on the next page.

Figure 2: Qualitative results for Synthetic Data Generation. **Row-1**: Here we show the synthetic images generated by our synthesizer based on feedback from the baseline SSD trained on the VOC 2007-2012 dataset. These images evoke misclassifications from the baseline SSD primarily because they present human instances in unforeseen/unrealistic circumstances. **Row-2**: SSD failures on VOC 2007 test images after finetuning the baseline SSD with composite images such as those shown in Row-1. We notice the the three failure cases are a) person instances at small scales, b) horizontal pose, and c) severe occlusion. **Row-3**: Synthetic images generated by our synthesizer after feedback from the finetuned SSD. We notice that our synthesizer now generates small scales, horizontal pose and severely truncated instances.

Figure 3: Qualitative improvements on the GMU Kitchen benchmark. Green and red bounding boxes denote correct and missing detections respectively.

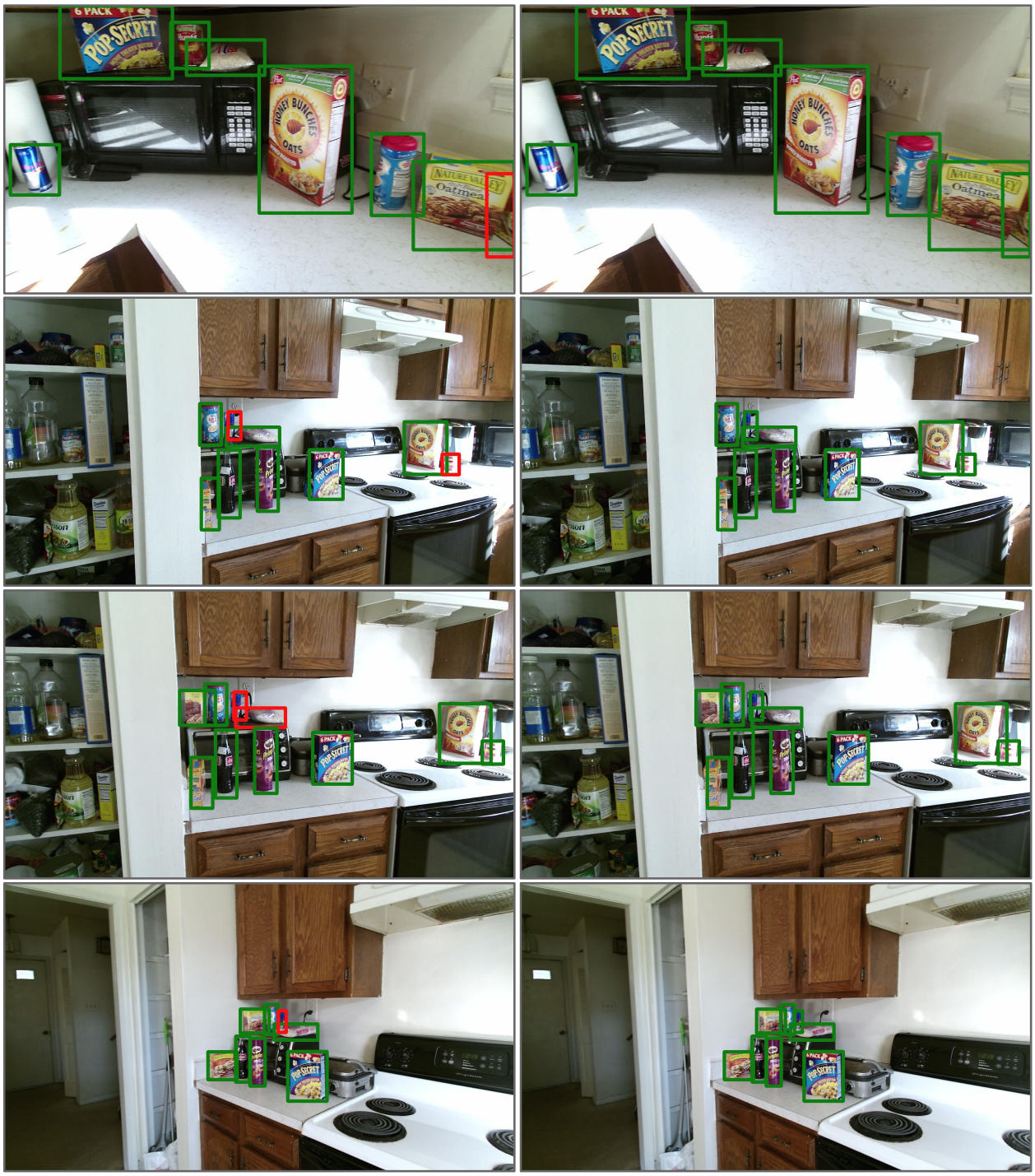

Figure 4: More qualitative improvements on the GMU Kitchen benchmark. Green and red bounding boxes denote correct and missing detections respectively.

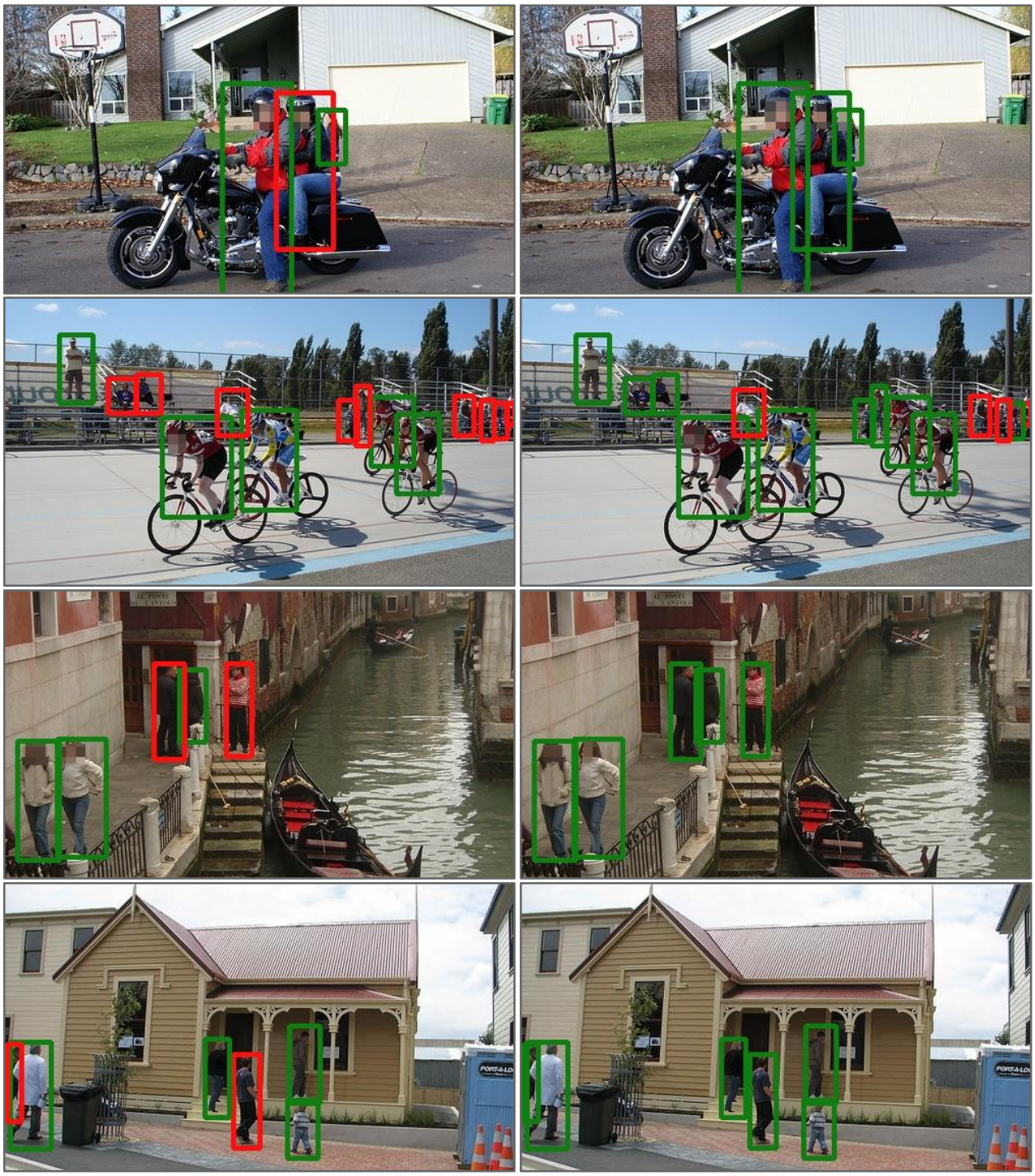

Figure 5: Qualitative improvements on the Pascal VOC person detection benchmark. Green and red bounding boxes denote correct and missing detections respectively.

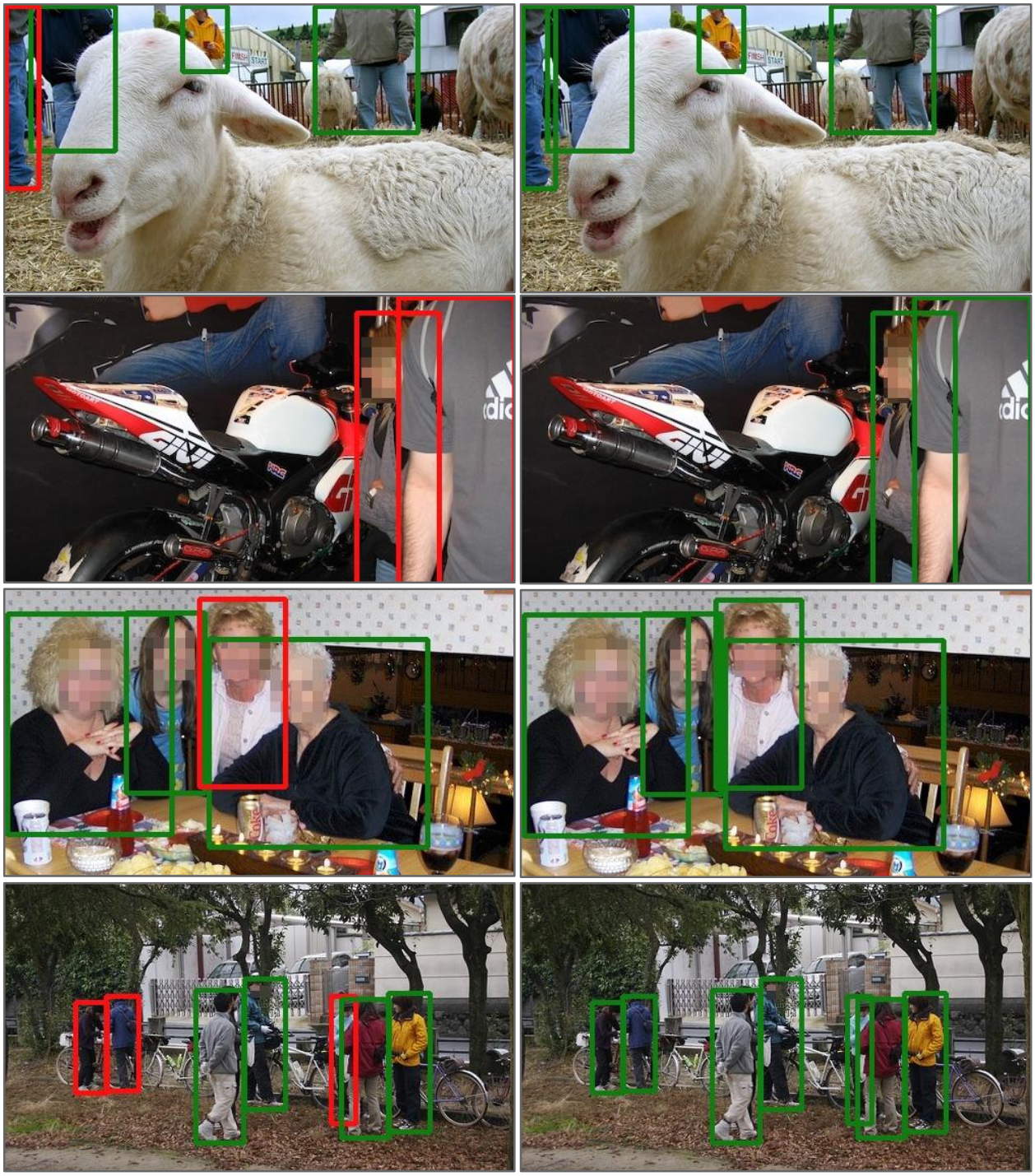

Figure 6: More qualitative improvements on the Pascal VOC person detection benchmark. Green and red bounding boxes denote correct and missing detections respectively.

References

[1] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu. Spatial transformer networks. *CoRR*, abs/1506.02025, 2015. 1

[2] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015. 2, 4

[3] P. Ramachandran, B. Zoph, and Q. V. Le. Searching for activation functions. *CoRR*, abs/1710.05941, 2017. 2

[4] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 2

[5] T. Wang, M. Liu, J. Zhu, A. Tao, J. Kautz, and B. Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. *CoRR*, abs/1711.11585, 2017. 3

[6] Y. Zhao, Y. Tian, W. Shen, and A. Yuille. Towards resisting large data variations via introspective learning. 2018. 2