Supplementary Material: Towards Automated Testing and Robustification by Semantic Adversarial Data Generation

Rakshith Shetty[1], Mario Fritz[2], and Bernt Schiele[1]

[1] Max Planck Institute for Informatics, Saarland Informatics Campus
{rshetty,schiele}@mpi-inf.mpg.de
[2] CISPA Helmholtz Center for Information Security, Saarland Informatics Campus
fritz@cispa.saarland

In this supplementary material we provide additional material to support our work, including architecture details, hyperparameters of adversarial optimization and more visual examples of synthesized adversaries[3]. Section 1 provides further details about the synthesizer architecture, left out from the main paper due to space constraints. We also show more examples of appearance interpolations produced by our model in Figure 2. Section 2 discusses hyperparameters used during semantic adversarial appearance optimization and provides more visual examples of intermediate images produced during adversarial optimization in Figure 3. Section 3 presents more details of the human study with qualitative examples and an analysis of the correlation of the typicality rating with the detector performance. Section 4 presents qualitative examples of different failure modes discovered by the semantic adversary in Figure 7 and discusses them.

1 Synthesizer Architecture

Synthesizer model consists for three sub networks 1) Shape encoder, 2) Appearance encoder and 3) Decoder Network. Now we will discuss the architecture details of each of these networks.

Shape Encoder. This network has a UNet [2] structure with a series of downsampling layers, residual bottleneck layers followed by bilinear upsampling layers as shown in Table 1. The idea is to downsample and encode the input image into low resolution feature vectors, using which part segmentation map is generated by slowly upsampling these features. The downsampling bottleneck uses convolution and pooling layers ImageNet pretrained VGG-19 model available in the PyTorch Library. We retain 5 blocks from the VGG-19 architecture, and keep only 4 max pooling layers to obtain the feature maps of 8x8 resolution and 512 dimensions. One-hot encoding of the class label is concatenated to above and provided as additional conditioning to the first Residual block. This results in input features of 8x8 resolution and 592 dimensions as seen second row of Table 1. Input image is always 128x128 resolution and the output part segmentation map is 64x64 resolution. We set the number of parts to 15. We experimented with

[3] Code will be made available in github.com/rakshithShetty/SemanticAdversary

Num	Layer type	Inp Spat Dim	Inp Dim	Out Dim	Shortcut
1-5	VGG-19 backbone	128x128	3	512	-
6	Residual Block	8 x 8	592	592	-
7	Upsample	8 x 8	592	592	-
8	Residual Block	16 x 16	1184	336	4
9	Upsample	16 x 16	336	336	-
10	Residual Block	32 x 32	672	208	3
11	Upsample	32 x 32	208	208	-
12	Conv 1x1	64 x 64	208	16	-

Table 1: The architecture for the Shape encoder. The description of Residual Blocks is presented in Table 2

Num	Layer type	Spat Dim	Inp Dim	Out Dim	Shortcut
-	Input	w x h	d_i	-	-
1	Conv2D 1x1	w x h	d_i	d_o	-
2	Conv2D 1x1	w x h	d_i	$d_o/2$	-
3	BatchNorm2D + ReLU	w x h	$d_o/2$	$d_o/2$	-
4	Conv2D 3x3	w x h	$d_o/2$	$d_o/2$	-
5	BatchNorm2D + ReLU	w x h	$d_o/2$	$d_o/2$	-
6	Conv2D 1x1	w x h	$d_o/2$	d_o	-
7	BatchNorm2D + ReLU	w x h	d_o	d_o	-
-	Output	w x h	d_o	d_o	1

Table 2: The architecture for the Residual block with input feature dimension d_i and output feature dimension d_o. Note here that layer 2 takes the activations at the Input layer and not the output of layer 1. Layer 1 is used to apply the shortcut when the input and the output dimensions (d_i and d_o) are different.

part bottlenecks of 4,8,15 and 32 dimensions, and found that the model performs well for either 8 or 15, with 15 having better image quality and 8 having slightly better interpolation performance.

Appearance Encoder. Table 3 shows the architecture of our Appearance encoder. It has a similar UNet structure to the Shape encoder. The downsampling is done again using the VGG-19 backbone. Instead of residual block, the appearance encoder just uses Conv Layers with instance norm. It maps the input image of 128x128 resolution to a 64x64 resolution spatial appearance codes each of 256 dimensions. At layer 4, the generated part segmentation heatmap (15 dimensions) is also concatenated with the feature vectors from the VGG backbone to get an input feature vector of 223 dimensions. This gives appearance encoder the access to the spatial part map generated by the Shape encoder and helps it extract appearance codes for the right parts.

Num	Layer type	Inp Spat Dim	Inp Dim	Out Dim	
1-3	VGG-19 backbone	128x128	3	512	-
4	Conv2D 3x3 stride 2	64 x 64	223	256	-
5	InstNorm + LeakyRelu	32 x 32	256	256	-
6	Conv2D 3x3 stride 1	32 x 32	256	256	-
7	InstNorm + LeakyRelu	32 x 32	256	256	-
8	Conv2D 3x3 stride 1	32 x 32	256	256	-
9	Instnorm + LeakyRelu	32 x 32	256	256	-
10	Upsample	32 x 32	256	256	-
11	Conv2D 3x3 stride 1	64 x 64	512	256	5
12	InstNorm + LeakyRelu	64 x 64	256	256	-

Table 3: The architecture for the Appearance encoder. Note that the layers of the VGG backbone are shared between the appearance and the shape encoder

Decoder Network. Figure 1 shows the architecture of our Decoder network. At the first layer, it takes as input a downsampled binary part segmentation map (resolution 4x4) along with the one-hot class label of the object to be generated. This is followed by a series of residual block and spatially adpative instance norm (SPADE) layers. At each residual block, apart from the features from the previous layers, part segmentation map of appropriate resolution is also input. The residual block used here has the same structure as described in Table 2, except that batchnorm layers are skipped. This is because the normalization is done in the SPADE layers. The SPADE layer [1] helps input the appearance codes to the decoder. The apperance codes modulate the activations by controlling the mean and variance applied to the features after instance normalization. Per part appearance codes are converted to a spatial map by copying them to the corresponding part locations using the binary part map. This spatial appearance maps are passed through a 1x1 convolutional layers to obtain mean and variance of the decoder feature vectors at the corresponding layer. The mean and variance is different at each location based on the appearance code at that location. We found in our experiments that this spatial adaptive normalization is key to obtain high quality synthesis when dealing with big datasets like COCO with large appearance diversity.

Figure 2 presents more examples comparing appearance interpolations produced with our model and the Gaussian bottleneck model on the COCO dataset. We can see that using the binary part segmentation bottlneck helps preserve the spatial structure much better. Lot of part details which get blurred in the gaussian bottleneck model, is well preserved in our synthesizer.

2 Semantic Adversarial Optimization

Our semantic adversarial optimization of appearance is performed by optimizing the interpolation co-efficients to maximize the detector loss. This optimization is

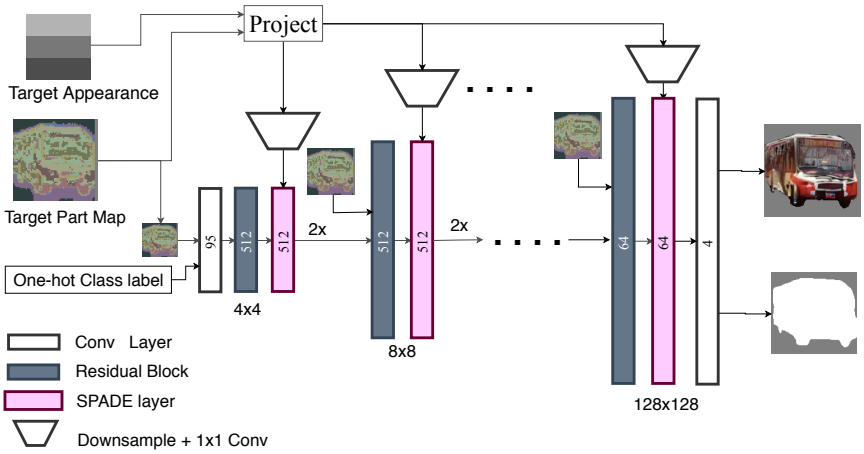

Fig. 1: Architecture of the decoder network

performed using signed gradient descent. That is gradients are computed for the interpolation co-efficients and they are updated in the direction of these gradients by using a fixed step size.

$$a_k^n = a_k^n + \gamma * \text{sign}\left[\frac{\partial \mathcal{L}_{\text{det}}}{\partial a_k^n}\right] \tag{1}$$

We found that signed gradient descent converges faster than than using the full gradients. This is because the detector objectness and class confidences is often very high and in the saturation region. This causes very small gradient magnitudes, causing optimization to often get stuck. Using sign of the gradient and fixed step size γ avoids this problem. Our optimization is run for 10 steps with $\gamma = 0.5$. At the end of 10 steps, the configuration which gives the highest detector loss is picked.

For test and training data generation, appearance of a randomly selected instance from the image is adversarially optimized to fool the detector. When editing two objects, if there is only one valid object in the image, we randomly synthesize a second object at a location which does not overlap with other objects, and optimize it's appearance. This way there are always at-least two object being edited in this configuration.

Figure 3 shows more qualitative examples which illustrate the intermediate outputs during appearance optimization. We can see that through appearance manipulation, how in the eyes of the object detector a person turns into a parking meter (row 1) or disappears (row 2), airplanes appear as kites (row 3 and 4), a car turns into a truck (row 5), a bear turns into a cow and a bannana now appears as toothbrush.

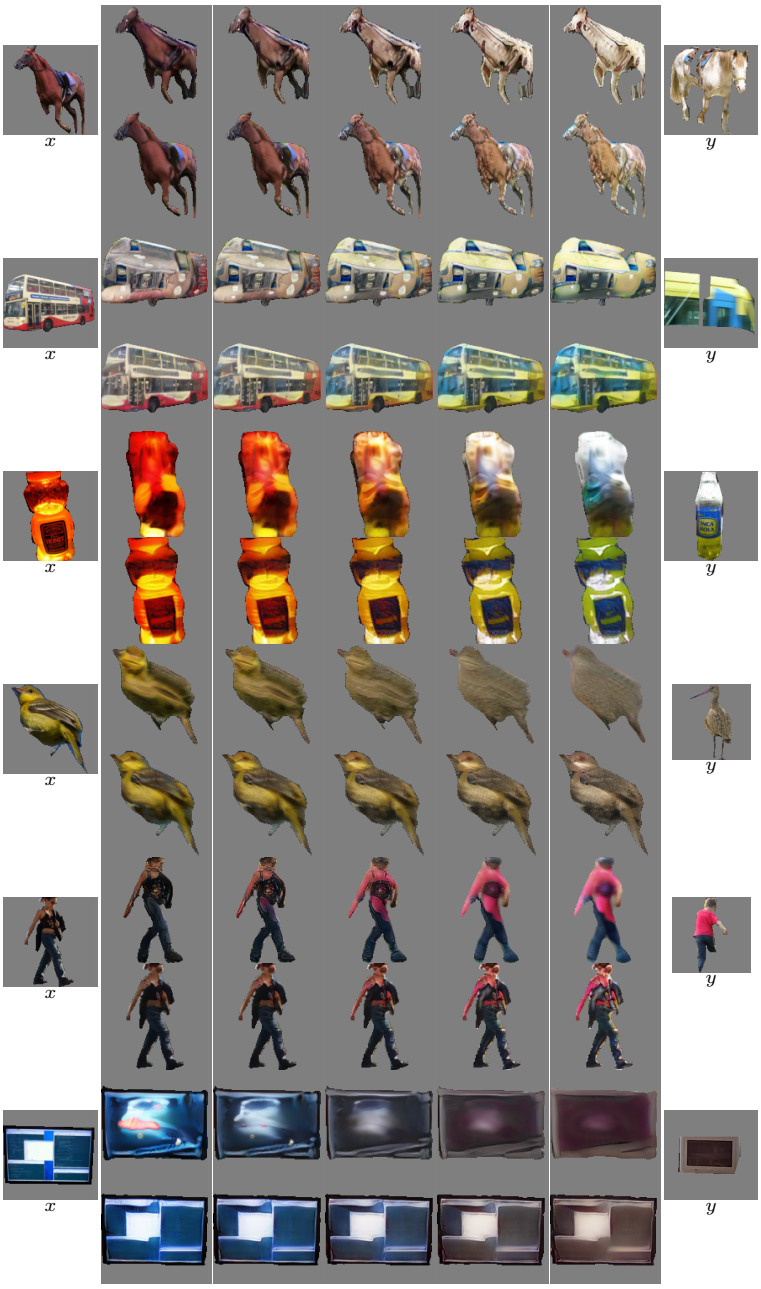

Fig. 2: Appearance interpolations with our binary part segmentation bottleneck (2nd rows) and the Gaussian bottleneck model (1st rows). The objects are generated using shape code from x and appearance by interpolating vectors from x and y

Fig. 3: Illustrating the intermediate results when optimizing the appearance to fool the detector.

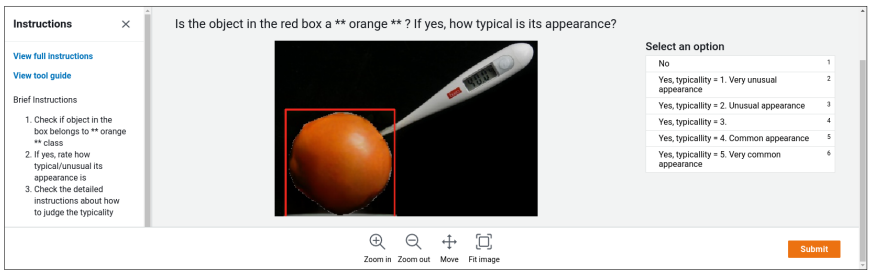

Fig. 4: Screenshot of the interface presented to the human observers in our study.

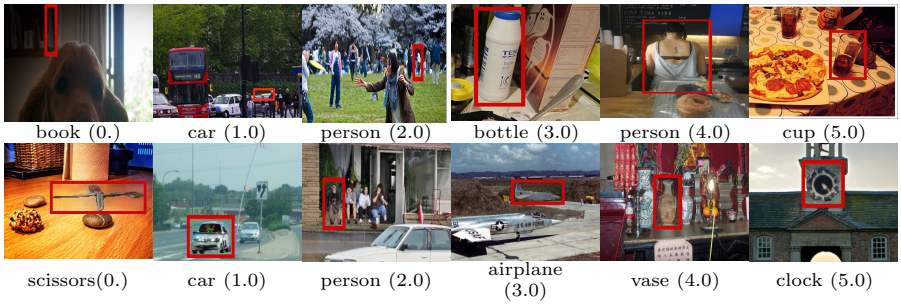

Fig. 5: Qualitative examples illustrating the spectrum of human typicality ratings. Top row shows real instances and bottom row shows adversarially semantically edited instances. Each instance is annotated with the label and the average typicallity rating. Typicallity rating of 0 are the cases where human observers judged that the object does not belong to the specified class.

3 Human study analysis

We conduct a human study to understand if the semantic adversary is able to preserve the edited object within the true class boundary. The object is within the true class boundary if human observers agree that the shown instance belongs to the designated class. Apart from keeping the edited appearance within the class boundary, we also want the semantic adversary to generate hard samples which are rare/novel combinations of appearances seen the training data. To measure this we ask human observers to also rate how typical the shown object instance is to that class. Each human observer is shown an image containing an object highlighted with a red bounding box. They are asked to first judge if the object belongs the specified class. If it does belong to the specified class, they are asked to rate how typical the object instance is from 1 to 5, with 1 being very rare and 5 being very typical. To prime the users, we present an example in the instructions. Detailed instructions given to the participants are as follows.

1. Look closely at the object marked by the red bounding box.
2. If you can't spot box, it might be surrounding the whole image. Look closely at the borders.
3. First determine if this object is a *class*.
4. If it does belong to the specified category, rate how typical is the apperance of this object for this category.
5. Typicallity rating goes from 1 to 5 with 1 being very unsual appearance to 5 being very common/typical.
6. For example a clearly visible yellow colored banana is considered very typical (rating 5).
7. Similarly, a green banana could be considered rarer (rating 3), whereas a purple/rainbow colored banana should be considered very rare/atypical (rating 1) .

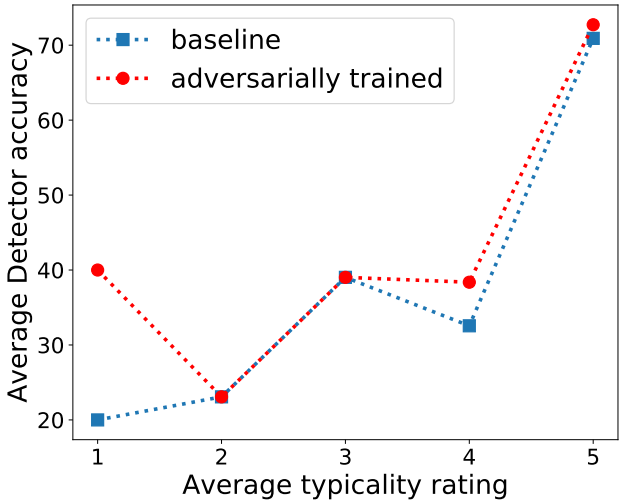

Fig. 6: Comparing the accuracy of baseline and adverarially trained detector performance across typicality rating of real test images. Here the typicallity rating shown is the mean typicality rating given to the image by the three annotators.

8. The data contains a mix of real/synthetic images. However do not base your judgement solely based on if the image is real/synthetic, but if the object depicted matches your idea of a typical object of the prescribed category.

The interface of the user study is shown in Figure 4. To get a baseline for the rating we also get same number of real instances annotated. These are mixed in the order in which the instances are shown to an annotator is randomized. Each instance is shown to three unique annotators and their responses are collected. Majority vote is taken to obtain the results presented in table 2 of the main paper. We saw from this table that in most of the instances (93%), human observers agree that the semantically edited object preserves the class label. Figure 5 show some qualitative examples to illustrate how the human observers rated the typicality of different instances. We see from the real examples that clearly visible instances are rated with high typicality, whereas the smaples which are unusual (bottle with rating of 3.0) or difficult to see are rated with lower typicality. Simlarly with synthetic samples, instances which have unusual appearance (airplane, person, car) are rated lower.

Correlation of typicality with detector performance. Figure 6 shows the average typicality rating of a real instance compared to the detector performance on these instances. We measure the detector performance by using the accuracy of the detector (confidence threshold=0.5) on these instances. Looking at the baseline detector performance (blue curve) we clearly see a strong correlation between detector performance and typicality. The baseline detector has a high accuracy of $\tilde{7}0\%$ on very typical instances (rating=5), which drops down to 25% and 20% on rare instances with typicality=2 and 3 respectively. This might partly explain the

drop in performance we see with the semantic adversarial examples. Since our semantic adversarial examples have lower typicality on average compared to the real samples (see figure 7 in the main paper), detector fails to recognize them and drops performance. Figure 6 also shows that adversarial training helps improve the perfromance on less typical instances. We see that accuracy increases by 20% on the least typical instances and by 5% on instances with typicality rating=4.

4 Qualitative examples of failure modes

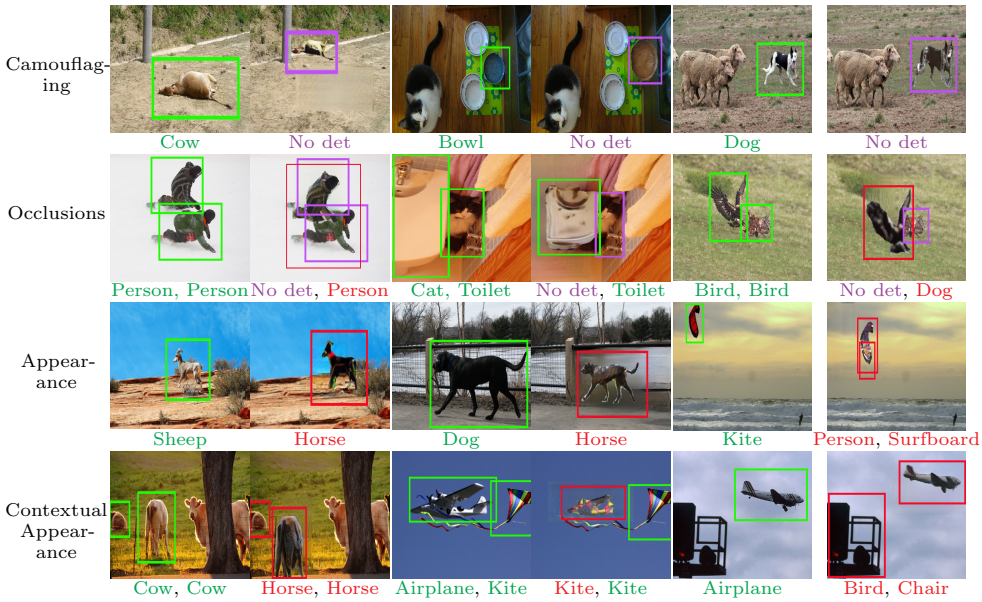

Fig. 7: Qualitative examples of the failure cases discovered by our semantic adversary. **Green** boxes are correct detections, **purple** boxes indicate missed detections and **red** boxes show the misclassified objects. Only relavant detections are marked.

Figure 7 presents more examples of the four types of error cases synthesized by our semantic adversarial optimization.

- **Camouflaging** - First row of Figure 7 shows examples where appearance of the object is altered to be less distinct and blending with the background, causing missed detections. We see appearance of a cow, a bowl and a dog being camouflaged, causing detector to not see these objects.
- **Occlusion** - Second row of Figure 7 show more examples of missed detections due to occlusions created by the edited object by moving closer to other objects in the image. We see a person occluding another, a cat getting

occluded by a toilet, and bird occluding another, all causing the detector to miss the occluded objects.
- **Appearance** - Third row in Figure 7 shows examples of the new object appearance confusing the detector into misclassifying the object. We see cases of sheep and dog being misclassified as horse and kite being misclassified as a person with a surfboard.
- **Contextual Appearance** - Last row of Figure 7 shows the examples where with a change in appearance of the edited object causes contextual bias to override and cause misclassification. While the edited airplane appears as a kite in the middle column, in the other two examples objects which are not modified are also mis-classified due to change in prediction on the edited object. In the first column edited cow is classified as a horse, which also affects the detection of nearby cow, now also misclassified as a horse. Similarly in the last column, edited airplane appears as bird to the detector, causing it to detect the nearby structure as a chair.

5 Comparison to pixel-level adversarial attacks

More commonly studied form of adversarial attacks are pixel-level attacks, which break detectors with small changes not visible to the human eye. In contrast, our work breaks the detector with visible appearance changes which does not alter the class to a human viewer. This helps us discover different failure modes with large appearance changes compared to a Lp norm restricted attacker. However, it is useful to compare the effectiveness of our semantic adversarial attack against the pixel-level L_p norm bounded attacks. It would also be interesting to see if training against our semantic adversary has any benefits to robustness to pixel-level adversarial attacks.

To study this, we apply the standard L_∞ norm bounded PGD attack on objects (restricted to single object box) on the baseline detector as well as our best coco detector (SA#2 x4). We restrict the $L\infty$ attack to a single object for fair comparison to our attack, which also affects only the object area.

The results are presented in Table 4. From these results, we see that our semantic adversarial training has a small but consistent improvement against the L_∞ adversary compared to the baseline detector. Repeating the same attack on image-level with epsilon=8/255 shows the same trend with baseline dropping to mAP=11.4 compared to mAP=13.4 of our model.

The above results also show that our semantic adversary has similar effectiveness as the object-level L_∞ attack, with $epsilon = 8/255$. Our attack on a single object shown in Table 1 of the main paper, drops the detector mAP to the same level as the object-level attack with e=8/255 (mAP = 58.7 vs 59.5 with our attack). Ofcourse, with larger ϵ, L_∞ attack is more effective. **Training with adversary.** We also compare the effectiveness of training with standard adversarial data to our approach in Table 5. Adding the L_∞ (with $\epsilon = 8/255$) adversarial data to the coco training set, we train with the same settings as our models. Performance of the resulting model is shown in Table 5. We see that

Epsilon	$\epsilon = 2/255$	$\epsilon = 4/255$	$\epsilon = 8/255$	$\epsilon = 32/255$
Base+FT	76.2	70.1	58.7	44.0
SA#2 x4	**77.3**	**71.9**	**59.6**	**44.5**

Table 4: Results of attacking l_∞ attacks on the baseline object detector and our best detector trained with semantic adversarial data. There is a small but consistent improvement in the robustness of our SA#2 x4 detector against $L\infty$ adversary.

Testset	COCO	VOC	Unrel
Baseline	46.2	66.9	38.8
L_∞	46.3	66.8	38.4
SA	**46.7**	**67.3**	**39.4**

Table 5: Comparing the effect of training with images produced by L_∞ based attack to training with our semantic adversarial data. For fair comparison, the version of our model only attacking a single object is compared to the $L\infty$ attack on a one object.

training with adversary does not improve generalization and under-performs our model *SA*. This observation is inline with similar results in literature [3].

Together, these results show that our semantic adversarial training provides better generalization than training on Lp norm adversary. It also provides a small boost in robustness to Lp norm attacks

References

1. Park, T., Liu, M.Y., Wang, T.C., Zhu, J.Y.: Semantic image synthesis with spatially-adaptive normalization. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2019) 3
2. Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In: International Conference on Medical image computing and computer-assisted intervention. pp. 234–241. Springer (2015) 1
3. Tsipras, D., Santurkar, S., Engstrom, L., Turner, A., Madry, A.: Robustness may be at odds with accuracy. In: Proceedings of the International Conference on Learning Representations (ICLR) (2019) 11