Supplementary Material
Triggering Failures: Out-Of-Distribution detection by learning from local adversarial attacks in Semantic Segmentation

1. Implementation details & hyper-parameters

For our implementation, we use Pytorch[1] and will release the code after the review. We share each hyper-parameter in Table 1. We train ObsNet with SGD with momentum and weight decay for at most 50 epochs using early-stopping. ObsNet is not trained from scratch as we initialize the weights with those of the segmentation network. We also use a scheduler to divide the learning rate by 2 at epoch 25 and epoch 45. We use the same data augmentation (*i.e.* Horizontal Flip and Random Crop) for training of the segmentation network and as well as for ObsNet. As there are few errors in the training of ObsNet, we increase the weight of positive examples in the loss contribution (Pos Weight in Table 1).

Params	CamVid	StreetHazards	Bdd Anomaly
Epoch	50	50	50
Optimizer	SGD	SGD	SGD
LR	0.05	0.02	0.02
Batch Size	8	6	6
Loss	BCE	BCE	BCE
Pos Weight	2	3	3
LAA shape	rand shape	rand shape	rand shape
LAA type	$\min_{p(c)}$	$\max_{p(k\neq c)}$	$\max_{p(k\neq c)}$
epsilon	0.02	0.001	0.001

Table 1: Hyper-parameters to train ObsNet on the different datasets.

2. Ablation on ObsNet architecture, ϵ and LAA

One contribution of our work is the ablation we do on the architecture of the observer network compared to previous methods. We highlight that the skip connections are essential for reaching best performance. For the smaller architecture, instead of keeping the same architecture as the segmentation network, we design a smaller variant: a convolutional network with three convolutional layers and a fully connected layer. This architecture mimicks the one used by ConfidNet [10].

[1] A Paszke et al., *PyTorch: An Imperative Style, High-Performance Deep Learning Library*, NIPS 2019

	Type	fpr95tpr ↓	AuPR ↑	AuRoc ↑	ACE ↓
	MC Dropout	49.3	97.3	90.1	0.463
	ObsNet base	54.2	97.1	89.1	0.396
$\min_{p(c)}$	all pixels	53.2	97.1	89.5	0.410
	sparse pixels	61.1	97.1	89.2	**0.387**
	class pixels	45.6	97.3	90.3	0.428
	square patch	47.4	97.3	90.1	0.461
	rand shape	**44.6**	**97.6**	**90.9**	0.446
$\max_{p(k\neq c)}$	all pixels	51.9	97.1	89.6	0.405
	sparse pixels	54.2	97.2	89.6	**0.374**
	class pixels	46.8	97.2	89.9	0.432
	square patch	45.5	**97.4**	90.5	0.464
	rand shape	**44.6**	**97.4**	**90.6**	0.446

Table 2: Ablation on adversarial attacks.

Method	fpr95tpr ↓	AuPR ↑	AuRoc ↑	ACE ↓
Smaller archi.	60.3	95.8	85.3	0.476
w/o skip	81.3	92.0	74.4	0.551
w/o input img	57.0	96.9	88.2	0.455
w/o pretrain	55.7	96.9	88.7	0.419
ObsNet full	**54.2**	**97.1**	**89.1**	**0.396**

Table 3: Ablation ObsNet without LAA training.

Next, we outline most of the experiments we make on LAA. First, there are two different kinds of setups, we can either minimize the prediction class (*i.e.* $\min_{p(c)}$) or maximize instead a different class (*i.e.* $\max_{p(k\neq c)}$), with $p = Seg(x)$ the class vector, $c = \max_p$ the maximum class prediction and k a random class. Then, we attack with five different strategies: all pixels in the image, random sparse pixels, the area of a random class, all pixels in a square patch and all pixels in a random shape. We show in Table 2 the complete results on CamVid ODD. We can see that that random shape is the most effective. We use the FSGM because it's a well-known and easy-to-use adversarial attack.

Method	fpr95tpr ↓	AuPR ↑	AuRoc ↑	ACE ↓
Softmax [25]	61.9	96.5	84.4	0.480
Void [6]	79.9	90.7	67.3	0.504
MCDA [1]	65.8	96.3	83.1	0.440
Temp. Scale [19]	61.9	96.6	84.6	**0.302**
ODIN [32]	58.3	97.2	87.9	0.478
ConfidNet [10]	<u>52.2</u>	97.5	<u>88.6</u>	0.412
Gauss Pert. [15,41]	60.2	96.8	85.6	0.497
Deep Ensemble [30]	55.3	97.5	88.1	<u>0.343</u>
MC Dropout [17]	52.5	<u>97.9</u>	88.5	0.443
ObsNet + *LAA*	**47.7**	**98.1**	**90.3**	0.370

Table 4: Error detection evaluation on CamVid (best method in bold, second best underlined).

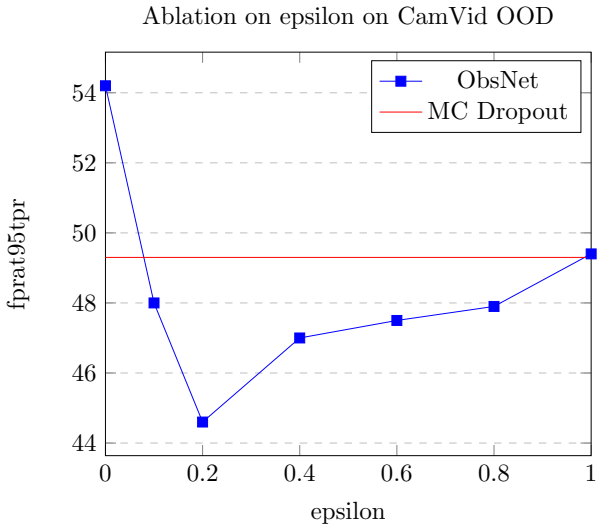

Figure 1: Evolution of the Fpr at 95 Tpr for different values of epsilon on CamVid OOD.

Since our goal is to hallucinate OOD objects, we believe the location and the shape of the attacked region are the important part.

As shown on Figure 1, we can see that the best ϵ for the attack is 0.02 with a random shape blit at a random position in the image. We can also see that even with a large ϵ, ObsNet still achieves reasonable performance.

3. Error detector

The observer is trained to assess whether the prediction differs from the true class (which is always the case for OOD regions), so it also tends to assign low confidence scores for in-domain regions with likely high errors, as shown in Figure 2. This behavior is not caused by ObsNet, but depends on the accuracy of the main network at test time and should lessen with more accurate networks. This effect shows that our method can be used for error detection, and outperforms all other methods, as illustrated in Table 4.

4. Additional Experiments: DeepLab v3+

We show on Table 5, the results on BDD Anomaly with a more recent Deeplab v3+[2] with ResNet-101 encoder. Our methods performs the best, while methods like ConfidNet do not scale when the segmentation accuracy increases as they have fewer errors to learn from.

Method	fpr95tpr ↓	AuPR ↑	AuRoc ↑	ACE ↓
Softmax [25]	60.3	95.8	81.4	0.228
Void [6]	68.8	90.2	74.0	0.485
MCDA [1]	68.1	95.1	78.8	0.265
ConfidNet [10]	64.5	95.4	80.9	0.254
Gauss Pert. [15,41]	61.4	<u>96.1</u>	<u>82.4</u>	<u>0.186</u>
MC Dropout [17]	<u>60.0</u>	96.0	82.0	0.219
ObsNet + *LAA*	**58.8**	**96.3**	**83.0**	**0.185**

Table 5: Evaluation on Bdd Anomaly (best method in bold, second best underlined), with a DeepLab v3+.

5. CamVid OOD dataset

For our experiments, we use urban street segmentation datasets with anomalies withheld during training. Unfortunately, there are few datasets with anomalies in the test set. For this reason we propose the CamVid OOD that will be made public after the review. To design CamVid OOD, we blit random animals in test images of CamVid. We add one different such anomaly in each of the 233 test images. The rest of the 367 training images remain unchanged. The anomalous animals are *bear*, *cow*, *lion*, *panda*, *deer*, *coyote*, *zebra*, *skunk*, *gorilla*, *giraffe*, *elephant*, *goat*, *leopard*, *horse*, *cougar*, *tiger*, *sheep*, *penguin*, and *kangaroo*. Then, we add them to a 13th class which is *animals/anomalies* as the corresponding ground truth of the test set.

This setup is similar to the Fishyscape dataset [6], without the constraint of sending a Tensorflow model online for evaluation. Thus, our dataset is easier to work with. We

[2]LC Chen et al., *Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation*, ECCV 2018

Method	fpr95tpr ↓	AuPR ↑	AuRoc ↑	ACE ↓
Softmax [25]	67.5	94.7	82.5	0.529
ConfidNet [10]	58.4	96.4	86.8	0.462
Gauss Pert. [15,41]	61.8	95.8	85.7	0.473
Deep Ensemble [30]	63.9	96.5	86.4	0.468
MC Dropout [17]	52.8	97.2	88.5	0.483
ObsNet + *LAA*	**42.1**	**97.7**	**91.4**	**0.423**

Table 6: Error detection evaluation on CamVid with random square attacks (best method in bold).

present some examples of the anomalies in Figure 3 with the ground truth highlighted in cyan.

6. Adversarial Attacks Detector

In safety-critical applications like autonomous driving, we know that the perception system has to be robust to adversarial attacks. Nevertheless, training a robust network is costly and robustness comes with a certain trade-off to make between accuracy and run time. Moreover, the task to *only* detect the adversarial attack could be sufficient as we can rely on other sensors (LiDAR, Radar, etc.). Although, in this work we do not focus on Adversarial Robustness, empirically we note that ObsNet can detect an attack. To some extent this is expected as we explicitly train the observer to detect adversarial attacks, thanks to the LAA.

Indeed, our observer can detect the area where the attack is performed, whereas the MC Dropout is overconfident. Furthermore, in Table 6, we evaluate the adversarial attack detection of several methods. We apply a FGSM attack in a local square patch on each testing image. Once again, we can see that our observer is the best method to capture the perturbed area.

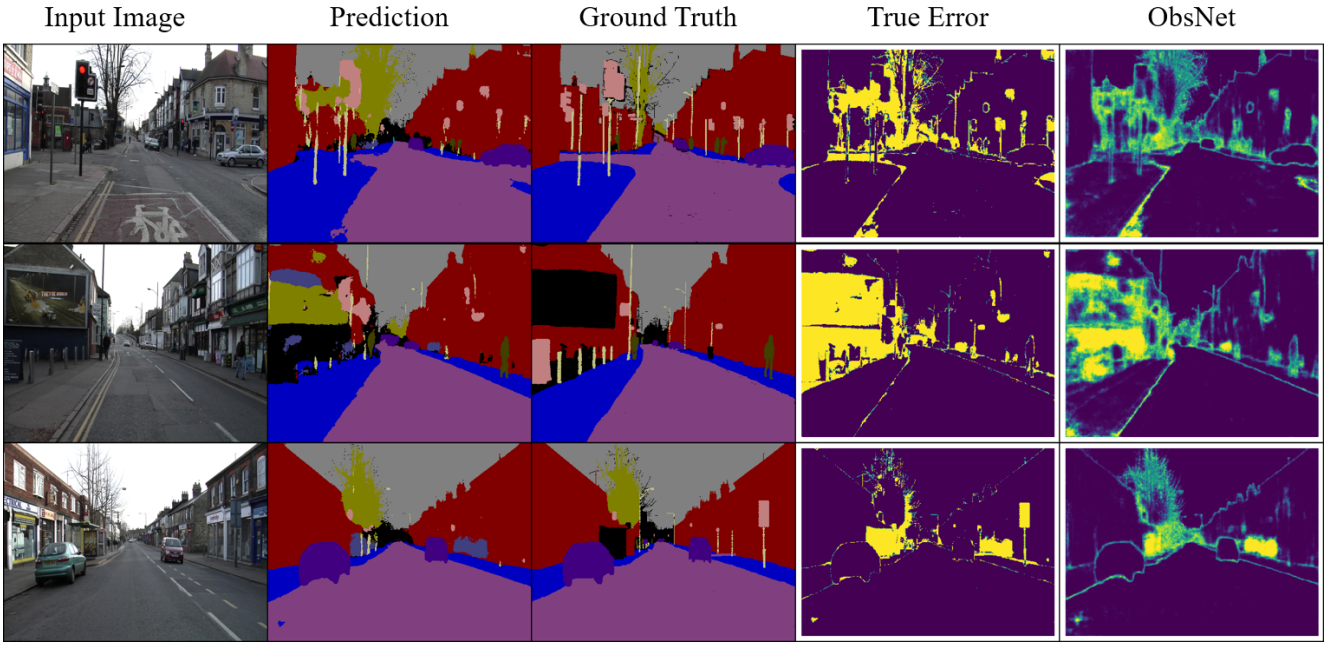

Figure 2: Evaluation of the error detection on the test set of CamVid. ObsNet prediction is close to real errors even without OOD objects.

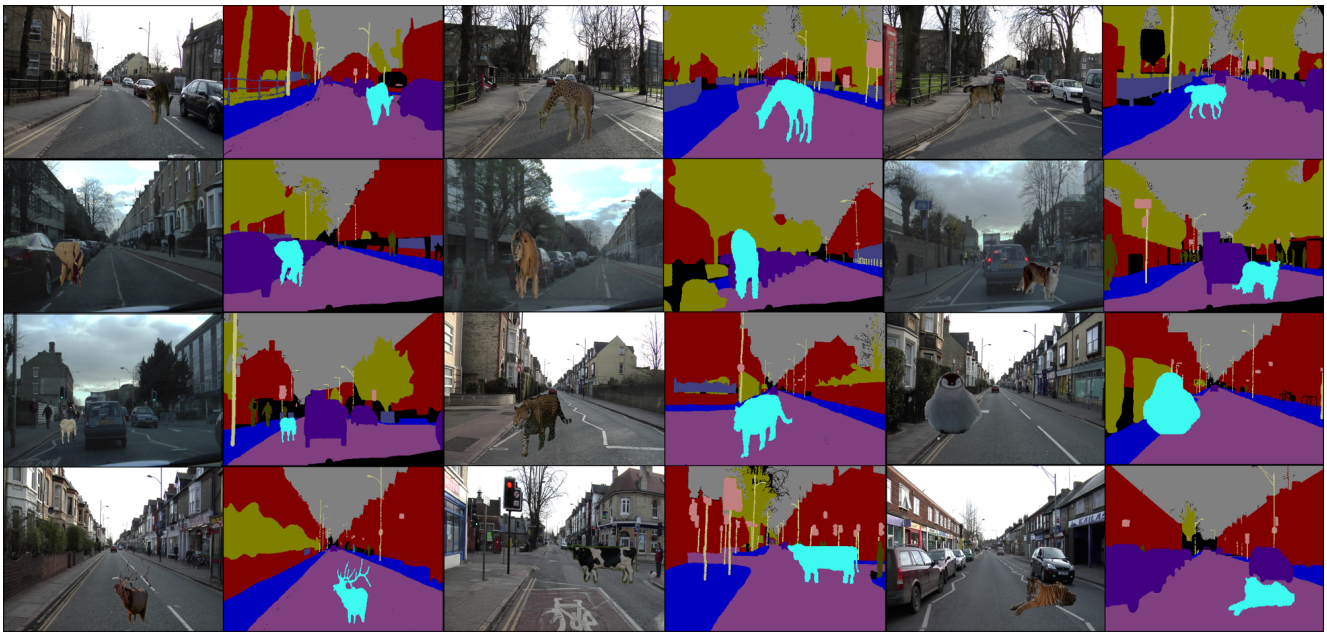

Figure 3: Examples of our dataset with anomalies and the ground truth.