OCTET: Object-aware Counterfactual Explanations — Supplementary material —

Mehdi Zemni[1], Mickaël Chen[1], Éloi Zablocki[1], Hédi Ben-Younes[1], Patrick Pérez[1], Matthieu Cord[1,2]
[1] Valeo.ai, Paris, France [2] Sorbonne Université, Paris, France

Contents

1. Technical Details

1.1. Decision models.

The decision model used in the main paper is the same DenseNet [3] as the one used in STEEX [4]. However, in addition to the 'Move Forward' class, we also study 'Stop', 'Can turn Left', 'Can turn Right' classes which were not discussed in STEEX.

1.2. BlobGan backbone.

The original BlobGAN [1] was trained on datasets of indoor scenes. We present here the changes we made to train it on BDD. Firstly, in order to generate rectangle images, we change the size of the input feature grid from 16×16 to 8×16, but increase the number of convolutional and upsampling blocks by 1, resulting in outputs images of resolution 256×512. Secondly, the number of objects in the driving scenes is usually larger compared to the indoor scenes. Therefore, we increase the number of blobs from $K = 10$ to $K = 40$. However, multiplying the number of blobs by 4 increases the number of layout network parameters. To keep the complexity reasonable, we decrease the size of the feature vectors describing the blobs from $d_{in} = 768, d_{style} = 512$ to $d_{in} = d_{style} = 256$. This model has 69.9M parameters in the generator (28.6M in the layout network and 41.2M in the synthesis network). We trained the generator for 17 days (1.5M iteration) on one NVIDIA A100 GPU with a batch size of 10.

1.3. Training the encoder

The encoder E is trained to predict the blob parameters from input images. Its architecture is similar to that of the discriminator of StyleGAN-2, except for the output size of the last layer. One challenge of predicting the blob parameters is that we have a large number of blobs, leading to a large number of trainable parameters in the encoder. To avoid this issue, we opt to have the encoder not output directly the blob parameters but instead the features of the penultimate layer of the layout network. The output layer of the encoder is then a vector, which we denote h, of size 1024; h can easily be mapped to the blob parameters by forwarding them through the last layer of the layout network.

The encoder is first pre-trained using generated images with the following reconstruction objective:

$$L_{\text{encoder}}^{\text{pretrain}} = L_2(h, E(G(h))), \tag{1}$$

where h is obtained by sampling a noise vector and forwarding it through the first layers of the layout network. For the sake of clarity, we omitted the layout network final layer that has to be applied to h before being used as input to G. Pre-training is done for 150k iterations using ADAM optimizer with a learning rate of 0.005 and batch size of 8.

Then, in the second stage, the encoder is fine-tuned on real and generated images with the following objective:

$$\begin{aligned} L_{\text{encoder}}^{\text{finetune}} = & L_2(x, G(E(x))) \\ & + \lambda_{\text{LPIPS}} L_{\text{LPIPS}}(x, G(E(x))) \\ & + \lambda_{\text{latent}} L_2(h, E(G(h)) \\ & + \lambda_{\text{decision}} L_2(f_M(x), f_M(G(E(x)))), \end{aligned} \tag{2}$$

	FID (↓)	LPIPS (↓)	Decision preserv. (↑)
Eq. 3	53.3	42.3	91.3%
w/o L_{LPIPS} (image)	50.2	56.2	92.8%
w/o L_2 (image)	57.0	42.7	91.4%
w/o L_2 (decision feat.)	54.4	41.5	73.0%
w/o L_2 (latent)	52.0	41.8	91.3%

Table 1. **Ablation of the image inversion optimization process** (Eq. 3). The 'Decision preserv.' is the percentage of images that yield the same decisions as their reconstruction using M.

where h is obtained in the same way as during pre-training of the encoder, and x by sampling from the dataset. This objective focuses on three different aspects of image inversion. First, we have to make sure that images that we encode as latent parameters with E can be reconstructed by re-applying the generator using these latents. This cycle consistency is enforced by a perceptual L_{LPIPS} loss [6] and the L_2 loss between real and reconstructed images. Second, we also ensure that generated images $G(z)$ can be encoded back into latent space using an L_2 loss between the generative latent parameters z and their estimation by the encoder $E(G(z))$. This helps keeping the latents predicted by the encoder in the generator domain; this term is only used for generated images. Finally, to encourage the reconstructed image to be faithful to the decision model, we use an L_2 distance between the features $f_M(x)$ of the input and the reconstructed image $f_M(G(E(x))$. This term makes sure to preserve the features that are important to the decision model M. Those features must contain both the decision taken by the decision model, but also capture the perceptual semantics that led to the decision. In particular, for the DenseNet decision model that we have, we consider activations at the last convolutional layer of the decision model (DenseNet). This is to ensure that the decisions are kept unchanged on the reconstruction but also that features that led to those decisions remain the same. This finetuning stage is conducted with ADAM, for 150k steps, with a learning rate of 0.002. Hyper-parameters are found after coarse manual inspection on some qualitative samples, $\lambda_{\text{LPIPS}} = 1$, $\lambda_{\text{latent}} = 0.1$, $\lambda_{\text{decision}} = 0.05$. The choice of these hyper-parameters is not critical as obtained latent codes z are then refined in an optimization phase, as explained in the main paper (Sec. 3.4).

1.4. Time complexity

OCTET takes ∼28s per counterfactual image, in a batch of 16. The inversion step amounts to 85% of that time and the CF optimization 15%.

	LPIPS (↓)	L_2 (↓)	L_1 (↓)	Decision preserv. (↑)
Pretrained enc. (after Eq. 1)	57.2	0.230	0.324	63.1%
Finetuned enc. (after Eq. 2)	54.6	0.175	0.278	71.5%
w/o L_2 on f_M	54.8	0.176	0.279	61.4%
L_1 instead of L_2	55.3	0.186	0.285	66.3%

Table 2. **Ablation of the finetuning process of the encoder** (Eq. 2). The 'Decision preserv.' is the percentage of images that yield the same decisions as their reconstruction using M.

2. Further ablations

We present in this section ablation studies for the image inversion optimization process. In particular, we ablate the different terms of the loss. Moreover, we recall that the initial values in this optimization process are given by the output of an encoder, and we ablate its training objectives as well. In addition to the visual reconstruction metrics (FID, LPIPS and pixel-wise distances), we also evaluate the semantic fidelity of the reconstructions with a 'Decision preservation' score. More precisely, the score measures the proportion of reconstructions that yield the same decision as the original image when presented to the model M. This property is important as decision switches that occur during the reconstruction phase are guided neither by the target class nor by the studied model M and therefore can hinder the downstream task of creating a reliable counterfactual explanation.

2.1. Image inversion

We present in Tab. 1 an ablation study of the terms of Eq. 3 of the main paper that we recall below:

$$\begin{aligned}\phi^q, \psi^q = \arg\min_{\phi,\psi} \; & L_{\text{LPIPS}}(G(\phi,\psi), x^q) + L_2(G(\phi,\psi), x^q) \\ & + L_2(f_M(x^q), f_M(G(\phi,\psi))) \\ & + L_2((\phi,\psi), E(x^q)). \end{aligned} \tag{3}$$

We observe that the first two terms improve FID and LPIPS while the last two do not seem to influence those scores. However, without the third term, we note a huge drop (91.3% vs. 73.0%) in the number of reconstructed images that conserve the model's original decision. The last term is a standard safeguard in the literature [7]: although there is no improvement in terms of LPIPS and FID, we observed that, without it, some objects were lost in the reconstruction, especially grey cars that blend in the road.

2.2. Encoder training

Tab. 2 reports results for the ablation of the encoder training losses (Eq. 1 and Eq. 2). The loss on features f_M helps preserve the original decision of M while keeping reconstruction quality. Using it, 71.5% of auto-encoded images yield the same decision as the input vs. 61.4% without.

	FID (↓)	LPIPS (↓)
STEEX	60.2	0.435
OCTET	**53.3**	**0.423**

Table 3. **Reconstruction capacities of OCTET and STEEX.** We check the quality of reconstruction with FID (is the reconstructed image realistic?) and LPIPS (is the reconstructed image close to the input image?). The values below were computed on the validation set of BDD segmentation dataset (1000 images). Note that STEEX uses ground truth segmentation masks as additional input.

Also, we chose the L_2 pixel loss as in the BlobGAN paper. With L_1, results are slightly degraded.

3. Reconstruction quality

We compare the ability of OCTET and STEEX to reconstruct real images. To measure the reconstruction quality, we use the FID [2] between all reconstructions and the set of real query images. We also measure the mean L_{LPIPS} [6] distances between all pairs of real and reconstructed images.

In Tab. 3, we report FID and LPIPS scores for reconstructed images. Overall, we observe that the backbone used in OCTET and our inversion strategy (encoder + optimization) leads to images that are closer to the input image (lower LPIPs) but also more realistic (lower FID) which means that they are closer to the original data distribution. Moreover, we stress that OCTET does not use any ground-truth segmentation maps, unlike STEEX.

In Fig. 1, for some query images (1st column), we show some qualitative results of:

- images obtained with the encoder E which are then used as starting points of the optimization described in Eq. 3 of the main paper (2nd column);
- OCTET reconstructions, which are images obtained after further optimization with Eq. 3 of the main paper (3rd column);
- STEEX reconstructions (4th column).

Overall, while the encoding step is able to grasp the rough structure and colors of the scene, we can observe that the optimization steps that follow are key to get the fine and precise details of the scene. Moreover, we can see that STEEX reconstructions sometimes miss important details (lines and brake lights) that are well captured in the latent space we use for OCTET. Finally, on some small background details, STEEX reconstructions are sometimes more accurate and faithful to the query image (e.g., the shape of trees). However, STEEX makes use of semantic maps that indicate the semantic class of each pixel. These maps are strong guides but are very costly to produce for end-users.

3.1. Discussion on sparsity of changes

Sometimes, counterfactual explanations do not display sparse changes with respect to the original query image. In fact, the main source of non-sparsity comes from the challenges of GAN inversion. Despite our efforts and contributions allowing us to outperform previous works in this regard, our reconstruction already introduces changes (see Fig. 1). The counterfactual optimization step itself does not degrade LPIPS further (see Fig. 4 of the main paper and Tab. 3). In Fig. 2 (not cherry-picked), we illustrate the sparsity of the CF optimization itself by starting from generated queries to build the CF, thus not necessitating a reconstruction step. As the sparsity in this step is satisfactory, we decided not to add a pixel-level regularization that would also heavily penalize explanations involving object displacement.

4. Finding the blob index to target

We discussed in the main paper how to assess the importance of specific blobs with OCTET. But for practical use, we need to identify which blob corresponds to a given object in the scene. We here explain how, taking advantage of BlobGAN [1], the generative backbone we use, we are able to find the blob identity.

BlobGAN learns to represent scenes as a collection of blobs distributed on a canvas in an unsupervised fashion. As discussed in the original paper for *indoor scenes*, even without supervision from object positions and classes, the model learns to associate certain blobs with certain objects. We observed that similar properties emerge when training the model on *outdoor driving scenes*. For instance, in Fig. 3, we visualize the correlation between blobs and the semantic classes for cars and roads. We use a pre-trained semantic segmentation network to measure the number of pixels that disappeared from each class when removing a blob by setting its size to a negative number. Using such figures, we are able to determine for instance that blobs 14, 18, 23, 29, 30, and 35 are very likely to correspond to car-blobs. We then visualize in Fig. 4 the distribution of the spatial positions of the center of the blobs on the canvas. To be clear, for each blob index, we plotted the location of its center and accumulated the plots over 10k generated images. This figure shows that the blobs have a localized position which is consistent with the fact that the blobs have a semantic meaning. Combining the location and the semantic class of the blob, we can infer that blob 30 is likely to correspond to a car in the middle of the image, while blob 35 is a car on the right for instance.

Using this knowledge, we can then directly intervene on the spatial parameters of the blobs to see how it affects the image and confirm our hypothesis. We show the results of such manipulation in Fig. 5 and Fig. 6. These findings are

Figure 1. **Examples of inversion results for OCTET (ours) and STEEX [4].**

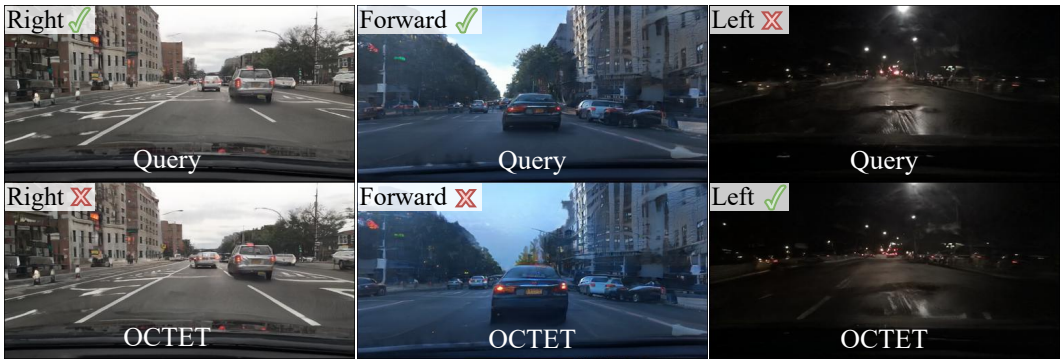

Figure 2. **Counterfactual explanations on generated query images.** Neither the images nor the target classes are cherry-picked.

consistent across images and make it fairly straightforward to identify the correct blob for object-targeted counterfactuals presented in the main paper.

5. Details on the User Study

5.1. Protocol details

Here, we describe more precisely the details of the user study (Sec. 4.5). The experiment was conducted as an online form, with no interaction with any operator. The respondents are all voluntary, and we targeted participants that have familiarity with deep learning. Participants are randomly split across the two groups (Control group and group with explanations). We show in Fig. 7 a description of the study and the templates we used for the forms.

To build the online form, we needed 28 distinct query images: 16 were used for the observation phase and 12 for the questionnaire. Those were randomly sampled, with

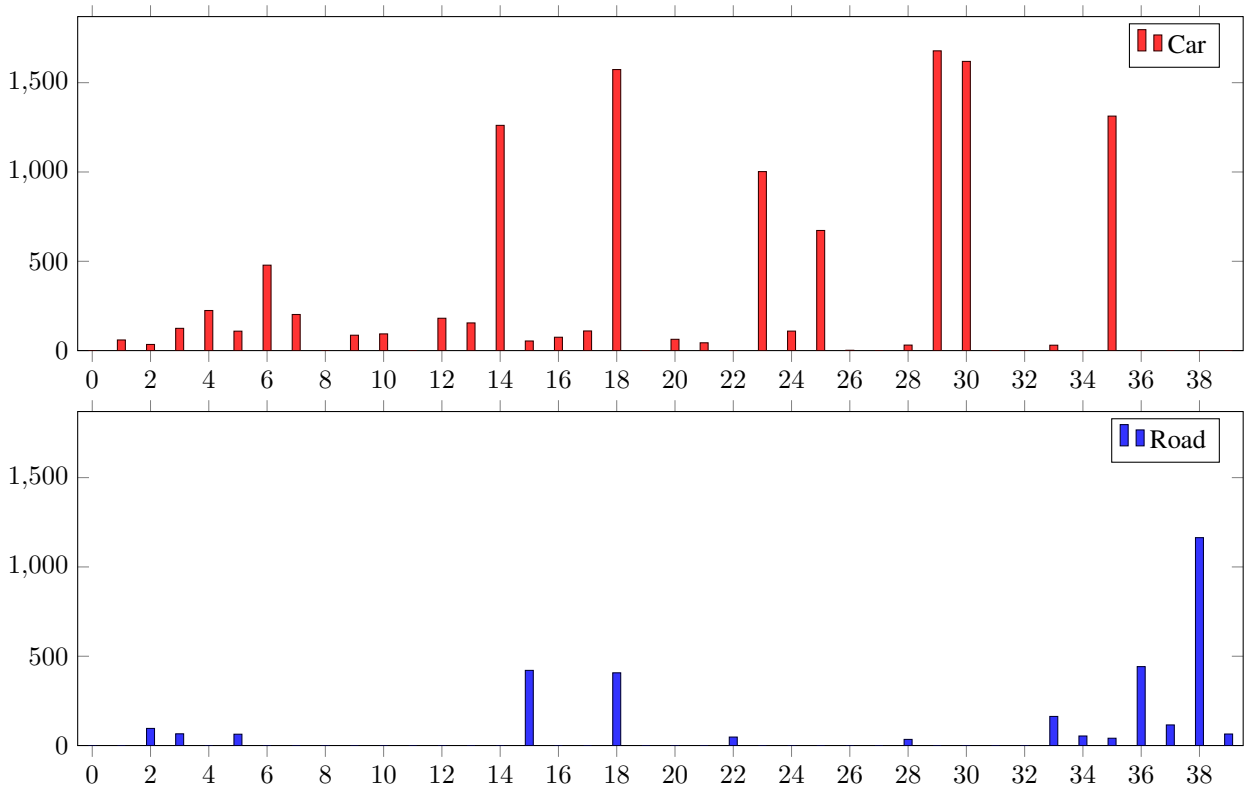

Figure 3. **Blobs semantic meaning.** We visualize the correlation between the blobs and the semantic classes 'car' (*top*) and 'road' (*bottom*). The x-axis displays blob ids, while values in the y-axis represent the mean number of pixels that are no longer from class c when removing blob k computed by sampling 200 different latent codes. The number of pixels for class c is estimated using a pre-trained semantic segmentation network.

no overlap between the two sets. Unknown to the participants, we made sure that inside each set there are an equal number of true positives, true negatives, false positives, and false negatives. This prevents the user's from over-relying on their prior knowledge of the task, and should also facilitate the detection of interesting behaviors of the model. For the observation images, we have to compute the predictions from the decision model and the counterfactual explanations. The form for the test group contains all three elements while the form for the control group only contains the images and the decision, but no explanation. For the questionnaire, we simply provide the raw images to both groups, without any additional information. Using those same image sets for the two groups ensures that the comparison between them is more reliable. However, it does not control for the variance induced by image selection. To obtain more robust results, we build 5 versions for each form by randomly sampling 5 times the 28 query images. This leads to a total of 10 variations of the forms (5 for the control group and corresponding 5 for the test group). We randomly sample the assignment of one version of the form to each participant.

For the bias detection study, we first presented them with one additional test image, the same to all participants, for which they had to predict the output of the decision model. We were not interested in their prediction this time, but we then asked them: "On the last image, please explain the factors that drove your choice in a few words". We also asked them two additional questions:

- "Did you manage to identify any particular behavior of the decision model?"
- "Did you manage to identify any problem or unexpected behavior in the decision model?"

The goal of those questions was to lead them to describe any peculiar behavior they would have inferred from the observation phase.

5.2. Collected responses

The collected answers to the free-form questions are presented in Tab. 4. Unknown to the participants, the decision model looked at the presence of cars on both the left and the right side of the road to make its prediction on the 'Turn Right' label. No participant in the control group mentioned

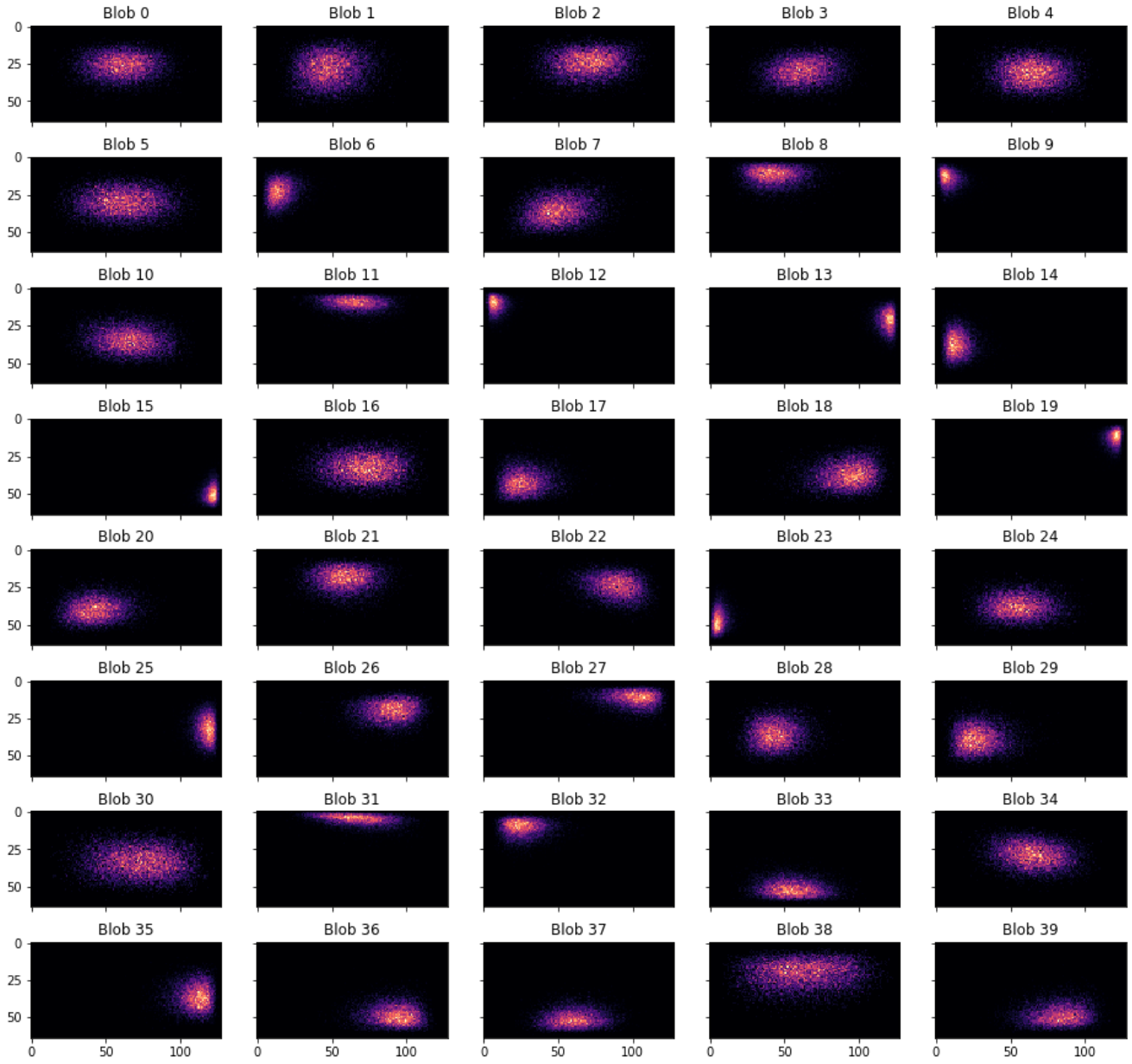

Figure 4. **Blobs spatial distribution.** We visualize the spatial distribution of blob centroids. By combining the information about the semantic meaning of the blobs and their spatial localization, we can precisely label main blobs (e.g., blobs representing the front car vs. blobs representing cars on the right).

this issue while 65% of those that had access to the OCTET explanations did.

6. Preliminary experiments on LSUN dataset

In Fig. 8, we present preliminary results using the official pre-trained BlobGAN [1] generator on 3 classes of LSUN. The decision model is a 3-class classifier trained on LSUN to distinguish between kitchens, living rooms and dining rooms. We use generated images as queries. We can observe for instance that while the presence of a sofa is a clear distinguishing feature for the decision model, chair style contributes as well (last column). Also, while the explanations include layout changes impossible with STEEX [4], the position of objects does not seem as important as in the BDD experiments as objects are not displaced as much.

Figure 5. **Resizing cars.** We change the size of certain blobs representing cars. In addition to being able to change their size, we can make them appear and disappear. We stress that we are doing *manual edition* by intervening on the size parameter of the blobs. This contrasts with other figures of the main paper where we are doing *counterfactual explanations* and changes are automatically found with the optimization process to explain the decision of a model.

References

[1] Dave Epstein, Taesung Park, Richard Zhang, Eli Shechtman, and Alexei A. Efros. Blobgan: Spatially disentangled scene representations. In *ECCV*, 2022. 1, 3, 6

[2] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *NeurIPS*, 2017. 3

[3] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *CVPR*, 2017. 1

[4] Paul Jacob, Éloi Zablocki, Hedi Ben-Younes, Mickaël Chen, Patrick Pérez, and Matthieu Cord. STEEX: steering counterfactual explanations with semantics. In *ECCV*, 2022. 1, 4, 6

[5] Fisher Yu, Yinda Zhang, Shuran Song, Ari Seff, and Jianxiong Xiao. LSUN: construction of a large-scale image dataset using deep learning with humans in the loop. *CoRR*, abs/1506.03365, 2015. 9

[6] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018. 2, 3

[7] Jiapeng Zhu, Yujun Shen, Deli Zhao, and Bolei Zhou. In-domain GAN inversion for real image editing. In *ECCV*, 2020. 2

Figure 6. **Moving cars.** We change the position of certain blobs representing cars as pointed by the arrow by changing their centroid coordinates. We stress that we are doing *manual edition* by intervening on the position parameters of the blobs. This contrasts with other figures of the main paper where we are doing *counterfactual explanations* and changes are automatically found with the optimization process to explain the decision of a model.

Figure 7. **User study overview.** The study is conducted as an online form in two phases. First, in the observation phase, the participant is shown examples to analyze the prediction of the model. Then, in the questionnaire, they are asked to guess the prediction of the model. The control group (*bottom part*) is only shown images and associated decision in the observation phase. The OCTET group (*top part*) has, in addition, access to counterfactual explanations.

Figure 8. **Qualitative results on LSUN dataset [5].**

Group	Question 1: On the last image, please explain the factors that drove your choice in a few words	Question 2: Did you manage to identify any particular behavior of the decision model?	Question 3: Did you manage to identify any problem or unexpected behavior in the decision model?
Control	white lanes visible right. A Car is also at the right of the screen but not a dense lane of cars	yellow lanes means can't turn right. If there is a dense lane of cars can't turn but if there is a few can turn. If the ground is covered with some white marks can turn.	yes the model does not understand if the right lane is occupied or going in the other direction. When the lane is free but there is a dense lane of cars parked the algorithm predicts that we can't turn right
	Car detected on the right	The decision is "Can't turn right" when there are yellow ligns, or there is a car or an obstacle on the right.	The model does not turn right even when the road seems empty, or does turn right when the right path is very narrow
	Presence of the car	The model focus mostly on ground lines and the presence of a car	Sometimes the model get confused when ground lines are not typical
	12th example looked the same, the front car being too close to the ego-car.	No	Similar paterns (e.g. yellow lines) didn't seem to be taken into consideration by the model
	Black car on the right	Don't turn right if there is a car ahead on the right lane.	Not sure, but might not turn right if the car ahead (in the same line) is close.)
		only a car in the same lane but the right lane is free −¿ "can't turn". Cars on both lane −¿ "can turn".	problem with yellow lanes
	white line on the right	white line on the right	full white line on the right
	yellow line, visibility of other cars, lighting/contrast		lighting affected the model a lot
	The is a car on the right	If it sees a car on the right far away, doesn't turn right. If it's very close, doesn't. If it's a few meters ahead, checks that is not parked ?(brake lights on?) Or that the light is green? Not that much sensitive to double yellow lines	Bit very sensitive to double yellow lines
	Right lane is clearly occupied by a car, the classifier can handle such situation	Hardly, maybe tend to say 'can't turn right' while right lane is open because he sees a car on a third lane (to the right of the lane to the right)	see previous answer
	It seems that if the front car is too close, the prediction will always be can't turn	If there was a double Yellow line on the right, the model would always return false	
	It is a situation never seen before (car stop behind another, next to another), in doubt, you can't turn right.	Not really, it was mostly "right", but I didn't understand the pattern of the wrong predictions.	Yes, some predictions were wrong sometimes (e.g., crossing a full line).
	due to the car on the lane next to our car	yes, I think the model is very well in detecting the two yellow parallel lines on the street indicating a prohibition of reel swapping	Yes, problem turn right is NOT turn 90 degree angle right but just turn right to the next lane as in reel swapping. I think this decision should be differentiated cause I find "turn right" confusing as it can mean both = also a problem for the model
		no	no
	Lines on road (solid vs dashed) + new objects on the right (Can't turn) + there is no car or a bit far (Can turn)	Not clear, It seems that if there is solid line on the right, a close car or new objects it can't turn	yes, many false negative
	the road, the cars, the lines on the road (continuous/dashed..)		sometimes turning right is permitted by the model while there is a continuous line on the road
	obstructing car on the right	free road, available turn further down the road = right turn ok; car present on the right lane = right turn not ok	the model does not understand the direction of traffic
	The image is unlike the others. And in case of doubt, I assume that the model is conservative	Double lines make it not turn right	Does not pay too much attention to other cars
	car too close	no	no
	the car is too close from the other car	i think : car in front of the car and yellow line	
OCTET	No car on the **left**	If a car is on the **left** the model predict can't turn right	It seems that the model is disturbed by the car on the **left**
		looking for cars on the **left**, or less often in front of our car	Model should look also look to the right. Often, a car on the **left** is not an issue to turn right
	No cars stationed on the **left**, no continuous white band	The model might be disproportionately affected by lane width and parked cars	The model sometimes focus on irrelevant factors
	Car in front and on the right too close to me	To decide if the car can turn right, the model is looking if there are closed cars on the right side, in front AND on the **left** side	Yes, to decide if the model can turn RIGHT, the model is also looking whether there is a car near on the **LEFT** side
	Multiple cars, multiple brakes lights, car too close	Having a car on the **left** or multiple brakes is more likely to be "can't turn" prediction	Sometimes it says you can turn right although there does not seem to be a road on the right
	Cars are too close and this seems to be a no no for the decision model	If there are cars too close it seems not to allow to turn right	
	The car in front of me in close and the lights are on which seems to be understood by the model as "can't turn right"	When there is a car in the bottom **left** part of the image, it's tagged "can't turn right"	The results are very counter-intuitive because one would expect that 'can't turn right" should be the answer if there is a car or an obstacle on the right (not on the **left**)
	because the right lane is occupied	The presence of a double yellow line on the right prohibits turning right	I did not understand many examples where there was no double line and where the right lane seemed clear and where, however, turning right was prohibited.
	Can't turn right when front vehicle is close and when there is a car on the **left** or on the right	Influcene the decision : Yellow line, distance with the front car, position of vehicles on the right or on the **left**.	The model does not consider the lateral space on the right side.
	The car in front is too close, and there is a car on the right	To be able to overtake, we must not have cars overtaking us, nor cars that are too close, if possible good general visibility, in particular lines.	No
	There is a car too the right which is close.	It does not want to turn when there is a car to the **left**	Yes, as said before
	There is no car on the **left**, no double yellow can be seen	if the model detects an object car-shaped on the **left**, a double yellow line on the ground, or a long and large object to the ground on the right it doesn't turn right	it's decisions do not seem correct regarding driving ability
	Car ahead too close	- double yellow lines: no turn - cars on the **left**: no turn - car ahead too close: no turn - need a car ahead to evaluate if there is the space to turn right	cars on the **left** line: no tun - need a car ahead to evaluate if the road is large enought to turn
	car visible on the right	Yes when cars are closed either **left** or right the model predicts "can't turn right"	Yes, the understanding of the geographic position of other cars
	car on the right, too close to my car	car on **left**/right close to my car. continuous white line on the right. Yellow line often on the right and sometimes on the **left**	can't turn on right when cars or lines on the **left**
	In some images the model did not detect the car at the right hand part. SO here it seems "too far" to be detected by the model.	Seems to respect more the lines than the vehicles or pedestrians	Should prevent fro turn right when there is a car a the right...
	Car on the right too close to the driver's car	Look for the presence of white line + presence of a car within a radius	Sometimes a car is too close on the right side and the model still consider it can turn right. Sometimes the prediction is just incorrect (yellow line and still predicts it can turn right)
	There is a car next on the right that is too close to the initial car wanting to make a turn, so the model should detect that it is close enough and it's not safe to do a right turn .	Something i noticed is that when the line is continuous or not,	Yes the fact that is doesn't distinguish the nature of the line drawn in the ground
	1) there's a very close to the right. 2) the car just in front is very close.	failure to distinguish between right and **left**. When there's a closer car to the **left**, the decision is "can't turn right" even though the car can indeed turn right. (Naive question : are you using random flipping as data augmentation technique without changing the label during your training?)	failure to distinguish between right and **left**. When there's a closer car to the **left**, the decision is "can't turn right" even though the car can indeed turn right. (Naive question : are you using random flipping as data augmentation technique without changing the label during your training?)

Table 4. **Responses to the free-form questions in the user-study.** The participant had the option to leave the fields blank; they are still shown in the table. We highlight in bold every occurrence of the word 'left'.