# Supplementary Material

## ACTIVE: Towards Highly Transferable 3D Physical Camouflage for Universal and Robust Vehicle Evasion

## A. Overview

This supplementary material describes the details of our implementation and evaluation results that can not be included in the main paper due to page limit. Furthermore, additional experiments and more evaluation samples for all experiments on the main paper are also presented.

## B. Algorithm Details

### B.1. Triplanar Mapping

In this section, we provide a detailed algorithm and figures for triplanar mapping [14]. Algorithm 1 and Figure 1 describe the step-by-step process of extracting a projected texture using triplanar mapping.

**Algorithm 1** Triplanar Mapping Algorithm

**Input:** Depth Image $x_d$, Camera Parameters $\phi$, Projection Augmentation $\phi_{rd}$, Adversarial Texture $\eta$
**Output:** Projected Texture $\eta_p$
(1) Extract Surface Normal $x_{SN}$ and Surface World Coordinates $x_{SWC}$
Extract camera intrinsic matrix $\phi_{in}$ and extrinsic matrix $\phi_{ex}$ from $\phi$
Compute 3D Local Coordinates $x_{LC}$ from $\phi_{in}$ and $x_d$
$x_{SWC} \leftarrow x_{LC} \times \phi_{ex}$
Compute $x_{SN}$ from $x_{SWC}$ using the cross product of two tangent vectors
(2) Extract Triplanar Mask
Compute Triplanar Mask $x_{TM}$ based on the highest absolute value of $x_{SN}$ for each axis
Compute $x_{TM\_x}$, $x_{TM\_y}$, $x_{TM\_z}$ masked for each axis
(3) Extract Projected Texture $\eta_p$
Compute the Repeated Surface Coordinates $x_{RSC}$ by modulating the $x_{SWC}$ with the repeated texture size based on $\phi_{rd}$
Take the UV indices of $x_{UV\_x}$, $x_{UV\_y}$, $x_{UV\_z}$ from each axis of Repeated Surface Coordinates $x_{RSC}$
Assign the projected texture $\eta_{p\_x}$, $\eta_{p\_y}$, $\eta_{p\_z}$ for each axis from $\eta$ mapped with $x_{UV\_x}$, $x_{UV\_y}$, $x_{UV\_z}$
$\eta_{p\_x\_m} \leftarrow \eta_{p\_x} \times x_{TM\_x}, \eta_{p\_y\_m} \leftarrow \eta_{p\_y} \times x_{TM\_y}, \eta_{p\_z\_m} \leftarrow \eta_{p\_z} \times x_{TM\_z}$
$\eta_p \leftarrow \eta_{p\_x\_m} + \eta_{p\_y\_m} + \eta_{p\_z\_m}$

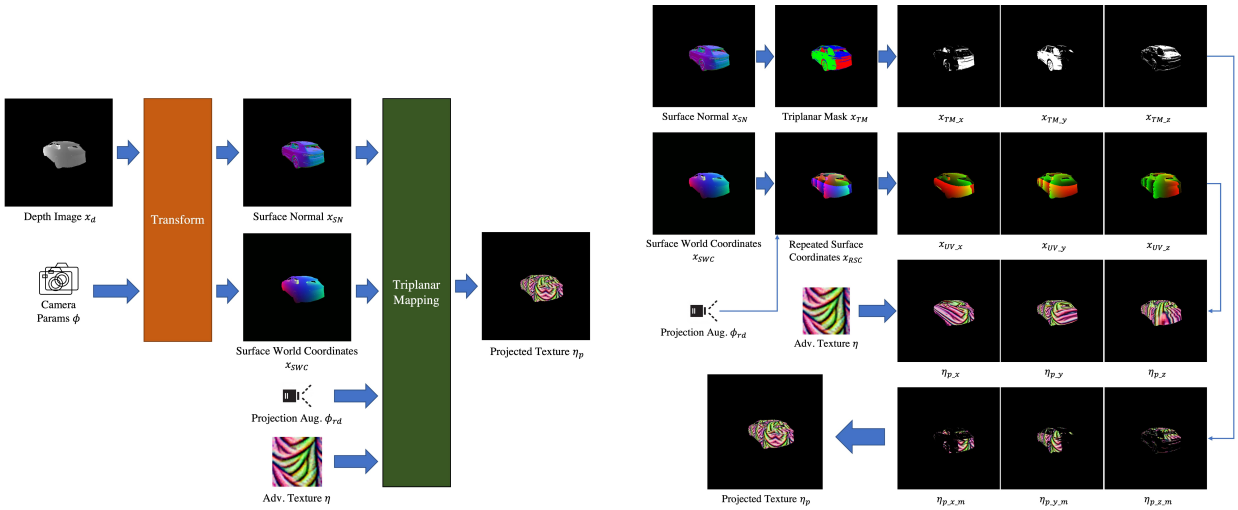
### B.2. Framework Procedure

This section describes the optimization of our adversarial texture using the ACTIVE framework. Drawing inspiration from the DTA framework [19], we employ the Neural Texture Renderer (NTR) for rendering our triplanar-mapped textures onto target vehicles while maintaining various physical properties. Prior to optimizing the adversarial texture, it is necessary to train the NTR in advance. A detailed explanation of the NTR can be found in Section D. Subsequently, during the ACTIVE training phase, we generate the optimal adversarial texture by minimizing $L_{total}$. To further evaluate the attack performance, we conduct a physical simulation (on UE4) wherein our adversarial texture is applied as a world-aligned texture to the target vehicle in UE4, as illustrated in the ACTIVE test phase in Figure 2 of our main paper.

## C. Implementation Details

### C.1. Dataset

**Cars.** We select multiple types of cars that are available on the CARLA [4] simulator. Five types of cars used for attack texture generation and robustness evaluation are Audi Etron, Citreon C3, Mercedes-Benz Coupe, Nissan Patrol, and Charger 2020. Additionally, five other cars used for universality evaluation are BMW GrandTourer, Audi A2, Mercedes-Benz C-Class, Jeep Wrangler Rubicon, and Tesla Model 3. All cars are visualized in Fig. 2.




(a) Triplanar mapping pipeline

(b) Triplanar mapping mechanism

Figure 1: Detailed pipeline and mechanism of Triplanar mapping [14]


(a) Multiple types of cars used for attack texture generation and robustness evaluation


(b) Multiple types of cars used for universality evaluation

Figure 2: All cars used in our evaluation

**For NTR Model Training.** We select a map in CARLA and randomly choose 75 random locations for training and 50 different locations for testing. On each location, we generate a car dataset using a camera with a 5-meters distance, 15-degree pitch, and every 30-degree rotation. For each transformation, we gradually change the car types and textures with eight boundary colors and one gray color (as reference image) for training. Additionally, we select 50 random colors for testing. In summary, we employ 50,625 photo-realistic images for training (consisting 5 cars x 75 positions x 12 camera poses x 9 colors), and 150,000 images for testing (consisting 5 cars x 50 positions x 12 camera poses x 50 colors).

**For Attack Texture Generation.** We use the same cars and camera poses, then choose 250 random locations on the same map to synthesize the reference images. In summary, we use 15,000 photo-realistic images as reference $x_{ref}$ (consisting 5 cars x 250 positions x 12 camera poses x 1 color). For each dataset, we also synthesize the car mask (i.e., the target object).

**For Attack Texture Evaluation.** For robustness, we evaluate Audi Etron at 50 random locations, while for universality, we assess multiple cars at 20 random locations. We use wider camera transformation to evaluate attacks on untrained camera distributions as default. We also define multiple camera settings, including $[[5, 7.5], [7.5, 10], [10, 12.5], [12.5, 15]]$ meter distances, $[[0, 15], [15, 30], [30, 45]]$ pitch degrees, and $[[0, 30], [30, 60], ..., [330, 360]]$ rotation degrees. Each value is selected randomly within the range for each evaluation sample. A total of 14,400 images (consisting 5 cars x 20 positions x 4 distances x 3 pitches x 12 angles) for a single texture evaluation are used in our experiments.

## C.2. Compared Methods

We provide detailed parameter setup of compared methods: Dual Attention Suppression (DAS) [25], Full-coverage Camouflage Attack (FCA) [22], Universal Physical Camouflage (UPC) [9] CAMOU [29], Enlarge-and-Repeat (ER) [27], and Differentiable Transformation Attack (DTA) [19] in our experiment. Except for texture taken from original codes, we re-optimize all methods targeting YOLOv3 [15] on Audi Etron car, and use the same simulated town for a fair comparison. Additionally, we include random and naive camouflage patterns to evaluate the model's robustness against arbitrary textures.

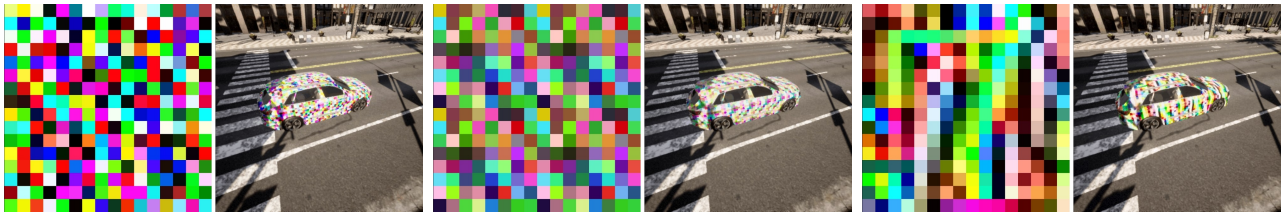
**DAS and FCA.** These methods use Neural Mesh Renderer (NMR) [10] to optimize the adversarial camouflage over a 3D car and attach it to the background, which is CARLA simulated town. We use the original texture from their official codes (DAS: https://github.com/nlsde-safety-team/DualAttentionAttack, FCA: https://github.com/idrl-lab/Full-coverage-camouflage-adversarial-attack) for our evaluations. Specifically, both textures are optimized for Audi Etron and CARLA Town10. We follow FCA for generating texture targeting YOLOv3 [15] as the white box model. Since these methods depend on a specific 3D UV map, we only evaluate them on robustness evaluation.

**UPC.** This method outputs a masked patch that can be attached to the target surface. Since UPC has different settings to reproduce in our environments and is specifically designed for targeting Faster R-CNN [16], we execute their official code: https://github.com/mesunhlf/UPC-tf and keep the original attack setup on cars, then use their attack patches in our environment. Following [19], we utilize decals in Unreal Engine to apply UPC patches onto target car surfaces, covering the hood, doors, rooftop, and back to ensure visibility from diverse perspectives. Since UPC has a different setting from ours, we only evaluate this method in transferability comparison. Fig. 3 shows samples taken from related works' original papers (left) and how we fully transfer the texture on Unreal Engine 4 [5] (right).


(a) DAS [25] (b) FCA [22] (c) UPC [22]

Figure 3: Transferring textures from the original paper (left) to our evaluation setting by fully rendering it on Unreal Engine 4 with epic settings (right).

**CAMOU, ER, and DTA.** These methods output repeated patterns that cover all target object surfaces as physical camouflage. We choose the same $16 \times 16$ texture resolution as the best original setting that forms mosaic-like patterns. We closely replicate the original papers' approach, but we rebuild the environment and target models same as our evaluation setup (optimizing texture on Audi Etron and targeting YOLOv3 [15]). For CAMOU, we generate the camouflage pattern using the same clone network architecture and other parameters as the original paper. For ER, we use the same parameters as the original paper, except changing the $p = 3$ and $r = 1$ parameters to produce the same $16 \times 16$ texture. For DTA, we use NTR as neural renderer in our paper and optimize the texture using the original parameters. All generated textures and the rendered cars are shown in Fig. 4.



(a) CAMOU [29] (b) ER [27] (c) DTA [19]

Figure 4: Re-implementation of previous works in our test environment. For each figure, the left side depicts the generated camouflage pattern, while the right side depicts the result of a car rendered by repeated texture using world-aligned texture.

In conclusion, the tables in our main paper were designed with *specific criteria* in mind. In particular, Tab. 2 compares only adversarial camouflage methods based on *Neural Renderer* (i.e., DAS, FCA, and DTA), aimed at assessing its suitability for digital-to-physical transferability. Next, Tab. 3 compares methods that can directly attack the *same target model* (YOLOv3), *vehicle*, and *simulated town*, with Random and Naive camouflage serving as additional benchmarks for robustness against arbitrary texture. Subsequently, UPC was *omitted* in Tab. 3 due to its loss being designed *only for attacking Faster R-CNN*. In contrast, Tabs. 4 and 5 place UPC for comparing universality but *exclude* DAS and FCA due to their *object-dependent textures* cannot be universally applied to different cars.

### C.3. Evaluated Models

We evaluate our method on multiple publicly-available state-of-the-art COCO [11] pre-trained object detection models including EfficientDet-D2 [20], YOLOv3 [15], SSD [12], Faster R-CNN [16], Mask R-CNN [7], YOLOv7 [21], Dynamic R-CNN [28], Sparse R-CNN [18], Deformable DETR [30], Pyramid Vision Transformer [26], MobileNetV2 [17], and YOLOX-L [6], with the default score threshold set to 0.30. Furthermore, we also evaluate the attack patterns on state-of-the-art pre-trained panoptic segmentation model including Max-DeepLab-L [23] and Axial-DeepLab [24] with SWideRNet [2] Backbone either pretrained on COCO [11] or Cityscape [3] dataset. For the object detection task, we evaluate the target car AP@0.5, while for the segmentation task, we evaluate the target car pixel accuracy. The details of all evaluated models, including model variant, publication venue, and implementation framework, are available in Table 1 for object detection models and Table 2 for segmentation models.

Table 1: All COCO pre-trained object detection models used in our evaluations. Models are sorted by publication year.

| Model Name | Model Variant | Venue | Year | ML Framework | Implementation | Link |
|---|---|---|---|---|---|---|
| Faster R-CNN | ResNet50 | NPIS | 2015 | TensorFlow | TF Object Detection API [8] | Here |
| SSD | ResNet50-FPN | ECCV | 2016 | TensorFlow | TF Object Detection API [8] | Here |
| Mask R-CNN | Inception ResNetV2 | ICCV | 2017 | TensorFlow | TF Object Detection API [8] | Here |
| YOLOv3 | YOLOv3 | ArXiv | 2018 | TensorFlow | TF Re-implementation | Here |
| MobileNetv2 | FPN Lite | CVPR | 2018 | TensorFlow | TF Object Detection API [8] | Here |
| EfficientDet | EfficientDet-D2 | CVPR | 2020 | TensorFlow | TF Object Detection API [8] | Here |
| Dynamic R-CNN | ResNet50 | ECCV | 2020 | Pytorch | MMDetection [1] | Here |
| Pyramid Vision Transformer | PVT-S | ICCV | 2021 | Pytorch | MMDetection [1] | Here |
| Deformable DETR | Two-Stage DDTR | ICLR | 2021 | Pytorch | MMDetection [1] | Here |
| Sparse R-CNN | ResNet50-FPN | CVPR | 2021 | Pytorch | MMDetection [1] | Here |
| YOLOX | YOLOX-L | ArXiv | 2021 | TensorFlow | TF Re-implementation | Here |
| YOLOv7 | YOLOv7 | ArXiv | 2022 | Pytorch | Official Implementation | Here |

Table 2: All pre-trained panoptic segmentation models used in our evaluations. Models are sorted by publication year.

| Model Name | Model Variant | Venue | Year | Pre-trained Dataset | Implementation | Link |
|---|---|---|---|---|---|---|
| Axial-DeepLab | Axial-SWideRNet-(1, 1, 4.5) | ECCV | 2020 | Cityscape | TF DeepLab2 | Here |
| MaX-DeepLab | MaX-DeepLab-L | CVPR | 2021 | Cityscape | TF DeepLab2 | Here |
| MaX-DeepLab | MaX-DeepLab-L | CVPR | 2021 | COCO | TF DeepLab2 | Here |

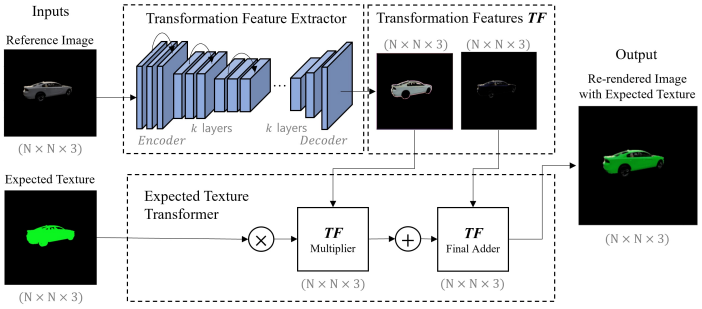
## D. Neural Texture Renderer (NTR)

### D.1. Improvement from Differentiable Transformation Network (DTN)

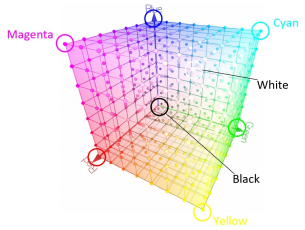
In this section, we present Neural Texture Renderer (NTR), the improvement of DTN by Suryanto et al. [19]. In particular, we found several redundancies in the DTN, especially in the Transformation Features *TF* and training process. Here, we describe per point how we managed to improve DTN in our NTR.

**Redundancy in Transformation Features.** DTN as in [19] uses four *TF*, including *TF Subtractor* + ReLU, *TF Adder*, *TF Multiplier*, and *TF Final Adder* that are assumed to cover all basic transformations. We theorize that the first two layers are redundant; the last two are already sufficient for encoding the *TF*. Therefore, we only use *TF Multiplier* and *TF Final Adder* in our final Transformation Features *TF* that are similar to weight and bias in Neural Network design. Our NTR architecture using the dense connection is depicted in Fig. 5a.

**Redundancy in Dataset.** DTN uses 50 flat random color textures for training the network. We believe that this is inefficient, particularly when we want to train the network with multiple objects. We hypothesize that training the network with eight boundary colors in RGB space and gray $[128, 128, 128]$ as the reference color is enough to make the network generalize other colors. The eight boundary colors consist of the primary colors (red $[255, 0, 0]$, green $[0, 255, 0]$, blue $[0, 0, 255]$), secondary colors (magenta $[255, 0, 255]$, yellow $[255, 255, 0]$, cyan $[0, 255, 255]$), white $[255, 255, 255]$, and black $[0, 0, 0]$, as in Fig. 5b.




(a) Our Implemented NTR Architecture




(b) Selected Boundary Colors in RGB Space

Figure 5: Neural Texture Renderer (NTR) Architecture and Improvement from DTN

## D.2. NTR Evaluation

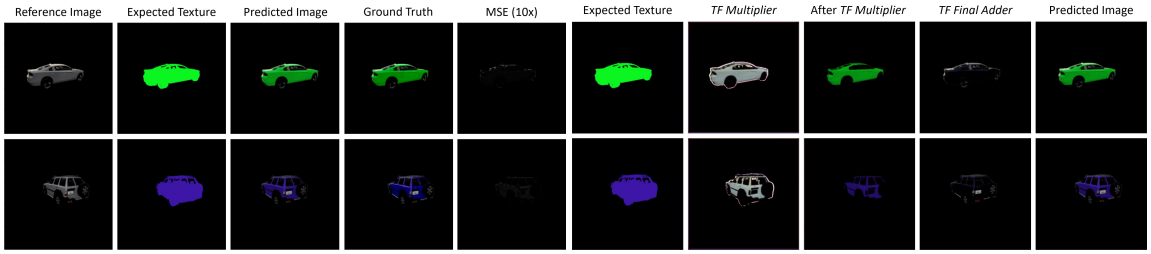
We then perform experiments to evaluate our hypotheses about redundancy in the DTN design. In particular, we train several models using the same machine, and each model is trained with 32 batch sizes for 20 epochs using a single Nvidia Tesla V100 GPU and Intel Xeon E5-2696 CPU. Evaluation results are as shown in Tab. 3, with DTN as the baseline. From the table, we can infer that training the model with only eight boundary colors and gray is enough to enable generalizing the test set, resulting in 82% efficiency. Furthermore, removing redundant *TF* does not hurt the performance, showing that the last two *TF* are enough to encode the expected transformation features.

Table 3: NTR Evaluation

| Model | N Colors | Total Dataset | Training Time | Tess Loss | Test SSIM |
|---|---|---|---|---|---|
| DTN (/w dense connection, k_layers=4) | 50 | 281,250 | 20.4 hours | 0.0312 | 0.986 |
| + Trained with Boundary Colors | 9 | 50625 | 3.7 hours | 0.0314 | 0.985 |
| ++ Removed Redundant *TF* | 9 | 50625 | 3.7 hours | 0.0314 | 0.985 |

## D.3. NTR Sample Predictions

Here, we present the sample prediction of NTR on the test set after the training is completed. Fig. 6a shows sample prediction result consisting of the reference image containing the target object, expected texture, predicted image (i.e., the output of NTR), ground truth, and MSE (i.e., the error between the predicted image and ground truth). As shown, NTR can learn how to render the expected texture given the reference image. Fig. 6b shows the process of transforming expected texture using the encoded *TF*. We can observe that *TF* multiplier is enough to encode the color transformation with the expected texture. *TF* multiplier can encode how the texture is minimized or emphasized and remove the non-applied texture part. The *TF* final adder enhances the image from the *TF* multiplier and adds the non-applied texture part from the reference image.




(a) NTR sample predictions on test set

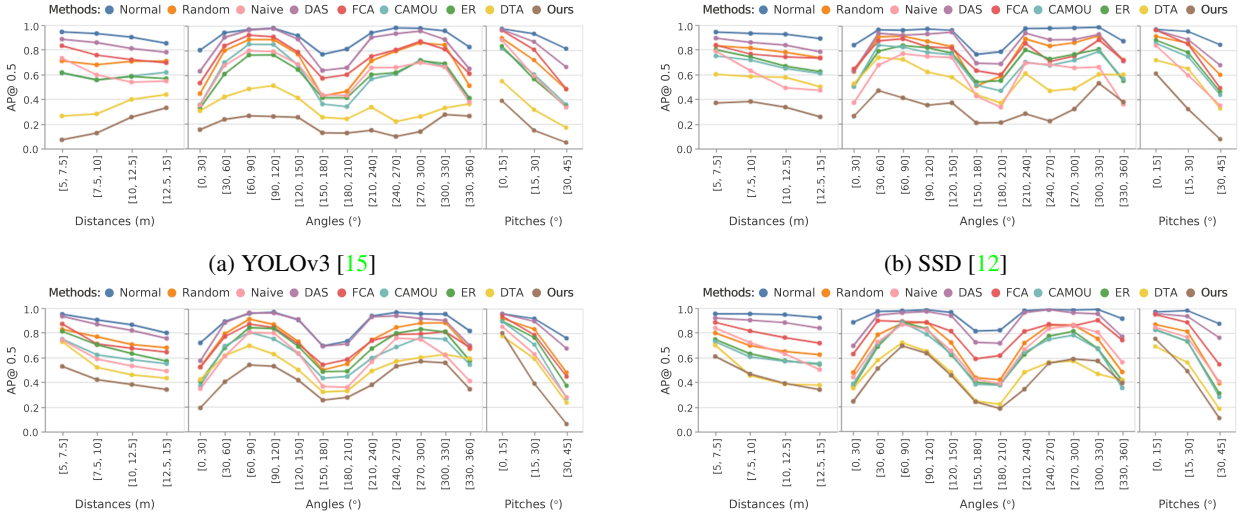
(b) Transforming expected texture with the encoded *TF*

Figure 6: NTR Sample Predictions

# E. Robustness Evaluation Details

## E.1. Detailed Performance on Various Camera Poses

We evaluate YOLOv3 [15] (a white-box model), and SSD [12], Faster R-CNN [16], Mask R-CNN [7] (which are black-box models) for each distance, rotation, and pitch. As shown in Fig. 7 and 8, our method has better attack performance than other methods and is relatively stable under various camera poses in most cases.




(a) YOLOv3 [15]

(b) SSD [12]

(c) Faster R-CNN [16]

(d) Mask R-CNN [7]

Figure 7: Attack comparison on different camera poses. Values are Average Precision @0.5 of the target car.

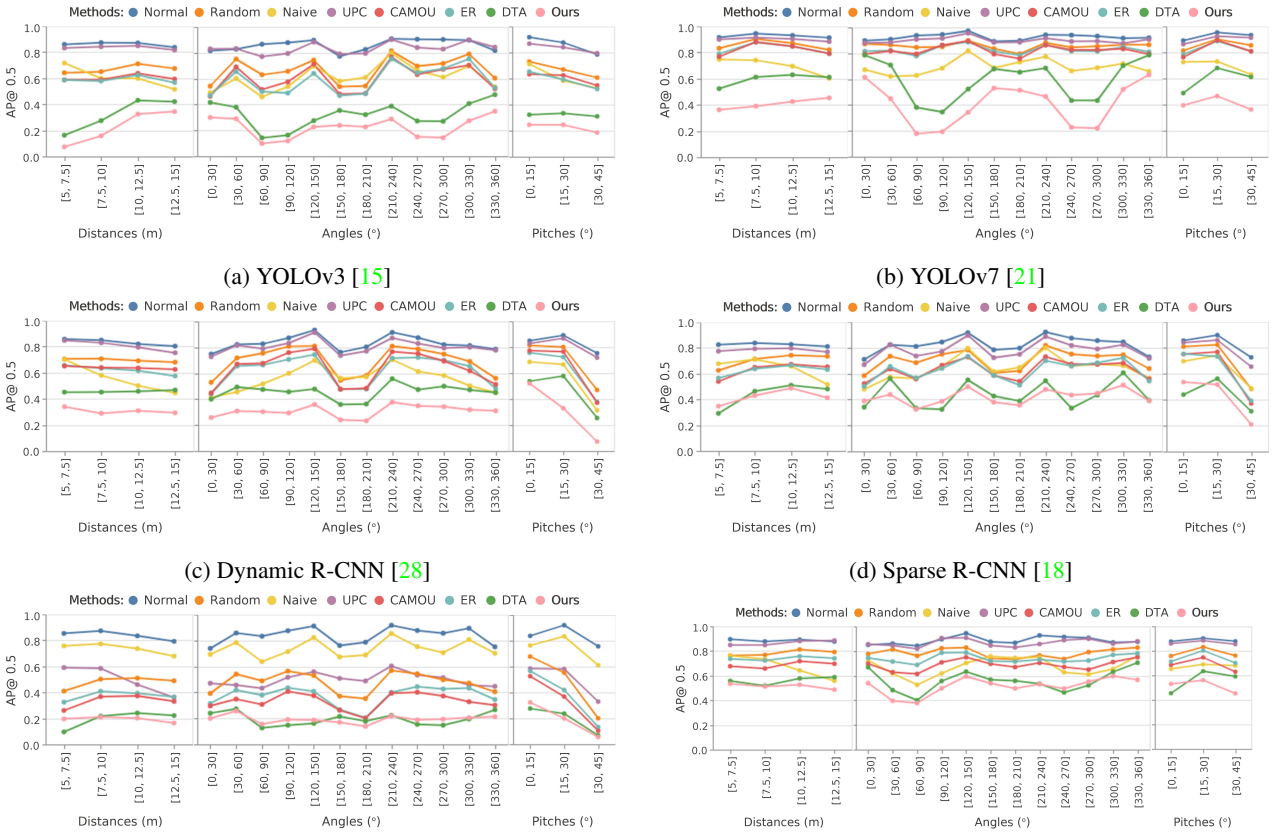

Figure 8: Sample predictions for each method and evaluated model. Green = correct; Yellow = partially correct; Red = misdetection. Zoom for detail.

## F. Universality Evaluation Details

We provide the detailed experimental results of the universality evaluation. First, we evaluate the transferability by trying out our method in different settings (i.e., different from the settings used in the robustness evaluation). We also evaluate the transferability on different classes (i.e., truck and bus) and a different task (i.e., segmentation models). Finally, we fabricate the car models with a 3D printer to evaluate our method's transferability to the real world.

### F.1. Transferability to Different Settings

Here, we use different settings from the robustness evaluation: different cars (as in Fig. 2b), different scenes, and diverse modern object detection architecture such as YOLOv7 [21], Dynamic R-CNN [28], Sparse R-CNN [18], Deformable DETR (DDTR) [30], and Pyramid Vision Transformer (PVT) [26]. As shown in Fig. 9, it is observed that our method outperforms previous works in most cases, and even when being evaluated in different settings as a black-box attack. Sample prediction for each model can be seen in Fig. 10.




(a) YOLOv3 [15]

(b) YOLOv7 [21]

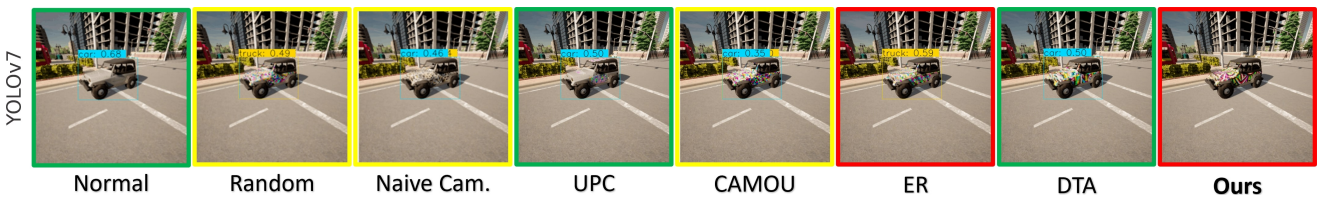
(c) Dynamic R-CNN [28]

(d) Sparse R-CNN [18]

(e) Deformable DETR [30]

(f) Pyramid Vision Transformer [26]

Figure 9: Universality evaluation on different camera poses. Target objects, models, and scenes differ from the training setting. Values are Average Precision @0.5 of the target car.



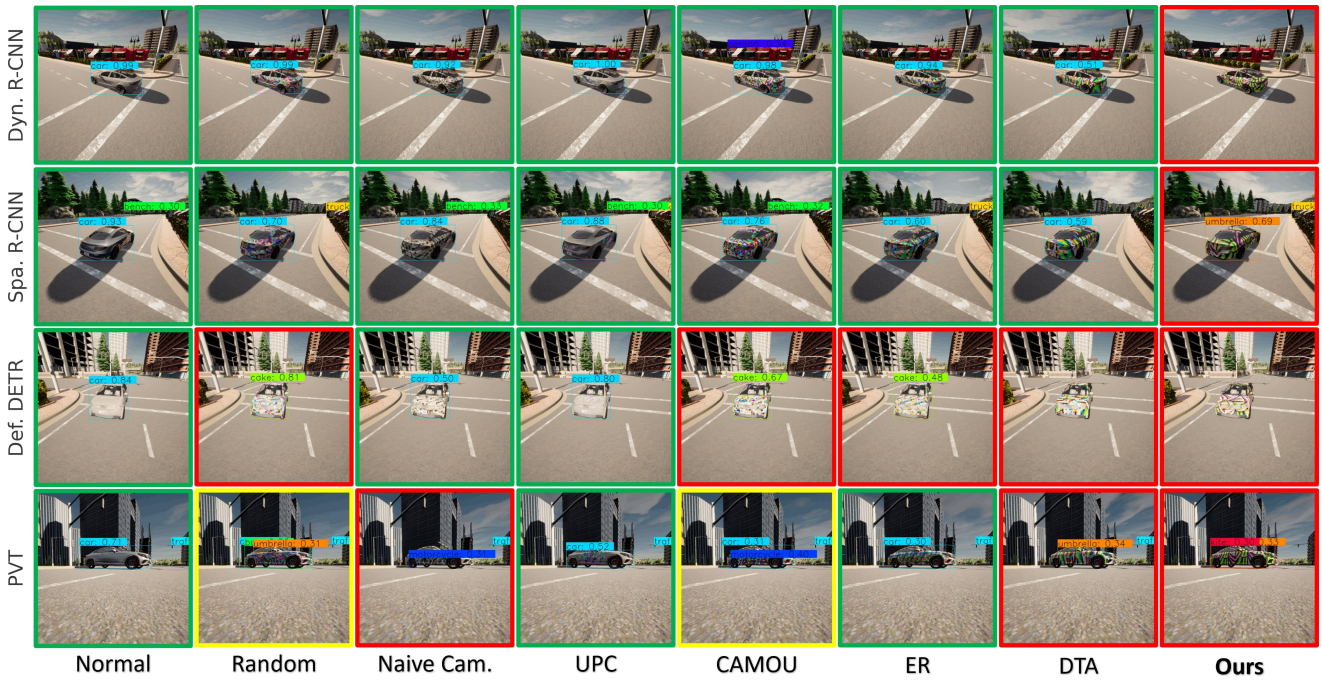



Figure 10: Sample predictions for each method on various models and cars. Green = correct; Yellow = partially correct; Red = misdetection. Zoom for detail.

## F.2. Transferability to Different Class (Truck and Bus)

We perform evaluations to inspect whether the proposed adversarial camouflage can be transferred to other vehicle classes. We select a truck and a Volkswagen bus —which are available on the CARLA —as the target objects, and use the same textures and evaluated models from the previous evaluation. The detailed attack performance comparison for each method and evaluated model can be seen in Fig. 11, showing that our camouflage still performs best compared to others. Additionally, YOLOv7 sample prediction for each method and object can be seen in Fig. 12.

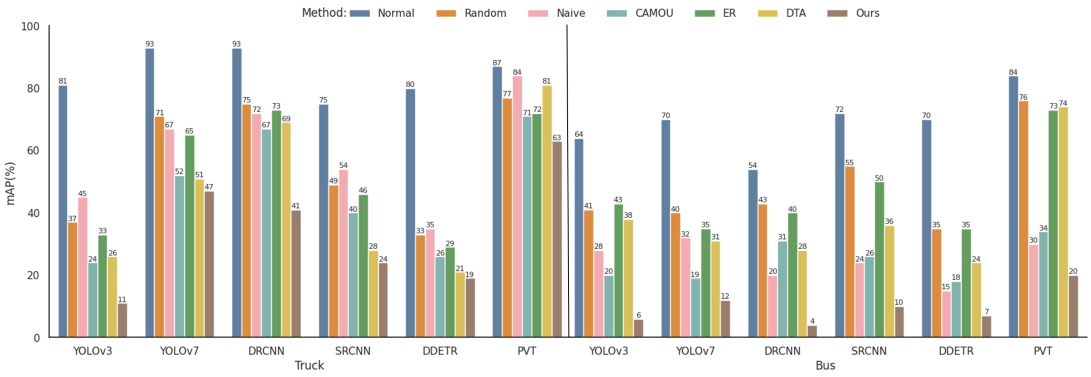
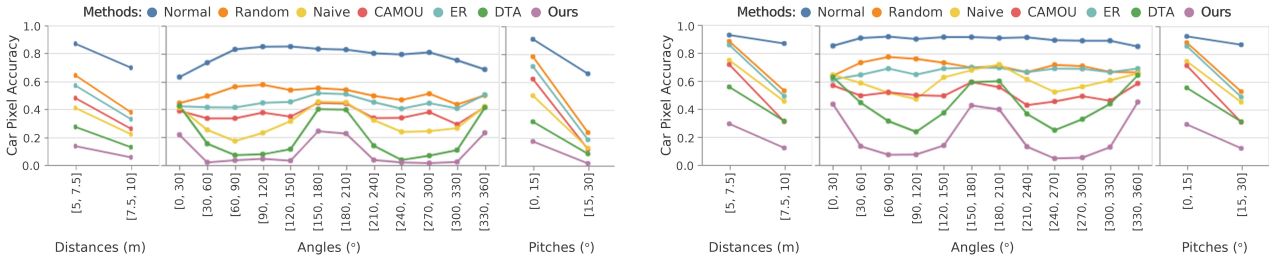



Figure 11: Transferability to different vehicle classes (truck and bus).

## F.3. Transferability to Different Task (Segmentation Models)

We use segmentation models such as MaX-DeepLab-L [23] and Axial-DeepLab [24] to evaluate the transferability to different tasks. We evaluate the pixel accuracy of the car label to show how the adversarial camouflage can decrease the prediction of the target object. Fig. 13 shows that our method yields significantly higher attack performance compared to other methods, even for the segmentation models. Sample predictions can be seen in Fig. 14.

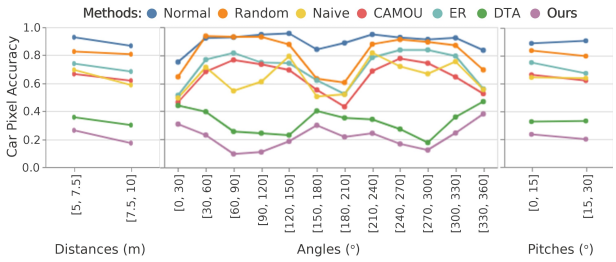

Figure 12: YOLOv7 sample predictions for truck and bus. Green = correct; Yellow = partially correct; Red = misdetection. Zoom for detail.




(a) Cityscape Pretrained MaX-DeepLab-L [23]




(b) Cityscape Pretrained Axial-DeepLab [24]




(c) COCO Pretrained MaX-DeepLab-L [23]

Figure 13: Transferability to segmentation models on different camera poses. Values are Pixel Accuracy of the target car.

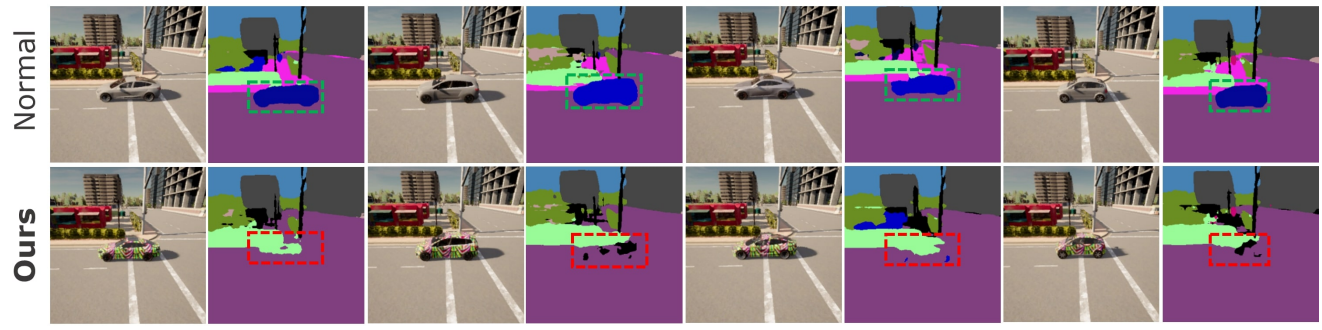
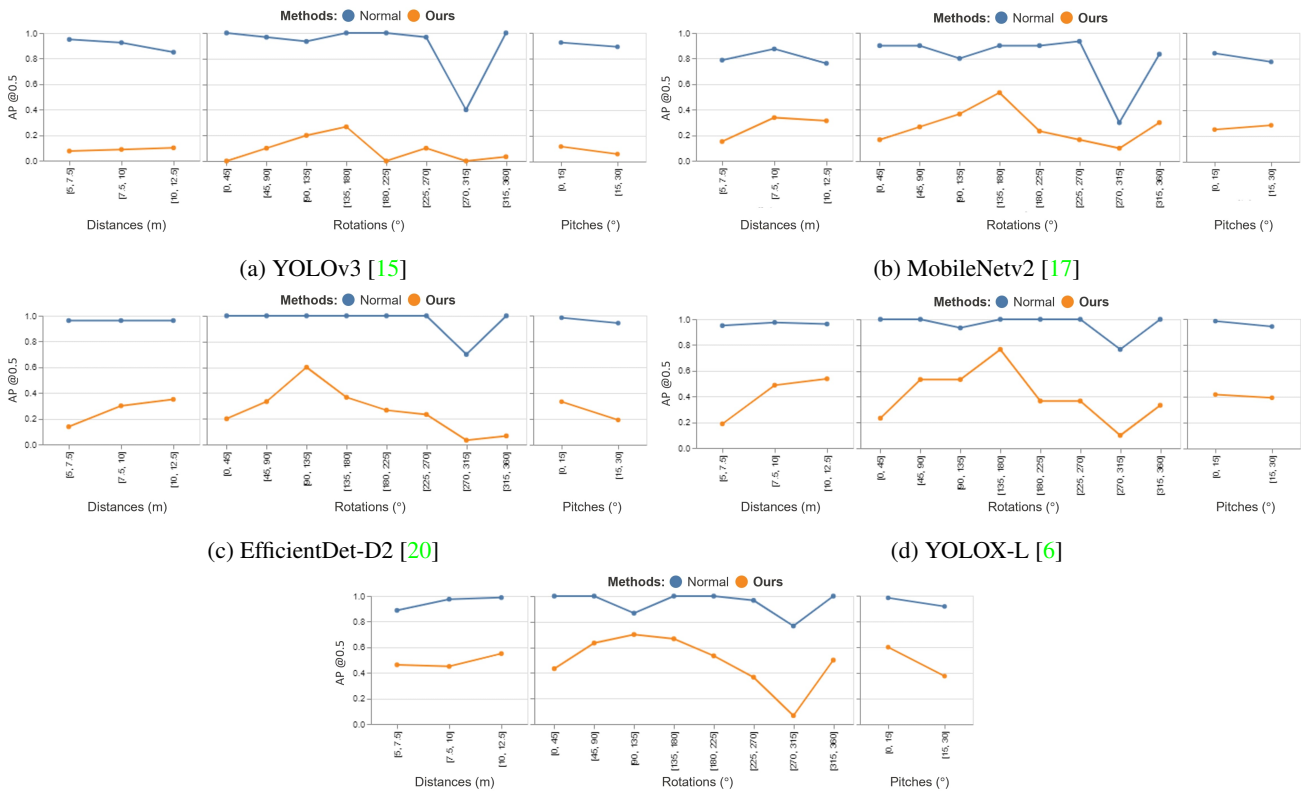



Figure 14: Sample predictions of normal and our camouflage car on Axial-DeepLab. The camouflage works universally on multiple cars.

## F.4. Transferability to the Real World

We conduct a real-world experiment by fabricating two 1:10-scaled Tesla Model 3s with a 3D printer: one for a normal and another for our camouflaged car with the texture targeting YOLOv3. We select the adversarial camouflage with the best attack performance (i.e., the lowest AP of YOLOv3) to reliably measure the transferability to the real world when producing a camouflaged car. We place the car models in five real-world locations and capture car images for every 45-degree interval in angles ([0, 45], [45, 90], . . . , [315, 360] degrees) with three different distances ([5, 7.5], [7.5, 10], [10, 12.5] meters) and two different pitches ([0, 15], [15, 30] degrees) using iPhone 11 Pro Max. As a result, a total of 240 images (5 locations $\times$ 8 rotations $\times$ 3 distances $\times$ 2 pitches) are taken for each camera pose, and 240 images for each car model. In summary, a total of 480 images are used for evaluation.

We evaluate practical real-time object detectors widely used in the real world: MobileNetV2 [17], EfficientDet-D2 [20], YOLOX-L [6], and YOLOv7 [21]. From Fig. 15, our adversarial camouflaged car can significantly decrease AP@0.5 of all object detection models in the real world. Fig. 16 shows that the normal car model is well detected, whereas the camouflaged car model is not detected as a car at all under various camera poses.




(a) YOLOv3 [15]

(b) MobileNetv2 [17]

(c) EfficientDet-D2 [20]

(d) YOLOX-L [6]

(e) YOLOv7 [21]

Figure 15: Transferability to real world on different camera poses. Values are Average Precision @0.5 of the target car.

(a) Normal car. Cars can be consistently detected with relatively high scores.

(b) Our camouflaged car. It can evade most of the detection with a high success rate, while background objects can still be detected correctly.

Figure 16: Sample predictions of our real-world evaluation using two scaled car models. We randomly sample 240 real-world images for a single car consisting of five locations, three distances, eight rotations, and two pitch angles as camera transformation sets. Zoom for detail.

## G. Ablation Study

### G.1. Effect of Different Losses With Respect to Performance

We perform a detailed analysis of our loss hyperparameters, including our smooth loss $\beta$ and camouflage loss $\gamma$ with respect to the attack performance in the white-box setting (targeting YOLOv3). We fix our stealth loss $\alpha$ as 1.0 and run a grid search for both $\beta$ and $\gamma$ with the value in the range of $[0.0, 0.25, 0.5, 0.75, 1.0]$. As shown in Fig 17, the smooth loss has a better correlation for increasing the attack performance than camouflage loss. Also, as mentioned by [13], extreme differences between adjacent pixels in the perturbation are unlikely to be accurately captured by cameras and may not be physically realizable; thus, the smooth loss becomes important in physical adversarial attacks, as verified in our experiment. On the other hand, utilizing our camouflage loss may result in a trade-off between attack performance and naturalness since it suppresses texture by using colors close to the extracted dominant background, limiting the use of other colors to enhance the attack. This can be seen from the AP@0.5 grouped by camouflage $\gamma$ showing that $\gamma = 0$ has the lowest mean compared to others.

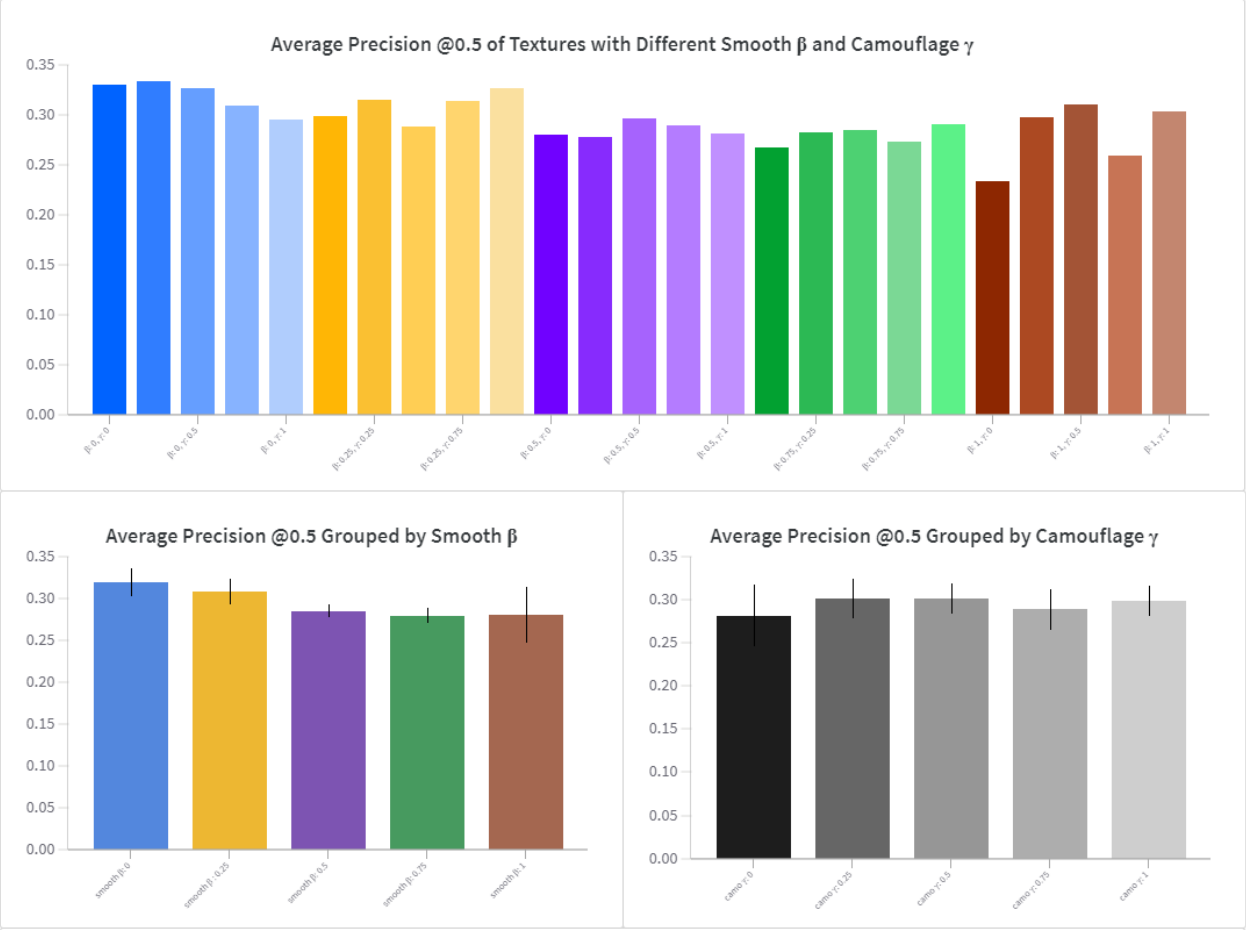



Figure 17: Performance graph for various smooth loss $\beta$ and camouflage loss $\gamma$.

### G.2. Effect of Different Losses with Respect to Naturalness

Here, we visualize the effect of different losses with respect to texture naturalness. As shown in Fig 18, we can see that both smooth and camouflage loss have a role in texture naturalness. We also observe that higher smooth $\beta$ results in lower pixels variance, yielding a smooth texture, while higher camouflage $\gamma$ results in softer color texture. Additionally, only utilizing the smooth loss may result in a bright and garish color texture, attracting more human attention, whereas only employing camouflage loss may result in a rough texture. Thus, utilizing both smooth and camouflage loss yields in a more natural result with a smoother and softer texture.

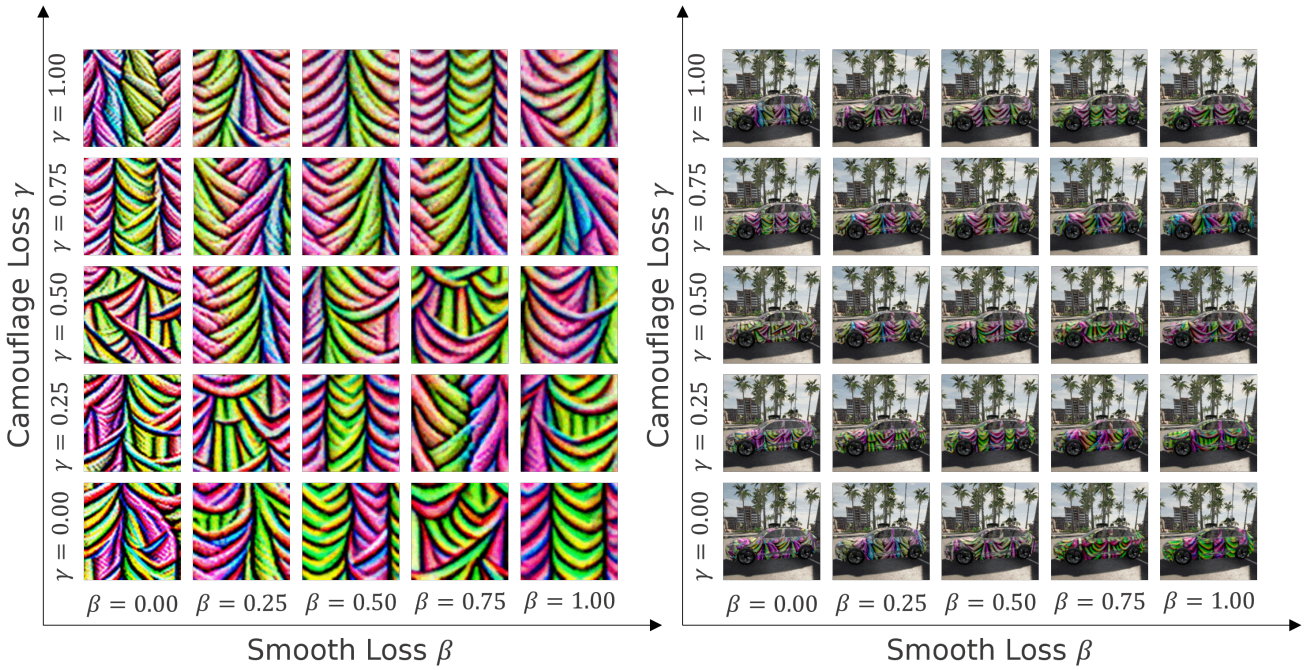



Figure 18: Our optimized adversarial camouflage texture with various smooth loss $\beta$ and camouflage loss $\gamma$: base texture (left), after rendered in UE4 (right).

### G.3. Effect of Different Target Models

We conduct experiments using a state-of-the-art (SOTA) model to assess how different target models affect attack performance. The results of our experiments are presented in Table 4, indicating that when targeting YOLOv7, a more robust model, we obtain a corresponding increase in the robustness of the texture.

Table 4: Universality evaluation on SOTA target model (YOLOv7)

| **Methods** (Target) | **Evaluated Model - Car AP@0.5** YOLOv3 | YOLOv7 | DRCN | SRCN | DDETR | PVT |
|---|---|---|---|---|---|---|
| Normal | 85.98 | 92.54 | 83.20 | 82.55 | 83.83 | 88.59 |
| Ours (YOLOv3) | **22.55** | 41.55 | 30.38 | 42.00 | 14.69 | 51.54 |
| **Ours (YOLOv7)** | 33.13 | **20.98** | **15.33** | **17.98** | **10.11** | **32.43** |

## References


[1] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. MMDetection: Open mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*, 2019. 4

[2] Liang-Chieh Chen, Huiyu Wang, and Siyuan Qiao. Scaling wide residual networks for panoptic segmentation. *arXiv:2011.11675*, 2020. 4

[3] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 4

[4] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017. 1

[5] Epic Games. Unreal engine. 3

[6] Zheng Ge, Songtao Liu, Feng Wang, Zeming Li, and Jian Sun. Yolox: Exceeding yolo series in 2021. *arXiv preprint arXiv:2107.08430*, 2021. 4, 10

[7] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017. 4, 6

[8] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7310–7311, 2017. 4

[9] Lifeng Huang, Chengying Gao, Yuyin Zhou, Changqing Zou, Cihang Xie, Alan L. Yuille, and Ning Liu. UPC: learning universal physical camouflage attacks on object detectors. In *CVPR*, volume abs/1909.04326, 2019. 3

[10] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3d mesh renderer. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3907–3916, 2018. 3

[11] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014. 4

[12] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016. 4, 6

[13] Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *CVPR*, pages 5188–5196. IEEE Computer Society, 2015. 12

[14] Kris Nicholson and Ashish Naicker. Gpu based algorithms for terrain texturing. 2008. 1, 2

[15] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018. 3, 4, 6, 7, 10

[16] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015. 3, 4, 6

[17] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018. 4, 10

[18] Peize Sun, Rufeng Zhang, Yi Jiang, Tao Kong, Chenfeng Xu, Wei Zhan, Masayoshi Tomizuka, Lei Li, Zehuan Yuan, Changhu Wang, et al. Sparse r-cnn: End-to-end object detection with learnable proposals. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 14454–14463, 2021. 4, 7

[19] Naufal Suryanto, Yongsu Kim, Hyoeun Kang, Harashta Tatimma Larasati, Youngyeo Yun, Thi-Thu-Huong Le, Hunmin Yang, Se-Yoon Oh, and Howon Kim. Dta: Physical camouflage attacks using differentiable transformation network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15305–15314, June 2022. 1, 3, 4

[20] Mingxing Tan, Ruoming Pang, and Quoc V Le. Efficientdet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10781–10790, 2020. 4, 10

[21] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. *arXiv preprint arXiv:2207.02696*, 2022. 4, 7, 10

[22] Donghua Wang, Tingsong Jiang, Jialiang Sun, Weien Zhou, Zhiqiang Gong, Xiaoya Zhang, Wen Yao, and Xiaoqian Chen. Fca: Learning a 3d full-coverage vehicle camouflage for multi-view physical adversarial attack. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 2414–2422, 2022. 3

[23] Huiyu Wang, Yukun Zhu, Hartwig Adam, Alan Yuille, and Liang-Chieh Chen. Max-deeplab: End-to-end panoptic segmentation with mask transformers. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5463–5474, 2021. 4, 8, 9

[24] Huiyu Wang, Yukun Zhu, Bradley Green, Hartwig Adam, Alan Yuille, and Liang-Chieh Chen. Axial-deeplab: Stand-alone axial-attention for panoptic segmentation. In *European Conference on Computer Vision*, pages 108–126. Springer, 2020. 4, 8, 9

[25] Jiakai Wang, Aishan Liu, Zixin Yin, Shunchang Liu, Shiyu Tang, and Xianglong Liu. Dual attention suppression attack: Generate adversarial camouflage in physical world. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8561–8570, 2021. 3

[26] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 568–578, 2021. 4, 7

[27] Tong Wu, Xuefei Ning, Wenshuo Li, Ranran Huang, Huazhong Yang, and Yu Wang. Physical adversarial attack on vehicle detector in the carla simulator. *CoRR*, abs/2007.16118, 2020. 3

[28] Hongkai Zhang, Hong Chang, Bingpeng Ma, Naiyan Wang, and Xilin Chen. Dynamic r-cnn: Towards high quality object detection via dynamic training. In *European conference on computer vision*, pages 260–275. Springer, 2020. 4, 7

[29] Yang Zhang, Hassan Foroosh, Philip David, and Boqing Gong. Camou: Learning physical vehicle camouflages to adversarially attack detectors in the wild. In *International Conference on Learning Representations*, 2018. 3

[30] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable {detr}: Deformable transformers for end-to-end object detection. In *International Conference on Learning Representations*, 2021. 4, 7