This CVPR 2020 paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the accepted version;
the final published version of the proceedings is available on IEEE Xplore.

3D Part Guided Image Editing for Fine-grained Object Understanding

Zongdai Liu[† 1], Feixiang Lu[† 2,6], Peng Wang[§ 2,5], Hui Miao[1], Liangjun Zhang[2,6],
Ruigang Yang[2,3,6] and Bin Zhou[* 1,4]

[1]State Key Laboratory of Virtual Reality Technology and Systems, Beihang University
[2]Robotics and Autonomous Driving Laboratory, Baidu Research [3]University of Kentucky
[4]Peng Cheng Laboratory, Shenzhen, China [5]ByteDance Research
[6]National Engineering Laboratory of Deep Learning Technology and Application, China

Abstract

*Holistically understanding an object with its 3D movable parts is essential for visual models of a robot to interact with the world. For example, only by understanding many possible part dynamics of other vehicles (*e.g.*, door or trunk opening, taillight blinking for changing lane), a self-driving vehicle can be success in dealing with emergency cases. However, existing visual models tackle rarely on these situations, but focus on bounding box detection. In this paper, we fill this important missing piece in autonomous driving by solving two critical issues. First, for dealing with data scarcity, we propose an effective training data generation process by fitting a 3D car model with dynamic parts to cars in real images. This allows us to directly edit the real images using the aligned 3D parts, yielding effective training data for learning robust deep neural networks (DNNs). Secondly, to benchmark the quality of 3D part understanding, we collected a large dataset in real driving scenario with cars in uncommon states (CUS),* i.e. *with door or trunk opened etc., which demonstrates that our trained network with edited images largely outperforms other baselines in terms of 2D detection and instance segmentation accuracy.*

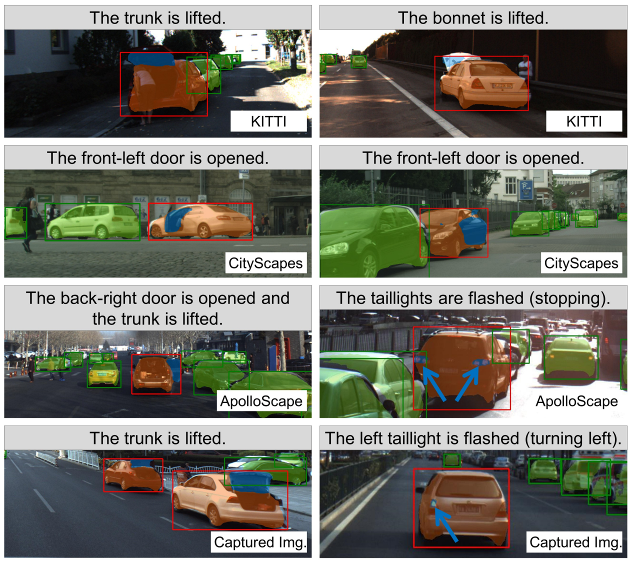
The trunk is lifted.
KITTI
The bonnet is lifted.
KITTI
The front-left door is opened.
CityScapes
The front-left door is opened.
CityScapes
The back-right door is opened and the trunk is lifted.
ApolloScape
The taillights are flashed (stopping).
ApolloScape
The trunk is lifted.
Captured Img.
The left taillight is flashed (turning left).
Captured Img.

Figure 1: Fine-grained parsing of cars in uncommon states on various datasets. The results include 2D detection (red bounding box), instance segmentation (orange mask), dynamic part segmentation (blue mask), and state description. Note that the common-state cars are with green color.

1. Introduction

An object, *e.g.* a car or a person, is commonly composed with articulated and movable 3D parts [47, 24]. Understanding an object with its 3D parts and their future states within images or videos is essential for the vision/perception system of a robot to decide its actions to interact with the world. For example, in the popular autonomous driving (**AD**) scenario, when a car parking on the road has its door opened, it will be very likely that someone would get off. As a response, the autonomous vehicle should immediately slow down, turn the steering wheel, and change line. Though, this case is not common, it is deadly if there is no such understanding behind, and in real driving scenario, there are many such cases as illustrated in Fig. 1.

However, the dominant visual perception systems with deep neural networks, though achieved great success in 2D/3D detection [34, 12], instance segmentation [17] and pose estimation [4, 22, 39], are based on coarse understanding of objects with bounding boxes or masks. In our opin-

[†]Joint first author
[§]Partial work is done at Baidu Research
[*]Corresponding author: zhoubin@buaa.edu.cn

ion, this is not sufficient for performing actions respect to 3D part dynamics of vehicles on the street.

This paper is a step forward to fill this missing piece, especially in AD scenario, by providing a model that enables detailed 3D part parsing of an object. To perform such a task, we first look through many popular AD datasets, such as KITTI [14], CityScapes [6] and ApolloScape [20, 46]. As shown in Fig. 1, we found firstly, cases where a car has its part moved as we discussed are existing in real driving scenarios. Secondly, the amount of cases is too scarce, *e.g.* only tens of cars, to train an effective model to dealing with 3D part understanding when these cases happened.

To generate enough amount of data for training a model understanding 3D parts, the common strategy is manually crowd sourcing large amount of real images [15], which will be labor expensive, while other solutions such as obtaining dataset with simulated environment and computer graphics [1, 45, 31] will have strong domain gap of car and scene appearance to realistic scenarios. To balance the two and automatically generate training data for current deep learning models [17], here we propose a 3D part guided image editing strategy, as illustrated in Fig. 2, by first fitting a 3D car model with dynamic parts in images, then re-rendering the car inside with re-configured parts and the realistic texture. Specifically, we adopt models from the ApolloCar3D [39] dataset where each car instance is fitted with a 3D model, and we define ten commonly happened dynamic parts, *i.e.*, bonnet, trunk, four doors, two headlights and two taillights, for each type of 3D car model. More specifically, for each part, we labelled its motion axis which constraints the range of possible movement. By sampling all the possible motion of a 3D car instance, our strategy automatically edit the 2D car instance inside images, yielding a large number of training samples.

Based on the generated data, we design and train a multitask network performing object understanding with fine granularity, including 2D detection, instance segmentation, dynamic part segmentation, and 3D car state description. Our deep model is significantly more robust in understanding cars in AD than models without our generated dataset.

Finally, to benchmark our model and strategies, to our best knowledge, we construct the first dataset with large amount of described uncommon states of cars in AD, *i.e.* with door or trunk open etc., which contains 1441 labelled street-view images, 1850 car instances, and 12 defined states. We evaluate the part understanding quality extensively with the dataset, and show our network and training strategies yields large improvements (over 8% relatively) in discovering and understanding these uncommon cases.

In summary, our contributions are in three aspects:

- We present a 3D part guided image editing pipeline for automatic training data generation, which helps to learn fine-grained object understanding models in AD.
- We design a multi-task network architecture which produces output of both instance level and part level object understanding.
- To benchmark our data generation strategies, and network architectures, we build a large dataset which contains 1441 real images with fine-grained annotation of objects in many uncommon states. It demonstrates the effectiveness of our approaches.

2. Related Work

Fine-grained object understanding is one of the center problem for autonomous driving. Our work is mostly related to two areas: datasets and vehicle for fine-grained parsing. We review the related works in the following.

Datasets for Autonomous Driving. Focusing on perception in autonomous driving, several datasets have been constructed and released. The first dataset is CamVid [2], which annotates 701 images with 32 semantic classes. The later released KITTI benchmark [14] contains multiple vision tasks (*e.g.*, optical flow, 2D/3D detection). However, it mainly annotates 2D/3D bounding boxes for each car, resulting in 7481 training and 7518 test images. Recently, CityScapes dataset [6] labelled vehicles with instance-level segmentation, which released 2975 training, 500 validation, and 1525 test images. ApolloScape [20] is a large-scale AD dataset for various 2D and 3D tasks. It performs pixels-level annotations for 2D scene parsing, providing about 140K images. ApolloCar3D [39] is a 3D instance car dataset built from real images in driving scenes. For each car instance in 2D image, 3D model and corresponding 6-DoF pose are manually labelled. Moreover, there exist other real street-view self-driving datasets (*e.g.*, Toronto [48], Mapillary [30], and BDD100K [51]) and synthetic datasets (*e.g.*, SYNTHIA [37], P.F.B. [35], and Virtual KITTI [11]). However, all of these datasets only annotate common cars with 2D bounding box or semantic/instance segmentation, while cars in **uncommon states** are ignored (*e.g.*, opened door or trunk, and flashed headlights or taillights). In an AD scenario, this information can predict further action of the vehicle, which becomes very important for safety.

Data Generation for Deep Network. Learning effective deep networks (*e.g.*, AlexNet [21], VGG [38], ResNet [18], and FPN [25]), depends on large amount of training data for each individual task. However, real data collection and annotation [8, 26, 20] are laborious. To avoid the difficulties of data labelling, synthetic data is widely used for training deep networks. Current image synthesis techniques can be roughly divided into two classes: 3D model rendering and 2D image 'cut-paste' [10]. Recently, several large-scale 3D model datasets have been released, such as ShapeNet [5], ModelNet [49] and ScanNet [7]. Researchers directly render 3D models to obtain 2D im-

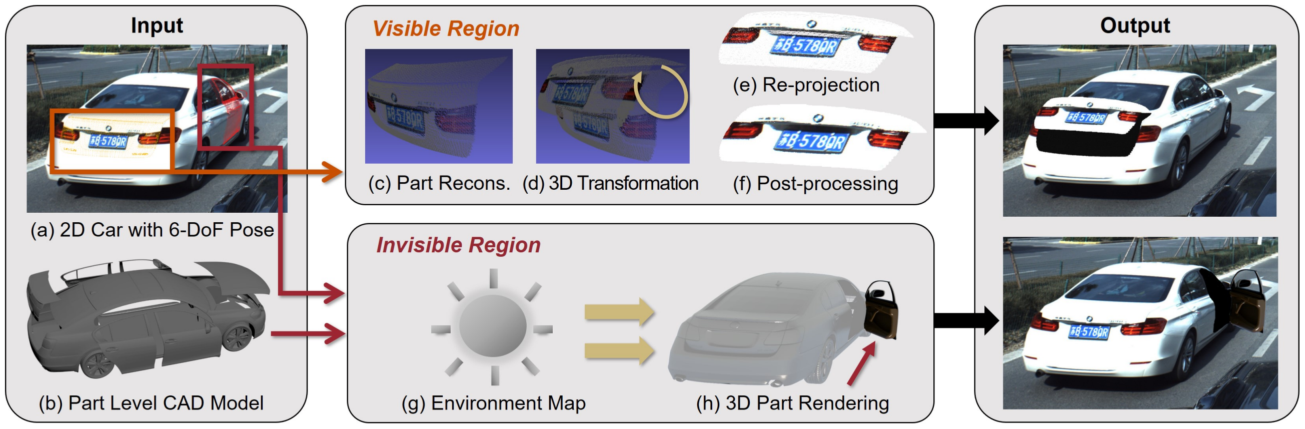

Figure 2: Overview of our data augmentation pipeline.

ages for training. However, rendering is a time-consuming work which requires pre-building the complex realistic 3D scenes. Therefore, some works cut objects from images, and then paste to other background to synthesize photo-realistic training data. However, the diversity of 'cut-paste' results is limited. Furthermore, it cannot handle the problem of occlusion.

Nevertheless, many computer vision tasks are beneficial from synthetic data, such as optical flow [3, 9], scene flow [28], stereo [32, 52], semantic segmentation [36, 37], 3D keypoint extraction [42], viewpoint [40], object pose [29], 3D reconstruction [16], and object detection [1, 13, 31, 45]. The key problem for these works is to fix appearance domain gap to realistic images. Domain randomization [43] is widely used for vehicle detection [31, 45], which gets the optimal performance. Alhaija *et al.* [1] take advantage of AR approach to overlay vehicle rendering results to the real street-view images, yielding augmented photo-realistic training data. Hinterstoisser *et al.* [19] show that by freezing a pre-trained feature extractor can train a good object detector with synthetic data only.

Fine-grained Parsing and Understanding. For AD, as discussed in Sec. 1, it is important to detect, segment, and parse the moving objects into part-level semantics. Here, state-of-the-art (SOTA) methods often rely on detecting and understanding pipelines. Specifically, an object is first separated using detectors such as one-stage methods (*e.g.*, SSD513 [12], YOLOv3 [33]) or two-stage methods (*e.g.*, Faster-RCNN [34], Mask-RCNN [17]); and then performed fine-grained recognition with object parts, such as part keypoints regression [39] and part segmentation [47, 50]. Most recently, Lu *et al.* [27] extend the part-level pixel-wise annotation to the part state inference problem, such that visual models can be more instructive. Our work follows this trend, while extends previous works with object part understanding in 3D to handle uncommon cases in AD scenario.

3. 3D Part Guided Image Editing

In this section, we introduce how to leverage the 3D parts to automatically edit the source 2D images. To achieve this goal, four essential components are required: 1) 3D part segmentation and motion axis annotation; 2) 3D transformation and 2D projection; 3) hole filling and image filtering; 4) invisible region generation.

Recently, Song *et al.* [39] published a 2D-3D alignment dataset: ApolloCar3D, which annotates the 3D model and 6-DoF pose for each 2D car instance. Based on the released 3D CAD models of cars, we manually segment out the **movable parts** (*i.e.*, bonnet, trunk and four doors) and the **semantic parts** (*i.e.*, two headlights and two taillights), respectively. For semantic parts, we directly project them to obtain the corresponding 2D regions, which are further edited to yellow or red flashed effects (the third row in Fig. 3). For movable parts, we firstly annotate their motion axis, then transform the 3D parts to guide 2D image editing. Note that the 3D models provided by ApolloCar3D are low-quality. It is difficult to obtain appropriate texture map from source image to perform photo-realistic rendering.

Instead, we render the 3D geometry parts to obtain corresponding depth map D, according to the global rotation $\mathbf{R_g}$, translation $\mathbf{t_g}$, and the camera intrinsic matrix $\mathbf{K}$. For each 2D pixel $\mathbf{u} = (u, v)^\top$ with depth value $D(\mathbf{u})$, we convert it to acquire 3D point $\mathbf{P} = (x, y, z)^\top$ through

$$\mathbf{P} = \mathbf{R_g^{-1}} \cdot \big(D(\mathbf{u}) \cdot \mathbf{K^{-1}} \cdot \dot{\mathbf{u}} - \mathbf{t_g}\big). \tag{1}$$

Here, $\dot{\mathbf{u}}$ is a homogeneous vector: $\dot{\mathbf{u}} = (\mathbf{u}^\top|1)^\top$.

Assuming the part locally transformed with a 3D rotation $\mathbf{R_o}$ along with the motion axis, and the axis translate $\mathbf{t_o}$ in the global coordinate. We compute the pixel's new position $\mathbf{u}'$ in the image domain, which is defined as:

$$\mathbf{u}' = \Big\lfloor \pi\Big(\mathbf{K} \cdot \big(\mathbf{R_g}(\mathbf{R_o}(\mathbf{P} - \mathbf{t_o}) + \mathbf{t_o}) + \mathbf{t_g}\big)\Big)\Big\rfloor. \tag{2}$$

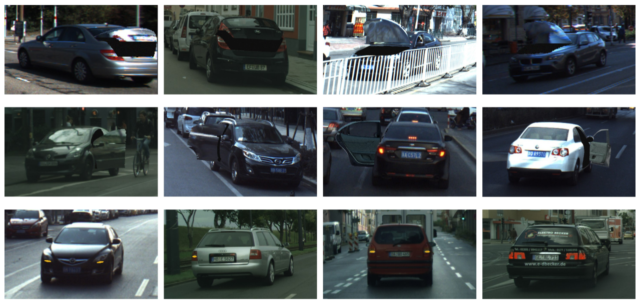

Figure 3: The generated cars in uncommon states by our approach. The editing results of movable parts (*i.e.*, trunk, bonnet, and four doors) are shown in the 1st row and the 2nd row. And the editing results of semantic parts (*i.e.*, two headlights and two taillights) are shown in the 3rd row.

Here, the function $\mathbf{u} = \pi(\mathbf{P})$ performs perspective projection of $\mathbf{P} \in \mathbb{R}^3 = (x, y, z)^\top$ including dehomogenisation to obtain $\mathbf{u} \in \mathbb{R}^2 = (x/z, y/z)^\top$.

Note that the transformed pixels are always sparse and noisy in the part region (Fig. 2 (e)). Here, we call the non-valued pixel as '**hole**'. In order to fill these holes, we perform the linear blending algorithm [41] to obtain the RGB values. After interpolating the non-valued pixels, we apply a bilateral filter [44] on the edited images. The smoothed results are shown in Fig. 2 (f) and Fig. 3.

Invisible region generation. For the case of opening door, we can generate visual compelling results if the car towards the camera. Once the car in the opposite direction, opening door will introduce some invisible regions in the original image. These invisible regions can be roughly divided into two classes: one is the reverse side of the part, another is the vehicle interior (*e.g.*, seat, steering wheel and engine). Empirically, the interior regions are always dark due to the inadequate illumination. Therefore, we directly fill interior regions with the gray color. Also, we have tried the random color and the patches from real images. However, according to the experimental results, we don't find obvious differences among them.

Compared with the interior regions, coloring the reverse side of part seems rather complex. As shown in Fig. 2, it is not appropriate to directly filling in pure color. Thus, we adopt the photo-realistic rendering pipeline to generate high-fidelity results of reverse side. Considering the low-quality models provided by ApolloCar3D, we firstly construct a small expert designed 3D model database for movable parts. Each part is designed by a professional artist with the commercial software, 3dsMax. The part materials are manually labelled and BRDF parameters are predefined. As shown in Fig. 2 (h), we use the environment map calculated [23] from ApolloCar3D to perform photo-realistic rendering. Our editing results are shown in Fig. 3.

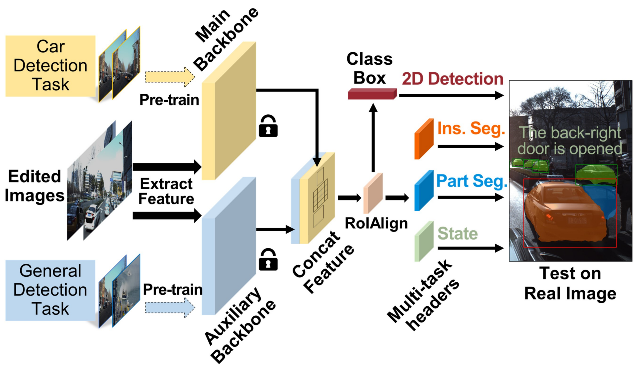

Figure 4: The architecture of our two-backbone network, which can output 2D detection, instance-level segmentation, dynamic part segmentation, and state description.

4. Network Architectures

We propose a novel multi-task deep neural network architecture shown in Fig. 4, which is used for fine-grained object understanding. In this section, we discuss the modules of our network and training settings in details.

4.1. Two Backbones

We aim to detect cars in uncommon states from real street-view images through only training on the editing images. To achieve the transfer ability from synthetic data to real data, there are two ResNet50-FPN [25] backbones in our network. We pre-train the **main backbone** both on ApolloCar3D [39] benchmark and CityScapes [6] benchmark using Mask-RCNN to extract the car body features guided by a car detection task. Simultaneously, we pre-train the **auxiliary backbone** on COCO dataset to learn the general features of the edited region (*e.g.*, the rendered parts) guided by a general detection task. Finally, we fix the parameters of these two backbones to train the network on the editing data. Indeed, experimental results in Sec. 6.4 demonstrate that we can get the optimal performance by freezing two backbones.

4.2. Dynamic Part Segmentation

We adopt the Mask-RCNN [17] to implement the task of dynamic part segmentation. In Mask-RCNN, the mask branch outputs a Km^2 dimensional binary mask for each RoI aligned feature map, where K is the number of class and m is the resolution. Besides, we take the dynamic part segmentation as a new channel, resulting in output containing a $(K + 1)m^2$ binary mask. Specifically, we feed 14×14 RoI aligned feature map to four sequential 256-d 3×3 convolution layers. A 2×2 deconvolution layer is used to up-sample the output to 28×28. Finally, we define the L_{part} as the average of per-pixel sigmoid cross entropy loss.

Datasets	Bonnet	Trunk	Doors				Headlights		Taillights				Total
	lifted	lifted	fl-o.	fr-o.	bl-o.	br-o.	l-tu.	r-tu.	l-tu.	r-tu.	stop	alarm	
KITTI	1	9	1	0	0	5	1	0	2	1	8	0	28
CityScapes	0	0	14	5	8	4	3	2	4	0	15	0	55
ApolloScape	0	23	29	0	59	157	15	18	23	27	33	16	400
ApolloCar3D	0	13	19	1	0	11	3	5	12	9	21	0	94
Capt. Images	15	405	232	66	79	346	19	17	25	18	44	7	1273
CUS Dataset	16	450	295	72	146	523	41	42	66	55	121	23	**1850**

Table 1: The constructed CUS dataset, which annotates 1850 car instances in uncommon states from 1441 street-view images. 'fl-o. (br-o.)' indicates the opened front-left (back-right) part, and 'l-tu. (r-tu.)' indicates turning left (right).

4.3. State Description

We use a binary variable to represent the existence of the particular part state (*i.e.*, 1 for existed and 0 for others). Then, we define the 'part state vector' as a concatenation of all binary variables. Our method regresses the part state vector through the sequential convolution layers and a fully connected layer in mask branch. Similarly, we define the L_{state} as the average sigmoid cross entropy loss.

4.4. Training Details

At first, we pre-train a Mask-RCNN with ResNet50-FPN backbone both on ApolloCar3D [39] benchmark and CityScapes [6] benchmark through a car instance segmentation task. Then, we initialize the main backbone by copying the parameters of the pre-trained network. Simultaneously, we pre-train the auxiliary backbone on COCO dataset using the same network architecture. Finally, we fix the parameters of these two backbones to train the network on the editing data. The multi-task loss is defined as:

$$\begin{aligned} L = L^p_{class} + L^p_{reg} + L^r_{class} + L^r_{box} \\ + L^r_{mask} + L^r_{state} + L^r_{part}, \end{aligned} \tag{3}$$

where $(.)^p$ and $(.)^r$ indicate RPN and RCNN, respectively. The subscript $state$ and $part$ denote the loss of state vector and part mask, respectively. We minimize our loss function using the SGD with a weight decay of 0.0001 and a momentum of 0.9. The learning rate is initially set to 0.002, and reduced by 0.1 for every 5 epochs.

5. CUS Dataset

To our best knowledge, none of existing datasets provides the detailed annotation of *cars in uncommon states* (**CUS**). To evaluate the quality of edited data and benchmark network performance, we construct a **CUS dataset** with real street-view images annotated. Specifically, we firstly look up the existing AD-oriented datasets, including KITTI [14], CityScapes [6], ApolloScape [20], and ApolloCar3D [39]. These four datasets have a total of 80,000 images, which labelled over 1 million car instances. However, very few of them are in uncommon states (Tab. 1).

To add more CUS data, we drive a car to capture images in various sites (*i.e.*, hospital, park, school, and urban road) and in different time (*i.e.*, morning, noon, and afternoon). Consequently, we capture about 150,000 images in total. After removed the blurred and overexposed images, we finally collect 1273 car instances to label.

As shown in Tab. 1, our dataset covers **10** dynamic parts (*i.e.*, bonnet, trunk, four doors, two headlights, and two taillights) and **12** uncommon states, which annotates **1850** car instances from **1441** images. For each car instance, we manually labelled the 2D bounding box, instance segmentation, dynamic part segmentation, and state description. Notice that our trained deep model is used directly for testing on CUS dataset without any 'domain adaptation' or 'fine-turning' strategies. We believe the built benchmark can effectively verify the quality of editing data, and quantitatively evaluate the network performance.

6. Results

6.1. Experimental Settings

Our network is trained on a 64-bit work station with a 8-core 3.4 GHz CPU, 4 Nvidia TITAN XP graphics cards, and Ubuntu 16.04 OS. The generated training data mostly comes from ApolloCar3D dataset which labelled the 3D model and 6-DoF pose for each car instance. Considering the obvious domain gap among different datasets, we further annotate 100 common car instances with 2D-3D aligned in KITTI, CityScapes, ApolloScape, and captured images, respectively. Then we perform the proposed editing approach to generate CUS data for training. The editing time for each car is about 3 seconds. Specifically, 0.5s for 3D points transformation and projection, 0.5s for hole filling and filtering, and 2s for invisible region generation.

The training time of our network depends on the data number. In general, training 25K images costs 24 hours. On the testing phase, we directly use the trained model to perform fine-grained understanding on CUS dataset. As shown

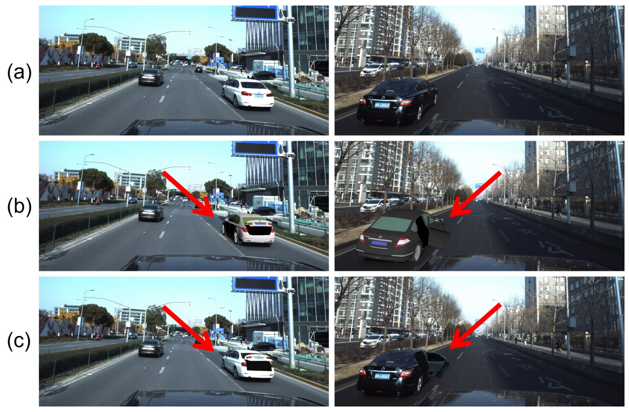

Figure 5: The training data of different approaches: (a) raw images; (b) rendering data; (c) editing data by our approach.

in Fig. 1, Fig. 7 (c) and Fig. 10, our network outputs holistic parsing results, including 2D detection, instance-level segmentation, dynamic part segmentation, and state description. Source code, data and more results can be found on the project page (*https://github.com/zongdai/EditingForDNN*).

6.2. Evaluation Metric

In Sec. 6.3, our network is compared with Mask-RCNN. Note that the proposed benchmark is only focused on **CUS**. While Mask-RCNN cannot distinguish the cars in *common/uncommon* states, which are both existed in the testing data. If we use the *AP* metric to evaluate this experiment, the detected common-state cars will decrease the precision, resulting in inaccurate *AP* value. Therefore, we compute the maximum of ***IoU*** values between the ground truth and the predictions to evaluate the network performance.

Different with Mask-RCNN, our two-backbone network can correctly detect the cars in uncommon states. For the ablation study in Sec. 6.4, we choose the ***mAP*** metric to evaluate the performance of 2D detection, instance segmentation, and part segmentation. For state description, we compute the match rate at each binary item between prediction state vectors and ground truth.

Methods	**2D Detection** (*IoU*)	**Ins. Seg.** (*IoU*)
Baseline 1	0.751	0.704
Baseline 2	0.758	0.712
Baseline 3	0.775	0.721
Baseline 4	0.766	0.713
Baseline 5	0.772	0.719
Ours	**0.862**	**0.815**

Table 2: 2D detection and instance segmentation evaluation results with different approaches on CUS dataset.

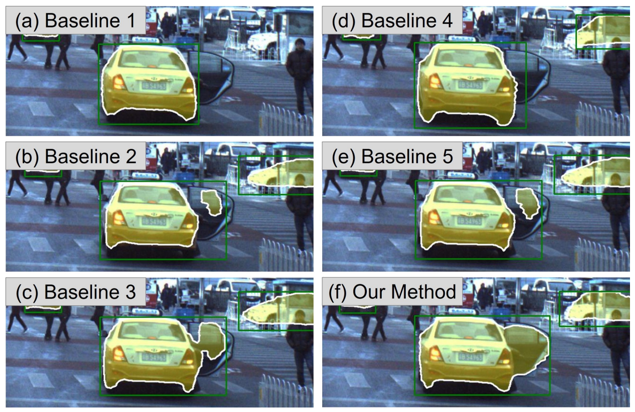

Figure 6: Visualization results on 2D detection and instance segmentation (five baseline methods *vs.* ours).

6.3. Comparison with Baseline Methods

To demonstrate our method can effectively improve the performance on 2D detection and instance-level segmentation, following baseline methods are compared (Tab. 2):

Baseline 1*: Mask-RCNN + Existing Datasets.* We train the Mask-RCNN network on the existing datasets (*i.e.*, KITTI, CityScapes, ApolloScape, and ApolloCar3D), which only annotate the common-state cars (Fig. 5 (a)). In the testing phase, we directly output the results of 2D detection and instance-level segmentation on CUS dataset.

Baseline 2*: Mask-RCNN + Rendering Data.* We implement the rendering-based data generation pipeline which is adopted in most image synthesis works [40, 31, 45]. Following [1], we construct 50 high-quality textured CAD models of cars, which are labelled the dynamic parts and motion axis. We transform the car models according to the 6-DoF pose and operate the dynamic parts to generate uncommon states. Here, we use *Blender* software to obtain rendering result which is further overlaid to background (Fig. 5 (b)). Consequently, we build a rendering dataset which consists of 25K images. We train the Mask-RCNN network on rendering data and test on CUS dataset.

Baseline 3*: Mask-RCNN + Editing Data.* We then train the Mask-RCNN network using our editing data (Fig. 5 (c)) which has the same number with the rendering data. We evaluate the trained Mask-RCNN network on CUS dataset.

Baseline 4*: Our Network + Existing Datasets.* We train our two-backbone network using the existing datasets which are introduced in *Baseline 1*.

Baseline 5*: Our Network + Rendering Data.* We train the proposed two-backbone network using the rendering data which is illustrated in *Baseline 2*.

Our method*: Our Network + Editing Data.* At last, we train our two-backbone network using the editing data. The quantitative results of these approaches are listed in Tab. 2.

Methods	**2D Detection** (*mAP*)	**Instance Seg.** (*mAP*)	**Part Seg.** (*mAP*)	**State Description**
Single Backbone Re-trained	0.136	0.114	0.144	0.149
Single Backbone Frozen	0.672	0.516	0.273	0.837
Two Backbones Frozen	**0.701**	**0.563**	**0.314**	**0.874**

Table 3: Ablation study of our network on 2D detection, instance segmentation, part segmentation, and state description.

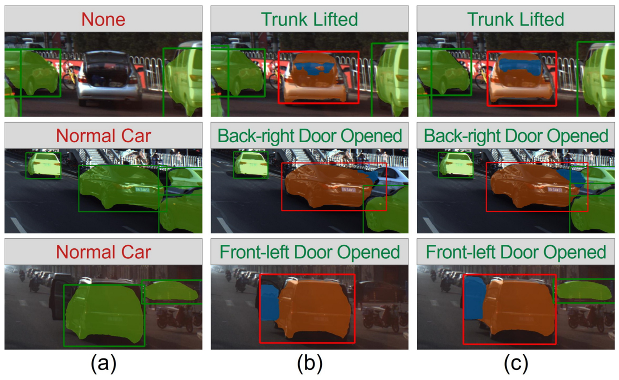

Figure 7: Visual results on ablation study of our network: (a) single backbone re-trained; (b) single backbone frozen; (c) two backbones frozen. The words with red/green color indicate the wrong/correct state descriptions.

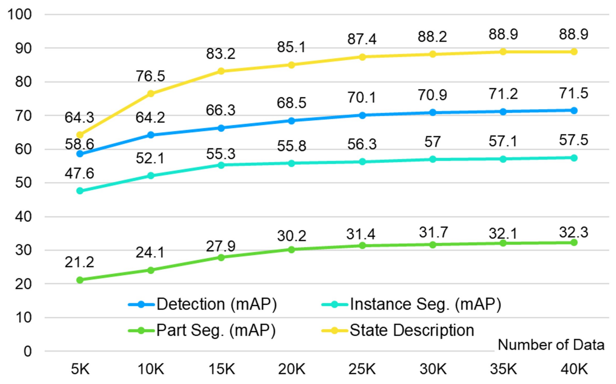

Figure 8: The performance of our two-backbone network with different number of training data.

The results of *Baseline 1* indicate that the model of Mask-RCNN trained by common-state cars can detect and segment the car body. However, the dynamic parts are always ignored (Fig. 6 (a)). The results of *Baseline 2* show that rendering data improves the network performance compared with *Baseline 1*. While the rendering data (Fig. 5 (b)) has the natural domain gap with the real captured images (Fig. 5 (a)). In addition, 3D rendering costs much 10x time than editing-based approach. The results of *Baseline 3* prove that Mask-RCNN trained by editing data outperforms existing datasets and rendering data. However, when we visualize the results of detection and segmentation in Fig. 6 (c), it is clearly shown that the visible parts are good while the reverse side of dynamic part suffers from errors.

Baseline 4 and *Baseline 5* are using our two-backbone network to train on the existing datasets and rendering data, respectively. However, the performance of both baseline methods are not improved significantly. Here, we emphasize that our two-backbone network is carefully designed to learn the editing data, especially the dynamic parts. Directly using our network cannot effectively learn other data, because they are in the different domain. Consequently, our two-backbone network trained by editing data gets the best performance, which advances other methods by over **8** percent on both tasks (Tab. 2). The main improvement comes from the invisible regions (Fig. 6 (f)).

6.4. Performance Analysis

The impact of our network structure. Besides 2D detection and instance segmentation, our network can detect cars in uncommon states, segment dynamic parts, and describe states. To illustrate the impact of our network structure, we conduct an ablation study as shown in Tab. 3 using constant number of training data (*i.e.*, 25K). We firstly retrain the single backbone, which is a common strategy in most deep networks (*e.g.*, [17]). The results show that it can hardly predict correct class of **CUS**, leading bad performance on these tasks (Fig. 7 (a)). We then freeze the single backbone pre-trained on COCO during the editing data training. The performance is improved due to relieving the over-fitting problem. However, the frozen backbone can not extract adequate features (Fig. 7 (b)). On the contrast, our two backbones which pre-trained on car detection task and general task can not only extract adequate features but also avoid over-fitting problem. It achieves the best performance on these tasks (Fig. 7 (c)).

The impact of the number of training data. Empirically, the performance of deep network largely relies on the number of training data. Here, we conduct an experiment to study the relationship between the number of data and network performance. Thanks to the fully automatic editing-based approach, we set the number of data from 5K to 40K with an interval of 5K to train our network. Fig. 8 shows the network performance on multiple tasks with respect to

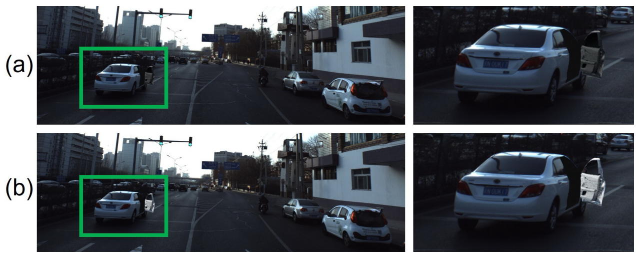

Figure 9: The rendering results with (a) and without (b) environment map.

Tasks	w/o Env. Map	with Env. Map
2D Detection (*mAP*)	0.688	**0.701**
Ins. Seg. (*mAP*)	0.538	**0.563**
Part Seg. (*mAP*)	0.221	**0.314**
State Description	0.844	**0.874**

Table 4: The impact of environment map.

different number of training data. We find that from 5K to 25K, the network performance is significantly improved. While from 25K to 40K, it is not sensitive to the number. In practice, we set the number of training data to 25K, which is a good compromise of efficiency and accuracy.

The impact of environment map. In the proposed data generation pipeline, we render the reverse side of dynamic parts to generate the invisible region data. While in the wild, illumination (or environment map) plays an important role which determines the rendered region whether is compatible with the surroundings. Here, we conduct an experiment to study the effectiveness of environment map. We utilize the same number of reverse side data with/without environment map (shown in Fig. 9) to train our network, and evaluate on the proposed CUS dataset. As shown is Tab. 4, the data rendered by environment map significantly improves the network performance. In particular, the dynamic part segmentation gets a **9.3** percent improvement.

6.5. Application

Our results can effectively support a number of high-level vision tasks. As shown in Fig. 10, we integrate human detection task to our network. Intuitively, there exists rich semantics between the human and the dynamic parts. For example, if someone stands near by the lifted trunk, it is very likely that he/she is taking the luggage. If someone bends to push the door of car, it implies he/she would get off. Besides action reasoning and interaction understanding, we can even infer the people's identity from the uncommon cases. For instance, if the front left door is opened, people near the door usually is a driver.

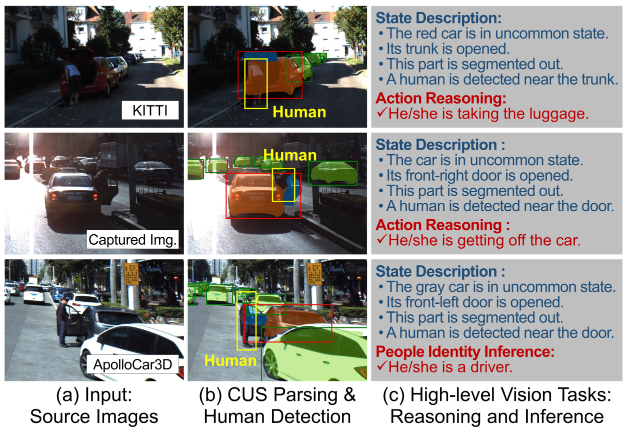
KITTI
Human
State Description:
• The red car is in uncommon state.
• Its trunk is opened.
• This part is segmented out.
• A human is detected near the trunk.
Action Reasoning:
✓He/she is taking the luggage.
Captured Img.
Human
State Description :
• The car is in uncommon state.
• Its front-right door is opened.
• This part is segmented out.
• A human is detected near the door.
Action Reasoning :
✓He/she is getting off the car.
ApolloCar3D
Human
State Description :
• The gray car is in uncommon state.
• Its front-left door is opened.
• This part is segmented out.
• A human is detected near the door.
People Identity Inference:
✓He/she is a driver.
(a) Input: Source Images
(b) CUS Parsing & Human Detection
(c) High-level Vision Tasks: Reasoning and Inference

Figure 10: The applications on action reasoning and people identity inference by understanding **CUS**.

7. Conclusion and Limitation

In this paper, we make the first attempt to analyse the cars in uncommon states (**CUS**). Instead of annotating large amount of images, we present an editing-based data generation approach which takes advantage of 3D parts. Our method is light-weight but high-efficiency, which advances the rendering-based methods by a large margin. To perform a holistic understanding for CUS, we propose a multi-task deep network which can simultaneously output 2D detection, instance-level segmentation, dynamic part segmentation, and state description. To benchmark the performance, we construct a CUS dataset which contains 1441 real images (1850 car instances) with fine-grained annotation. The experimental results show that our editing data and deep network perform well on CUS.

Nevertheless, there are a number of limitations, which point out our direction in the future work. First, AD is a huge and complex project, the uncommon states analysed in this paper are closed to cars. Some other objects, such as human and road, we will pay more attention to them. Second, the output of our network are mostly 2D results. We will extend this work to 3D space, such as 3D detection, 3D localization, and 3D reconstruction. Third, we will research CUS on the video sequences. Lastly, we will fuse multiple sensors (*e.g.*, RGB camera, stereo camera, Lidar, and Radar) to research the CUS problem.

Acknowledgement

We thank the anonymous reviewers for their valuable comments. This work was supported in part by National Natural Science Foundation of China (U1736217 and 61932003), National Key R&D Program of China (2019YFF0302902), and Pre-research Project of the Manned Space Flight (060601).

References

[1] Hassan Abu Alhaija, Siva Karthik Mustikovela, Lars Mescheder, Andreas Geiger, and Carsten Rother. Augmented reality meets computer vision.

[2] Gabriel J Brostow, Julien Fauqueur, and Roberto Cipolla. Semantic object classes in video: A high-definition ground truth database. *Pattern Recognition Letters*, 30(2):88–97, 2009.

[3] Daniel J Butler, Jonas Wulff, Garrett B Stanley, and Michael J Black. A naturalistic open source movie for optical flow evaluation. In *Proceedings of ECCV*, pages 611–625. Springer, 2012.

[4] F. Chabot, M. Chaouch, J. Rabarisoa, C. Teulière, and T. Chateau. Deep manta: A coarse-to-fine many-task network for joint 2d and 3d vehicle analysis from monocular image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[5] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], 2015.

[6] 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 *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3213–3223, 2016.

[7] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. IEEE, 2009.

[9] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick Van Der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2758–2766, 2015.

[10] Debidatta Dwibedi, Ishan Misra, and Martial Hebert. Cut, paste and learn: Surprisingly easy synthesis for instance detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1301–1310, 2017.

[11] Francis Engelmann, Theodora Kontogianni, Alexander Hermans, and Bastian Leibe. Exploring spatial context for 3d semantic segmentation of point clouds. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 716–724, 2017.

[12] Cheng-Yang Fu, Wei Liu, Ananth Ranga, Ambrish Tyagi, and Alexander C Berg. Dssd: Deconvolutional single shot detector. *arXiv preprint arXiv:1701.06659*, 2017.

[13] Adrien Gaidon, Qiao Wang, Yohann Cabon, and Eleonora Vig. Virtual worlds as proxy for multi-object tracking analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4340–4349, 2016.

[14] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

[15] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A Wichmann, and Wieland Brendel. Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness. *arXiv preprint arXiv:1811.12231*, 2018.

[16] Ankur Handa, Viorica Patraucean, Vijay Badrinarayanan, Simon Stent, and Roberto Cipolla. Understanding real world indoor scenes with synthetic data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4077–4085, 2016.

[17] 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.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[19] Stefan Hinterstoisser, Vincent Lepetit, Paul Wohlhart, and Kurt Konolige. On pre-trained image features and synthetic images for deep learning. In *The European Conference on Computer Vision (ECCV) Workshops*, September 2018.

[20] Xinyu Huang, Xinjing Cheng, Qichuan Geng, Binbin Cao, Dingfu Zhou, Peng Wang, Yuanqing Lin, and Ruigang Yang. The apolloscape dataset for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 954–960, 2018.

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[22] Abhijit Kundu, Yin Li, and James M. Rehg. 3d-rcnn: Instance-level 3d object reconstruction via render-and-compare. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[23] Jean-François Lalonde and Alexei A Efros. Synthesizing environment maps from a single image. *Technical Report CMU-R I-TR-10–24*, 2010.

[24] Huan Lei, Naveed Akhtar, and Ajmal Mian. Spherical kernel for efficient graph convolution on 3d point clouds. *arXiv preprint arXiv:1909.09287*, 2019.

[25] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2117–2125, 2017.

[26] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Proceedings of ECCV*, pages 740–755. Springer, 2014.

[27] Cewu Lu, Hao Su, Yonglu Li, Yongyi Lu, Li Yi, Chi-Keung Tang, and Leonidas J Guibas. Beyond holistic object recognition: Enriching image understanding with part states. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6955–6963, 2018.

[28] Nikolaus Mayer, Eddy Ilg, Philip Hausser, Philipp Fischer, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4040–4048, 2016.

[29] Matthias Müller, Vincent Casser, Jean Lahoud, Neil Smith, and Bernard Ghanem. Sim4cv: A photo-realistic simulator for computer vision applications. *International Journal of Computer Vision*, 126(9):902–919, 2018.

[30] Gerhard Neuhold, Tobias Ollmann, Samuel Rota Bulo, and Peter Kontschieder. The mapillary vistas dataset for semantic understanding of street scenes. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4990–4999, 2017.

[31] Aayush Prakash, Shaad Boochoon, Mark Brophy, David Acuna, Eric Cameracci, Gavriel State, Omer Shapira, and Stan Birchfield. Structured domain randomization: Bridging the reality gap by context-aware synthetic data. *arXiv preprint arXiv:1810.10093*, 2018.

[32] Weichao Qiu and Alan Yuille. Unrealcv: Connecting computer vision to unreal engine. In *Proceedings of ECCV*, pages 909–916. Springer, 2016.

[33] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[34] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*, pages 91–99, 2015.

[35] Stephan R Richter, Zeeshan Hayder, and Vladlen Koltun. Playing for benchmarks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2213–2222, 2017.

[36] Stephan R Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. In *Proceedings of ECCV*, pages 102–118. Springer, 2016.

[37] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio M Lopez. The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3234–3243, 2016.

[38] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[39] Xibin Song, Peng Wang, Dingfu Zhou, Rui Zhu, Chenye Guan, Yuchao Dai, Hao Su, Hongdong Li, and Ruigang Yang. Apollocar3d: A large 3d car instance understanding benchmark for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5452–5462, 2019.

[40] Hao Su, Charles R Qi, Yangyan Li, and Leonidas J Guibas. Render for cnn: Viewpoint estimation in images using cnns trained with rendered 3d model views. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2686–2694, 2015.

[41] Robert W Sumner, Johannes Schmid, and Mark Pauly. Embedded deformation for shape manipulation. *ACM Transactions on Graphics (TOG)*, 26(3):80, 2007.

[42] Supasorn Suwajanakorn, Noah Snavely, Jonathan J Tompson, and Mohammad Norouzi. Discovery of latent 3d keypoints via end-to-end geometric reasoning. In *Advances in Neural Information Processing Systems*, pages 2059–2070, 2018.

[43] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30. IEEE, 2017.

[44] Carlo Tomasi and Roberto Manduchi. Bilateral filtering for gray and color images. In *Proceedings of ICCV*, volume 98, page 2, 1998.

[45] Jonathan Tremblay, Aayush Prakash, David Acuna, Mark Brophy, Varun Jampani, Cem Anil, Thang To, Eric Cameracci, Shaad Boochoon, and Stan Birchfield. Training deep networks with synthetic data: Bridging the reality gap by domain randomization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 969–977, 2018.

[46] Peng Wang, Xinyu Huang, Xinjing Cheng, Dingfu Zhou, Qichuan Geng, and Ruigang Yang. The apolloscape open dataset for autonomous driving and its application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[47] Peng Wang, Xiaohui Shen, Zhe Lin, Scott Cohen, Brian Price, and Alan L Yuille. Joint object and part segmentation using deep learned potentials. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1573–1581, 2015.

[48] Shenlong Wang, Min Bai, Gellert Mattyus, Hang Chu, Wenjie Luo, Bin Yang, Justin Liang, Joel Cheverie, Sanja Fidler, and Raquel Urtasun. Torontocity: Seeing the world with a million eyes. *arXiv preprint arXiv:1612.00423*, 2016.

[49] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1912–1920, 2015.

[50] Fangting Xia, Peng Wang, Liang-Chieh Chen, and Alan L Yuille. Zoom better to see clearer: Human and object parsing with hierarchical auto-zoom net. In *Proceedings of ECCV*, pages 648–663. Springer, 2016.

[51] Fisher Yu, Wenqi Xian, Yingying Chen, Fangchen Liu, Mike Liao, Vashisht Madhavan, and Trevor Darrell. Bdd100k: A diverse driving video database with scalable annotation tooling. *arXiv preprint arXiv:1805.04687*, 2018.

[52] Yi Zhang, Weichao Qiu, Qi Chen, Xiaolin Hu, and Alan Yuille. Unrealstereo: A synthetic dataset for analyzing stereo vision. *arXiv preprint arXiv:1612.04647*, 1(2), 2016.