CrashCar101: Procedural Generation for Damage Assessment

Jens Parslov[*,1] Erik Riise[*,1] Dim P. Papadopoulos[1,2]
[1] Technical University of Denmark [2] Pioneer Center for AI
jens@parslov.com, erikriise@live.no, dimp@dtu.dk
https://crashcar.compute.dtu.dk

Abstract

In this paper, we are interested in addressing the problem of damage assessment for vehicles, such as cars. This task requires not only detecting the location and the extent of the damage but also identifying the damaged part. To train a computer vision system for the semantic part and damage segmentation in images, we need to manually annotate images with costly pixel annotations for both part categories and damage types. To overcome this need, we propose to use synthetic data to train these models. Synthetic data can provide samples with high variability, pixel-accurate annotations, and arbitrarily large training sets without any human intervention. We propose a procedural generation pipeline that damages 3D car models and we obtain synthetic 2D images of damaged cars paired with pixel-accurate annotations for part and damage categories. To validate our idea, we execute our pipeline and render our CrashCar101 dataset. We run experiments on three real datasets for the tasks of part and damage segmentation. For part segmentation, we show that the segmentation models trained on a combination of real data and our synthetic data outperform all models trained only on real data. For damage segmentation, we show the sim2real transfer ability of CrashCar101.

1. Introduction

Damage assessment is the task of determining the extent of damage resulting from an accident or a disaster. Vehicle damage is a common type that occurs from transportation and road accidents [42, 64, 65]. Damage assessment is essential for response organizations, insurance businesses, and rental agents. This is usually performed by experts who manually check vehicles on-site and evaluate their damages.

Recently, there have been several attempts to build automatic damage assessment systems with computer vision models [42, 63–65, 71]. However, automatically assessing damages on vehicles is a challenging task. It requires not only detecting and localizing the specific damages on the vehicles but also accessing the extent of the damage depending on the part of the vehicle which is damaged. From a computer vision perspective, this means going beyond the image-level classification problem and training a semantic segmentation model able to produce per-pixel category predictions for both damage types and semantic parts in new test images of vehicles. Training such a model requires huge amounts of annotated images where humans draw detailed outlines around every damage and part that appears in an image [9, 17, 31, 47, 72]. This process is expensive and prone to several erroneous labels especially when the annotation task is challenging [38, 39, 49, 53].

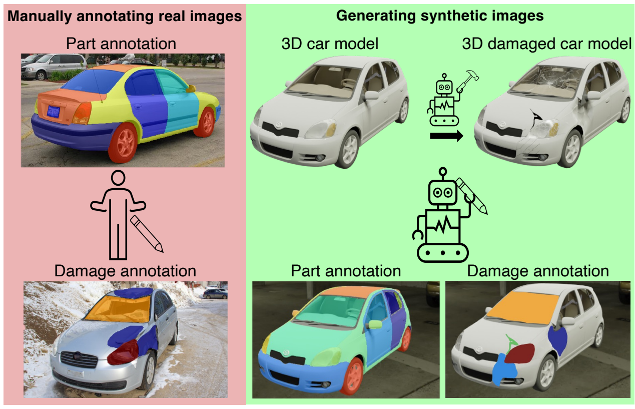


Figure 1. **Learning part and damage segmentation from synthetic data.** (Left) The standard approach of manually annotating real car images with pixel annotations. (Right) We propose to use synthetic 3D car models, destroy them using a procedural damage generation pipeline, and obtain realistic 2D images that come with pixel-accurate annotations for semantic parts and damages.

In this paper, we propose to overcome these issues by automatically generating realistic synthetic data to train a damage assessment system (Fig. 1). Realistic synthetic images have been used successfully in several semantic segmentation tasks [16, 26, 33, 46, 50]. Synthetic data can pro-

[*] Denotes equal contribution

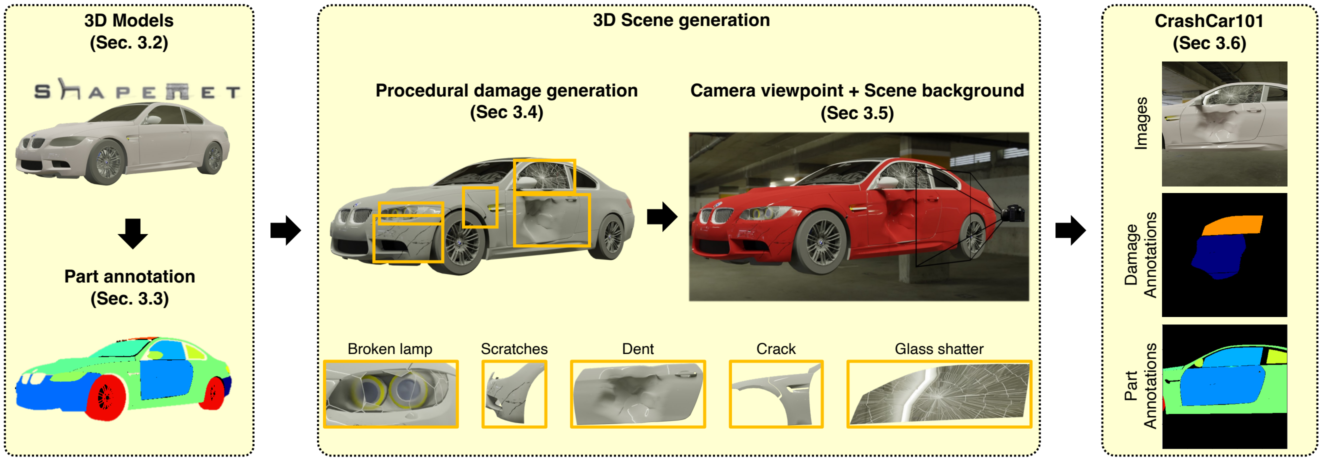


Figure 2. **Overview of our proposed procedural generation pipeline.** We first acquire and annotate 3D model cars (Sec. 3.2 and 3.3), and then we apply procedural generation to manipulate the texture and shape of the car to generate synthetic damage types (Sec. 3.4. Subsequently, we place a camera into the scene and assign a scene environment and car color (Sec. 3.5). Finally, we render a 2D image paired with perfect ground-truth pixel annotations for parts and damages. Given a set of 3D model cars with part annotations, this process is fully automatic, allowing us to render an arbitrarily large amount of data (CrashCar101 dataset, Sec. 3.6).

vide samples with high variability and perfect automatically generated pixel annotations. When they are generated via a procedural generation pipeline, they can lead to arbitrarily large sets of training data without any human intervention [10, 16, 22, 27]. We propose a novel procedural damage generation pipeline that creates realistic damages of various types on 3D car models [4] (Sec. 3). We focus only on cars as the most common vehicle type, but a similar procedure can be followed for other objects. We create damages by manipulating either the 3D mesh geometry (*dents*) or the physically-based material of the models (*scratches, cracks, shattered glass, broken lamps*). By controlling the model parameters, we can create damages at different scales, positions, shapes, and appearance variations.

Our procedural generation pipeline is shown in Fig. 2. We start with a 3D car model [4] where we annotate submeshes with semantic part labels. We apply our damage generation pipeline to obtain damages into the 3D car model. Then, we place the generated model into an urban HDRI scene that provides realistic lighting and background noise. Finally, we set the camera parameters in order to obtain a 2D image. The final image is paired with two generated segmentation maps with pixel-accurate labels, one for the car semantic parts and one for the damage types.

To validate our idea, we first annotate 99 car models from ShapeNetCore [4] with part annotations and then we execute our generation pipeline to obtain our CrashCar101 dataset that consists of 101,050 images paired with perfect part and damage segmentation. We conduct experiments on three real test datasets to show the usefulness of our synthetic dataset. For damage segmentation, our results in a few-shot learning scenario show that pre-training on CrashCar101 yields significantly better results (+6.3-17.9% mIoU at 1-shot and +4.4-7.0% at 5-shot) compared to using a pre-trained model on COCO [31] or ImageNet [11]. For part segmentation, we show that the segmentation models trained on a combination of real data and our synthetic data outperform all models trained only on real data.

2. Related work

Synthetic data has been a valuable asset for addressing several problems such as semantic segmentation for urban landscapes [3, 45, 46], object recognition [19, 43, 59], face-related tasks [66] and medical imaging [15]. Procedural generation is an important field as it gives the ability to generate millions of 3D scenes without any human intervention [37]. Procedural generation is widespread in video games [21, 52, 58], but it has also been used for synthetic data generation [23, 26, 44]. For example, in [26] cities and outdoor scenes are generated for semantic segmentation. In [44], a stochastic scene grammar from an indoor dataset is learned to generate new scene layouts. In [23], entire humans are generated and deep networks are fitted on the resulting images to regress a dense set of annotations.

Semantic part segmentation is the task of segmenting fine-grained parts within a target object such as cars [8, 33, 40], birds [48, 61], humans [8] or buildings [62]. Obtaining annotations for semantic parts is expensive and challenging since there is often no clear boundary to separate the object parts. For these reasons, the research has been focused on ways to reduce this manual annotation cost by training weakly supervised models [73], unsupervised models [24] or domain adaptation approaches on synthetic data [33].

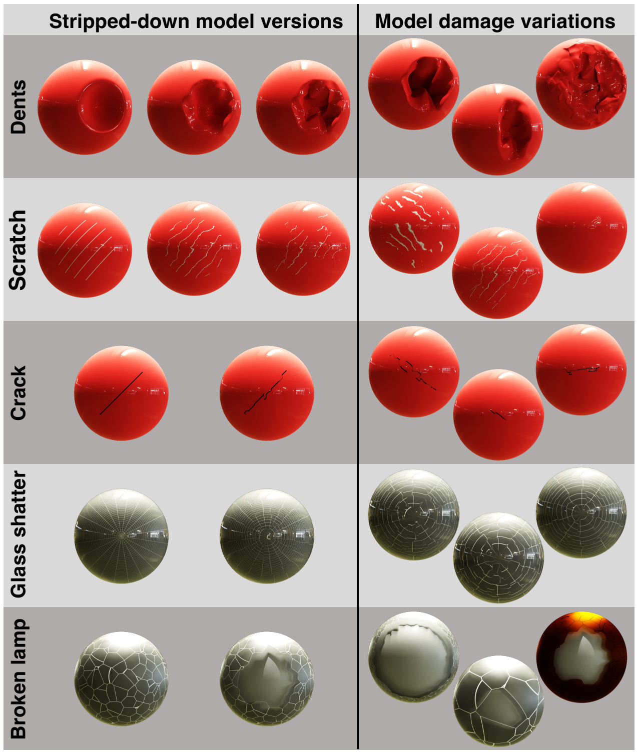


Figure 3. **Damage models on toy 3D spheres**. We show stripped-down versions of each model (left) and examples of how the different values of the model parameters affect the appearance and texture of the damage (right). The texture of the spheres resembles the texture of the corresponding car parts (dents, scratches and cracks on the body parts, glass shatters on the windows, and broken lamp on headlights.

No large-scale dataset for part segmentation on cars exists. Pascal-Part [8] includes 613 images with 13 part categories. CGPART [32] has 31 fine-grained part categories but only 40 images. In Sec. 4.1, we conduct experiments in both datasets and we show performance improvements for the task of part segmentation when using our synthetic data.

Damage detection is an essential problem for emergency response especially in cases of natural disasters, destructive events, or accidents [18, 35, 42, 64, 65]. Several papers have been published more closely related to our task, as they delve into the task of classifying damage in cars [2, 12, 13, 29, 41, 42, 56, 60, 63, 70]. However, with the exception of [63], they all focus on image classification tasks. To the best of our knowledge, there are only three publicly available datasets with labeled car damage [25, 34, 63]. [25] contains only image-level labels, while [34] contains only 80 images where all damage types are annotated as one category. The CarDD dataset [63], which was recently released, contains 4,000 images with pixel annotations for six damage types. In Sec. 4.2, we run experiments in CarDD showing the usefulness of CrashCar101 for the task of damage segmentation on real images.

Shape 3D models. The ShapeNet dataset [4] is a large collection of 3D object models. ShapeNetCore [51] is a subset of ShapeNet with models of 55 manually verified object categories, including cars. ShapeNetCore has been used in several computer vision applications such as aligning CAD models [1], predicting object-level intrinsics [55] or generating shapes from natural language [5]. In [36], PartNet extends a subset of ShapeNetCore [4] with fine-grained 3D part segmentation over 24 object categories. However, none of these categories include vehicles. In this paper, we annotate and provide fine-grained part segmentation for 27 semantic parts on 99 car models from ShapeNetCore [4] (Sec. 3.3).

3. Procedural damage generation

In this section, we explain each step of our procedural synthetic data generation pipeline (Fig. 2). All rendering, texturing and 3D mesh manipulation is conducted in the open-source software application Blender [1].

3.1. Overview

Fig. 2 presents an overview of our procedural generation pipeline. We first acquire a diverse set of 3D car models from the ShapeNetCore dataset [4] (Sec. 3.2) and we manually label the sub-meshes of these models with fine-grained part categories (Sec. 3.3). Then, we apply our proposed method for procedural damage generation which is presented in Sec. 3.4. After the damages are placed in the 3D car, we set the scene environment, and the car color and we place the camera into the scene leading to 2D images paired with pixel-accurate annotations for part categories and damage types (Sec. 3.5). We execute this pipeline and we obtain and render the CrashCar101 dataset which consists of 101,050 images (Sec. 3.6).

3.2. 3D models

We use 3D vehicle models from ShapeNetCore [4]. Even though the PartNet extends a subset of ShapeNetCore [4] with 3D part segmentation over 24 object categories, none of these categories include vehicles. As a result, in this paper, we manually annotate the sub-meshes of 99 selected 3D car models from ShapeNetCore with fine-grained part categories. We choose to use the part taxonomy from [33] which consists of 31 part categories. We further combine all four wheel categories into one and the two license plate categories into one. This results in 27 part categories.

3.3. Part annotation

Manually annotating every sub-mesh of every car model and mapping it to a part category is expensive. To make

[1] https://www.blender.org/

this process more efficient, we follow a human-in-the-loop approach. We start by manually annotating nine models from ShapeNetCore. Then, we obtain eight images for each annotated model from different viewpoints around the car and we use them to train a DeepLabv3 segmentation model [6, 20]. For each new car model from ShapeNetCore, we render the same eight viewpoints and predict the semantic part segmentation using the trained model. The meshes of the 3D cars are compared with the predictions using mIoU computed across the 8 images. For each part class, we rank the meshes using mIoU from least to most likely. In order to utilize the ranking, we created an interactive labeling tool and captured eight views of each car. By employing the mIoU metric, the tool displays the eight most likely meshes belonging to each part. A single car model in ShapeNetCore can have thousands of sub-meshes [4], making manual labeling expensive and tedious. According to our time recordings, this human-in-the-loop interactive approach was about 3× faster than the fully manual one.

3.4. Procedural Generation for Synthetic Damage

The goal of our procedural generation pipeline is to create realistic damages on 3D car models. We consider 5 damage types: *dents*, *scratches*, *cracks*, *broken lamps*, and *glass shatters*, which reflect the most common damage types [63]. In this section, we present damage generators that manipulate the shape and texture of the car. These generators apply procedural rules that model damages in generality. Each damage generator has interpretable input parameters that control the damage and can be sampled from pre-defined distributions to generate random variations of damages and lead to a final dataset with high variability.

To better understand the damage generators, we apply them on toy 3D spheres in Fig. 3. We show stripped-down versions of each damage model (Fig. 3 (left)) and how different input parameters affect the appearance and texture of the damage (Fig. 3 (right)). In the following ϵ_w, ϵ_{v_c}, ϵ_{v_d} and ϵ_p refer to Wave, Voronoi color, Voronoi distance and Perlin noise implementations of noise generators in Blender.

Dents are generated using a function $f_D(x) : \mathbb{R}^3 \to \mathbb{R}^3$ that maps the vertex coordinates of an undamaged car to perturbed coordinates that depict a damaged car. The dent generator is implemented using Blender Geometry Nodes. The function $f_D(\mathbf{x})$ consists of two components: $l(\mathbf{x})$ and $d(\mathbf{x})$. $l(\mathbf{x}) \in \mathbb{R}$ defines the length of the displacement vector, and $d(\mathbf{x}) \in \mathbb{R}^3$ defines the direction. The direction component $d(\mathbf{x})$ is defined as $\mathbf{n} + \epsilon_{v_c}(\mathbf{x})$, where $\mathbf{n}$ is the normal vector and $\epsilon_{v_c} : \mathbb{R}^3 \to \mathbb{R}^3$ is a noise generator that simulates a crumpling effect by adding noise to the displacement direction. The effect is visualized in the second and third spheres in Fig. 3 (first row). The length component $l(\mathbf{x})$ is defined as $d\frac{\cos\left(\pi\frac{a-||\mathbf{x}-\mathbf{c}+\epsilon_p(\mathbf{x})||}{a}\right)+1}{2}$, where $\mathbf{c}$ is the center coordinate of the dent, a is the area of the dent, d is the depth and $\epsilon_p : \mathbb{R}^3 \to \mathbb{R}$ is a noise generator which adds noise to the magnitude of displacement as illustrated on the first and second sphere in Fig. 3 (first row). The cosine function produces a gradual smooth transition into the dent. A car paint shader S_p is applied to texture the car's body, while a glass shader S_g is used for the windows. The integration of the remaining damage types is achieved through Blender's Mix Shader node, blending two input shaders based on a probability factor. To facilitate this blending process, functions $f : \mathbb{R}^3 \to [0, 1]$ are defined, mapping coordinates on the car's surface to a probability. This probability determines the mixture of shaders, combining the undamaged car shaders S_p and S_g with shaders containing damage effects. Let $m(p, S_1, S_2)$ denote the Mix shader in Blender where shader S_1 and S_2 are being mixed with probability p.

Scratches are generated by overlaying S_p with a scratch shader S_d using generated scratch marks. To generate scratch intensities we define

$$f_r(\mathbf{x}) = [a \geq ||\mathbf{x} - \mathbf{c} + \epsilon_{v_c}(\mathbf{x})||]$$
$$S_{scratch} = m(f_r(\mathbf{x})\epsilon_w(\mathbf{x}), S_d, S_p)$$

where [P] denotes the Iverson bracket notation and it equals 1 if the statement P is true and 0 otherwise. $f_r(\mathbf{x})$ is used to select the scratched region of the car, $\mathbf{c}$ determines the center, a determines the area, and $\epsilon_{v_c}(\mathbf{x}) \in \mathbb{R}^3$ adds noise as seen on the first and third sphere in Fig. 3 (second row). ϵ_w produces straight thin lines, which are manipulated using a distortion property as depicted on the first and second sphere in Fig. 3 (second row).

Cracks are generated by mixing S_p with a transparent shader S_α using generated crack masks. The crack mask is a line with a combination of smooth and sharp deviations. Let x_1 and x_2 be the first and second coordinates of $\mathbf{x}$. We define $\hat{x}_1 = x_1 + \epsilon_{v_{c_1}}$, $\hat{x}_2 = x_2 + \epsilon_{v_{c_2}}$ as noisy coordinates which produce deviation as shown on the first two spheres of Fig. 3 (third row). We propose $f_l(\mathbf{x}) = 1 - \max(1 - |\hat{x}_1 - \hat{x}_2|, 0)$ to produce a line and $f_r(\mathbf{x}) = \max(0, a - ||\mathbf{c} - \mathbf{x}||)$ to select the crack region.

$$S_{crack} = m(f_l(\mathbf{x})f_r(\mathbf{x}), S_\alpha, S_p)$$

Glass shatter are generated by combining two shaders: a glass shader S_g and a white non-transparent one S_w. Glass often shatters with lines radiating from the place of incident, alongside concentric spread rings. Once again ϵ_{v_c} is used to produce sharp deviations (see the first two spheres of Fig. 3(fourth row)). To produce concentric rings we use $f_c(\mathbf{x}) = \sin(||s(\mathbf{x} - \mathbf{c} + \epsilon_{v_c}(\mathbf{x}))||) < t$ where t defines the thickness, and s defines the scale. For the radial lines, we use the radial gradient texture in Blender ϵ_g and use $f_r(\mathbf{x}) = \epsilon_{v_d}(\epsilon_g(\mathbf{x}))\,||\mathbf{x}|| < t$, here ϵ_d is a noisy periodic function, which we use to determine the number of generated lines and adding noise to the distances between the

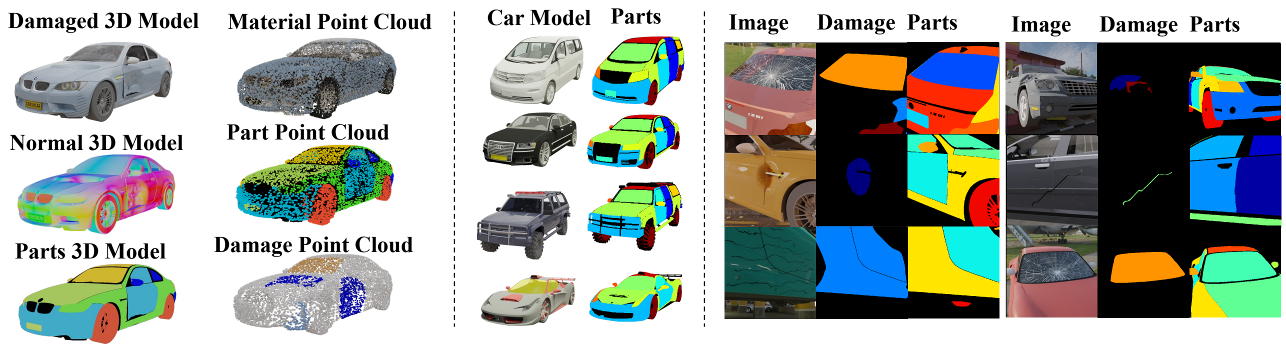


Figure 4. **CrashCar101 Synthetic Dataset.** (Left) Depicts a 3D damaged car model, including its normals and part labels, from which we extract point cloud representations of material, parts, and damage. (Middle) Displays a subset of car models with labeled parts, while (right) showcases diverse 2D images generated from the 3D models.

lines. The shatter shader is defined as

$$S_{shatter} = m([f_c(\mathbf{x}) \; or \; f_r(\mathbf{x})], S_w, S_g)$$

Broken lamps are generated by making a fractured shader S_f. To make S_f we mix a glass shader S_g with a white shader S_w. $S_f = m([\epsilon_{v_d}(\mathbf{x}) < t], S_w, S_g)$ where t determines the thickness of the fractures, an example of S_f can be seen at the first sphere of the last row in Fig. 3. A chunk is removed by mixing in a transparent shader S_α as depicted in Fig. 3 (second sphere, last row), to produce the final broken lamp shader S_{broken}.

$$S_{broken} = m([||\epsilon_{v_p}(\mathbf{x})|| < a], S_\alpha, S_f)$$

Position of damage center. To select the center of the damage $\mathbf{c}$, we follow the following procedure: We first select randomly one main damage among the 5 damage types. Then, one of the car parts that can contain this damage is selected with equal probability, and a random vertex with coordinates $\mathbf{c}_{main}$ from the part is selected. We set the parameter $\mathbf{c} = \mathbf{c}_{main}$ for the main selected type. All the other parameters are sampled from a uniform distribution. The minimum and maximum values are manually selected to be the most extreme values seen in realistic cases.

3.5. Camera viewpoint and scene background

The following section describes how the 3D scene is randomized to generate realistic synthetic 2D images. First, we describe how the camera is placed, such that the damage is visible and then, we describe how the car paint and background are randomized to generate realistic 2D images.

Viewpoint randomisation is done by randomising the camera position $\mathbf{v}$ and the camera rotation $\boldsymbol{\theta}$. Having placed the damage at some point $\mathbf{c}_{main}$ we now place the camera position at $\mathbf{c}_{main}$ and then translating in the direction of $R\left(\hat{n}_{\text{yaw}}, \theta_{yaw}\right) R\left(\hat{n}_{\text{pitch}}, \theta_{\text{pitch}}\right) \frac{v}{\|v\|}$ by some distance d. Where $R\left(\hat{n}, \theta\right)$ is the rotation matrix when $\hat{n}$ is the rotation axis and θ is the rotation angle. Note that we wish to control the pitch and yaw of the vector independently thus we select $\hat{n}_{\text{yaw}} = (0, 0, 1)$ and $\hat{n}_{\text{pitch}} = (-c_{main_2}, c_{main_1}, 0)$. Finally we obtain the camera coordinates $\mathbf{v}$ as:

$$\mathbf{v} = \mathbf{c}_{main} + R\left(\hat{n}_{\text{yaw}}, \theta_{yaw}\right) R\left(\hat{n}_{\text{pitch}}, \theta_{\text{pitch}}\right) \frac{\mathbf{c}_{main}}{\|\mathbf{c}_{main}\|} \cdot d$$

Now that the camera is placed, we select $\boldsymbol{\theta}$ such that $\mathbf{c}_{main}$ is perfectly centered in the frame. We now randomly select θ_1 and θ_2 in such a way that the main damage is jittered with respect to the raster coordinates. Now that the primary damage is determined we apply secondary damage. To ensure that the secondary damage is visible all vertices of the parts that can contain damage are transformed to raster coordinates, and only those that are contained in the frame are kept. Only the vertices within a certain distance to the camera are kept. Finally, a random vertex of one of the visible objects is selected. This second damage is applied with a probability of 0.5, thereafter 0.2 until no damage is applied.

Background randomization After completing the annotation step, the scenes for which the vehicles are to be placed are initialized. We collected a total of 338 urban scene 4K HDRIs from Polyhaven [2]. The HDRI provides realistic lighting and background noise. Further, to add variation, we sampled realistic vehicle colors from GTA V [3].

3.6. CrashCar101 dataset

We execute our procedural generation pipeline and we render the CrashCar101 dataset. CrashCar101 consists of 101,050 2D images paired with annotated damage and part segmentation. Fig. 4 shows examples of CrashCar101. Our procedural generators damage the car, from which we can extract 3D and 2D modalities. From this, we produce CrashCar101, a 2D image dataset containing both part and damage segmentations. A subset of our dataset does not

[2] https://polyhaven.com/hdris/urban

[3] https://wiki.rage.mp/index.php?title=Vehicle_Colors&oldid=21033

Dataset	Train	Val	Test	Part	Dmg
Pascal-Part [8]	490	61	62	✓	
UDAPART [33]	-	-	40	✓	
CGPART [32]	31,448	7,867	-	✓	
CrashCar101-Part	**14,175**	**1,575**	**1,575**	✓	
CarDD [63]	2,638	768	349		✓
CrashCar101	**83,604**	**8,311**	**9,135**	✓	✓

Table 1. **Datasets used in our experiments.** The top part of the table shows the datasets for the part segmentation experiments while the bottom part shows the ones for damage segmentation. The last two columns "Part" and "Dmg" indicate the existence of part and damage annotations in the dataset.

contain any damage, we refer to this subset as CrashCar101-Part. It consists of 17,325 images (175 images per car model) and we use it to train the part segmentation models in Sec. 4.1. We focus on 2D images, but generating 3D modalities is a feature available in the synthetic data generation pipeline. This is intended to enable further studies into the applicability of synthetic data in 3D research.

Regarding the damage categories, we show interesting statistics of our obtained dataset (Fig. 5). In Fig. 5a, we show the distribution of each damage size in terms of the percentage of pixels they occupy in each image. As expected, we observe that cracks are usually tiny, while glass shatter damages usually occupy much larger image parts. The distribution of damage occurrence on each part is presented in Fig. 5c showing a good balance between damage types. In Fig. 5b, we show the number of images containing each separate damage in CrashCar101. Note that damages can only occur on specific parts (e.g., dents, cracks, and scratches on metallic parts, glass shatters on window glasses, and broken lamps on head and tail lights).

4. Experimental results

This section presents our experimental results. We evaluate the usefulness of our CrashCar101 dataset on two tasks: semantic part segmentation (Sec. 4.1) and damage segmentation (Sec. 4.2). For each task, we compare the segmentation models trained on our dataset to models trained on real images and on combinations of real and synthetic data.

Implementation details. Unless stated otherwise, we use the following settings for all models of our two tasks. All models are trained with the focal loss function [30]. The focal loss is a modification of the standard cross-entropy loss that assigns higher weights to hard-to-classify examples, leading to improved performance on imbalanced datasets. For damage segmentation, we use 6 output dimensions (5 damage types plus the background category). For the part segmentation, we use either 28 or 12 output dimensions (including background) depending on which real dataset we evaluate the part segmentation models (see Sec. 4.1 for more details). The input images are resized to 256×256 for the task of part segmentation and 384×384 for the task of damage segmentation. We use a batch size of 64 for all models. For a fair comparison among all trained models, we perform the same set of augmentations while training. These are limited to random resize cropping, random rotation, and color jitter. For part segmentation models, we also train models without augmentations to evaluate the effect of augmentations on our data compared to real data. We train all models for 20 epochs using the Adam optimizer [28] with an initial learning rate of 0.0002. The only exception is the model trained on Pascal-Part, which due to the small size of the dataset was trained for 40 epochs. We perform early stopping based on the performance in the hold-out validation set. Note that each different training set has its corresponding validation set from the same domain. All experiments were run on a single Nvidia V100 GPU.

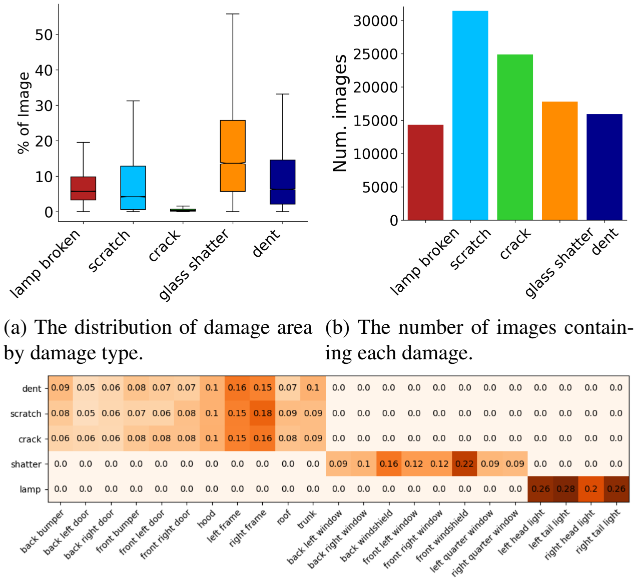
Evaluation. We use the mean intersection-over-union (mIoU) as our main evaluation metric for both tasks, as this is standard for evaluating any image segmentation task.

	Dataset	Augs	UDAPART		PASCAL-Part
	Parts		11	27	11
DeepLabv3 (RN50)	Pascal-Part [8]	✗	42.1	-	36.8
		✓	37.1	-	36.0
	CGPART [32]	✗	17.5	15.8	4.8
		✓	34.3	25.7	14.4
	CrashCar101-Part	✗	39.3	**42.8**	19.2
		✓	43.0	41.5	22.4
	CrashCar101-Part + Pascal-Part [8]	✗	43.1	-	36.0
		✓	**52.5**	-	**42.9**
DeepLabv3 (RN101)	Pascal-Part [8]	✗	40.9	-	36.8
		✓	40.0	-	37.3
	CGPART [32]	✗	17.8	15.6	5.6
		✓	38.1	30.1	13.5
	CrashCar101-Part	✗	40.4	**47.6**	20.0
		✓	50.1	47.2	26.4
	CrashCar101-Part + Pascal-Part [8]	✗	45.3	-	39.2
		✓	**52.3**	-	**44.2**
Segformer (b5)	Pascal-Part [8]	✗	40.0	-	37.1
		✓	38.0	-	36.7
	CGPART [32]	✗	27.6	19.2	8.0
		✓	52.4	52.7	24.8
	CrashCar101-Part	✗	46.4	48.8	26.0
		✓	**56.3**	**61.6**	31.1
	CrashCar101-Part + Pascal-Part [8]	✗	45.1	-	41.3
		✓	55.2	-	**45.6**

Table 2. **Part segmentation mIoU results.** We report the mIoU performance for each experiment with and without augmentations. For each test set (column) and model, we highlight in bold the best performance.

4.1. Part segmentation

Datasets. We use three publicly-available datasets: Pascal-Part [8], UDAPART [33] and CG-PART [32]. An overview of these datasets can be seen in Tab. 1. Pascal-Part contains part annotations for 15 object categories on the Pascal



(a) The distribution of damage area by damage type.

(b) The number of images containing each damage.

(c) Distribution of damage types appearing on car parts.

Figure 5. **Damage statisics in CrachCar101.** (a) The distribution of damage area for each damage. We observe the minimal area of cracks compared to the more extensive area of shattered glass. (b) The number of images containing every damage. It demonstrates a uniform initial damage selection, highlighting a subsequent preferential selection of more suitable damage types on visible parts. (c) The distribution of damage types on car parts. We observe a size-dependent occurrence of damage on distinct parts, wherein larger components exhibit heightened damage incidence.

VOC 2010 images [14]. We crop cars around their bounding boxes and keep those with at least 16,384 pixels and less than 75% background. This results in 612 images. The *license plate* category was removed due to the poor annotation and the limited representation. The pixels corresponding to this category are labeled as *front* or *back* category depending on their location. For UDAPART and CG-PART, we merge the four wheel classes to one and the two license plates to one to align with our part definition.

Evaluation sets. For evaluating our models, we use two test sets with real images: Pascal-Part [8] and UDAPART [33]. The Pascal-Part original test set consists of 62 test images and we evaluate our models using the annotations with 11 semantic parts. UDAPART consists of 40 images and we use the whole dataset for evaluating our models using the annotations with 27 semantic parts. To enable the training and testing across these datasets, we also evaluate models on UDAPART by merging the 27 fine-grained categories to the 11 coarser categories of Pascal-Part.

Training sets. We use four training sets to train our models: (a) the *Pascal-Part* training set with 490 real images, (b) the *CGPART* [32] training set with 31,448 synthetic images, (c) our *CrashCar101-Part* training set with 14,175 synthetic images, and (d) the combination of the *CrashCar101-Part* + *Pascal-Part* training sets. We train 18 part segmentation models in total using these sets. We train 12 models, three for each training set, with 11 part categories. We also train six more models with 27 part categories when the Pascal-Part set is not used (i.e., using the sets (b) and (c)).

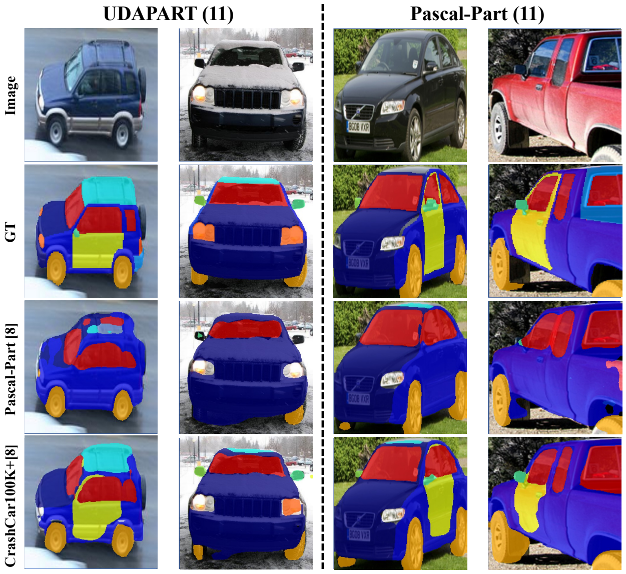


Figure 6. **Qualitative part segmentation results.** We show results from training on Pascal-Part and on CrashCar101+Pascal-Part using DeepLabv3 with a ResNet101 backbone. Both models were trained with augmentations (lines 10 and 16 in Tab 2). We observe that by including our synthetic data to the real training set, we obtain a model that yields better results.

Segmentation models. We employ three semantic segmentation models with and without augmentations. We utilize DeepLabv3 [6] with ResNet50 and ResNet101 backbones [20], both pre-trained on ImageNet [11]. Additionally, we use a B5-sized SegFormer model [67] pretrained on Cityscapes [9]. This approach provides a comprehensive evaluation of our dataset's part segmentation potential across diverse architectures and pre-training sources.

Results. We report the results of the 18 trained models with augmentations and without on our evaluation sets in Tab. 2. As expected, we observe that the augmentations make a substantial difference in the results of synthetic data (especially in the case of *CGPART* [32]). We observe that our synthetic data outperforms *CGPART*, the other state-of-the-art synthetic dataset in every instance. Even though the *CGPART* training set is about double in size compared to our training set (see Tab. 1), we show that our dataset is much better and more realistic due to our rendering procedure and the number of the different car models used (99 models in CrashCar101-Part vs. 6 models in CGPart).

Moreover, we observe that the most precise models are those trained on the combination of real and synthetic data. Still, when utilizing the Segformer model, our CrashCar101-Part dataset outperforms even the model trained on the combination.

Interestingly, we observe that when evaluating on the real images of UDAPart, the model trained only on our synthetic data (third row for each respective model) significantly outperforms the one trained on real images of Pascal-Part (top row of each respective model). In the case of Deeplabv3 using the ResNet101 backbone and Segformer models, there's a noticeable performance boost (+9.2-16.3% mIoU). We find the effect of the augmentations to be particularly interesting here. The synthetic data is not able to outperform real data without augmentations for the DeepLabv3 models, but when the training images are augmented, it outperforms even the real data trained without augmentations. Meanwhile, using the Segformer synthetic data performs better than real data outright. In Fig 6, we show qualitative test examples on both real datasets.

4.2. Damage segmentation

Damage dataset and evaluation set. We use the recently released CarDD dataset [63] which contains real images of damaged cars annotated with object segmentation masks for several damage categories. To align with the damage types of our synthetic data, we remove the category flat tires. Images containing only flat tires are filtered out and the pixels annotated as flat tires are set to background. An overview of the dataset can be seen at the bottom part of Tab. 1. We evaluate our models in this section on the CarDD test set [63] which consists of 349 manually annotated images.

Experimental setup. We evaluate the CrashCar101's sim2real transfer potential on damage segmentation using few-shot segmentation (FSS). FSS aims to segment novel objects with few annotations. Recent approaches [54,57,68,69] mitigate limited data by freezing the backbone, leveraging feature fusion and prototypes. For simplicity's sake, we perform FSS experiments in a similar fashion to the baseline method in [7]. We start by training on a large source dataset, namely COCO [31], ImageNet [11], or CrashCar101. We then fine-tune the model on $n \cdot k$ images from the CarDD training set and we use an internal validation set consisting of $n \cdot k$ to perform early stoppage. The parameter n denotes the number of shots and $k = 6$ is the number of classes (including background). The $n \cdot k$ images are selected such that all class labels are present. We experiment with various freezing strategies and we report results obtained without freezing because these perform best. To quantify the reduction in domain gap, we compare the pre-training on CrashCar101 to the one on COCO and ImageNet respectively.

Damage segmentation models. To show that the efficacy of CarCrash101 is architecture-independent, we perform FSS experiments using SegFormer and DeepLabV3 frameworks, each utilizing MiT-b5 and ResNet-101 backbones respectively. We adapt these models for damage segmentation by modifying the final layers to yield 6 channels, corresponding to distinct damage types.

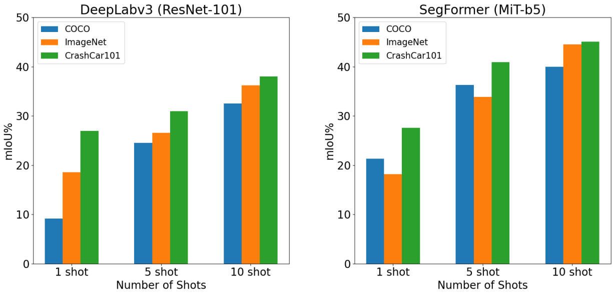


Figure 7. **Few-shot segmentation on damage segmentation using different model architecture**. Left: DeepLabv3 and right: SegFormer. Models pre-trained on CrashCar101 consistently outperform others regardless of the model architecture and without using any domain adaptation techniques.

Results. We report our FSS results in Fig. 7 where we show the mIoU performance of all models. Our results show that pretraining on CrashCar101 yields significantly better results (+6.3-17.9% mIoU at 1-shot and +4.4-7.0% at 5-shot) compared to using a pre-trained model on COCO or ImageNet. As expected, when the amount of real data increases the performance gain decreases, nonetheless we still see a marginal performance gain for the 10-shot experiments. Both SegFormer and DeepLabv3 perform better when pretrained on CrashCar101, which suggests that the improvement is independent of the model architecture. These results show that there is a smaller domain gap between CrashCar101 and CarDD than there is between COCO/ImageNet and CarDD. These results show the potential of the sim2real transfer of our dataset on the task of damage segmentation.

5. Conclusion

We proposed a procedural generation pipeline that creates damages on 3D cars. We executed our pipeline and rendered the CrashCar101 synthetic dataset. We showed that without any special modification or any domain adaptation methods, our CrashCar101 dataset is useful for training a damage assessment system that performs damage segmentation and semantic part segmentation on real images. We hope that our work will enable more work in this direction and lead to a more powerful synthetic data generation pipeline able to deal with a variety of different incidents such as natural disasters and damage assessment models that can operate on various objects beyond vehicles.

Acknowledgments. D. Papadopoulos was supported by the DFF Sapere Aude Starting Grant "ACHILLES". We would like to thank Thanos Delatolas for proofreading.

References

[1] Armen Avetisyan, Manuel Dahnert, Angela Dai, Manolis Savva, Angel X. Chang, and Matthias Nießner. Scan2cad: Learning cad model alignment in rgb-d scans. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2609–2618, 2019. 3

[2] Burak Balci., Yusuf Artan., Bensu Alkan., and Alperen Elihos. Front-view vehicle damage detection using roadway surveillance camera images. In *Proceedings of the 5th International Conference on Vehicle Technology and Intelligent Transport Systems - VEHITS,*, pages 193–198. INSTICC, SciTePress, 2019. 3

[3] Matteo Biasetton, Umberto Michieli, Gianluca Agresti, and Pietro Zanuttigh. Unsupervised domain adaptation for semantic segmentation of urban scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2019. 2

[4] 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, 2015. 2, 3, 4

[5] Kevin Chen, Christopher B Choy, Manolis Savva, Angel X Chang, Thomas Funkhouser, and Silvio Savarese. Text2shape: Generating shapes from natural language by learning joint embeddings. In *ACCV*. Springer, 2018. 3

[6] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *CoRR*, abs/1706.05587, 2017. 4, 7

[7] Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Frank Wang, and Jia-Bin Huang. A closer look at few-shot classification. *CoRR*, abs/1904.04232, 2019. 8

[8] Xianjie Chen, Roozbeh Mottaghi, Xiaobai Liu, Sanja Fidler, Raquel Urtasun, and Alan Yuille. Detect what you can: Detecting and representing objects using holistic models and body parts. In *CVPR*, 2014. 2, 3, 6, 7

[9] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. In *CVPR*, 2016. 1, 7

[10] César Roberto de Souza12, Adrien Gaidon, Yohann Cabon, and Antonio Manuel López. Procedural generation of videos to train deep action recognition networks. In *CVPR*, 2017. 2

[11] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-fei. ImageNet: A large-scale hierarchical image database. In *CVPR*, 2009. 2, 7, 8

[12] Najmeddine Dhieb, Hakim Ghazzai, Hichem Besbes, and Yehia Massoud. A very deep transfer learning model for vehicle damage detection and localization. In *2019 31st International Conference on Microelectronics (ICM)*, pages 158–161, 2019. 3

[13] Mahavir Dwivedi, Hashmat Shadab Malik, S. N. Omkar, Edgar Bosco Monis, Bharat Khanna, Satya Ranjan Samal, Ayush Tiwari, and Aditya Rathi. Deep learning-based car damage classification and detection. In Niranjan N. Chiplunkar and Takanori Fukao, editors, *Advances in Artificial Intelligence and Data Engineering*, pages 207–221, Singapore, 2021. Springer Singapore. 3

[14] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes (VOC) Challenge. *IJCV*, 2010. 7

[15] Maayan Frid-Adar, Idit Diamant, Eyal Klang, Michal Amitai, Jacob Goldberger, and Hayit Greenspan. Gan-based synthetic medical image augmentation for increased cnn performance in liver lesion classification. *Neurocomputing*, 321:321–331, 2018. 2

[16] Adrien Gaidon, Qiao Wang, Yohann Cabon, and Eleonora Vig. Virtual worlds as proxy for multi-object tracking analysis. In *CVPR*, 2016. 1, 2

[17] Agrim Gupta, Piotr Dollar, and Ross Girshick. Lvis: A dataset for large vocabulary instance segmentation. In *CVPR*, 2019. 1

[18] Ritwik Gupta, Richard Hosfelt, Sandra Sajeev, Nirav Patel, Bryce Goodman, Jigar Doshi, Eric Heim, Howie Choset, and Matthew Gaston. xbd: A dataset for assessing building damage from satellite imagery. *arXiv preprint arXiv:1911.09296*, 2019. 3

[19] Hironori Hattori, Vishnu Naresh Boddeti, Kris M Kitani, and Takeo Kanade. Learning scene-specific pedestrian detectors without real data. In *CVPR*, 2015. 2

[20] 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 (CVPR)*, June 2016. 4, 7

[21] Mark Hendrikx, Sebastiaan Meijer, Joeri Van Der Velden, and Alexandru Iosup. Procedural content generation for games: A survey. *ACM Trans. Multimedia Comput. Commun. Appl.*, 9(1), feb 2013. 2

[22] Charlie Hewitt, Tadas Baltrušaitis, Erroll Wood, Lohit Petikam, Louis Florentin, and Hanz Cuevas Velasquez. Procedural humans for computer vision. *arXiv preprint arXiv:2301.01161*, 2023. 2

[23] Charlie Hewitt, Tadas Baltrušaitis, Erroll Wood, Lohit Petikam, Louis Florentin, and Hanz Cuevas Velasquez. Procedural humans for computer vision, 2023. 2

[24] Wei-Chih Hung, Varun Jampani, Sifei Liu, Pavlo Molchanov, Ming-Hsuan Yang, and Jan Kautz. Scops: Self-supervised co-part segmentation. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 869–878, 2019. 2

[25] Neo. Kaitling. Car damage image dataset. `https://github.com/neokt/car-damage-detective`, 2017. 3

[26] Samin Khan, Buu Phan, Rick Salay, and Krzysztof Czarnecki. Procsy: Procedural synthetic dataset generation towards influence factor studies of semantic segmentation networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2019. 1, 2

[27] Samin Khan, Buu Phan, Rick Salay, and Krzysztof Czarnecki. Procsy: Procedural synthetic dataset generation towards influence factor studies of semantic segmentation networks. In *CVPR workshops*, pages 88–96, 2019. 2

[28] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 6

[29] Pei Li, Bingyu Shen, and Weishan Dong. An anti-fraud system for car insurance claim based on visual evidence. *CoRR*, abs/1804.11207, 2018. 3

[30] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2999–3007, 2017. 6

[31] T-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C.L. Zitnick. Microsoft COCO: Common objects in context. In *ECCV*, 2014. 1, 2, 8

[32] Qing Liu, Adam Kortylewski, Zhishuai Zhang, Zizhang Li, Mengqi Guo, Qihao Liu, Xiaoding Yuan, Jiteng Mu, Weichao Qiu, and Alan Yuille. Cgpart: A part segmentation dataset based on 3d computer graphics models. *arXiv preprint arXiv:2103.14098*, 2021. 3, 6, 7

[33] Qing Liu, Adam Kortylewski, Zhishuai Zhang, Zizhang Li, Mengqi Guo, Qihao Liu, Xiaoding Yuan, Jiteng Mu, Weichao Qiu, and Alan Yuille. Learning part segmentation through unsupervised domain adaptation from synthetic vehicles. In *CVPR*, 2022. 1, 2, 3, 6, 7

[34] LPLENKA. Coco car damage detection dataset. `https://www.kaggle.com/datasets/lplenka/coco-car-damage-detection-dataset`, 2020. 3

[35] Hiroya Maeda, Yoshihide Sekimoto, Toshikazu Seto, Takehiro Kashiyama, and Hiroshi Omata. Road damage detection and classification using deep neural networks with smartphone images: Road damage detection and classification. *Computer-Aided Civil and Infrastructure Engineering*, 33, 06 2018. 3

[36] Kaichun Mo, Shilin Zhu, Angel X Chang, Li Yi, Subarna Tripathi, Leonidas J Guibas, and Hao Su. Partnet: A large-scale benchmark for fine-grained and hierarchical part-level 3d object understanding. In *CVPR*, 2019. 3

[37] Sergey I Nikolenko. *Synthetic data for deep learning*, volume 174. Springer, 2021. 2

[38] Curtis Northcutt, Lu Jiang, and Isaac Chuang. Confident learning: Estimating uncertainty in dataset labels. *Journal of Artificial Intelligence Research*, 70:1373–1411, 2021. 1

[39] Curtis G Northcutt, Anish Athalye, and Jonas Mueller. Pervasive label errors in test sets destabilize machine learning benchmarks. *arXiv preprint arXiv:2103.14749*, 2021. 1

[40] Kitsuchart Pasupa, Phongsathorn Kittiworapanya, Napasin Hongngern, and Kuntpong Woraratpanya. Evaluation of deep learning algorithms for semantic segmentation of car parts. *Complex & Intelligent Systems*, 2021. 2

[41] Naeem Patel, Shantanu Shinde, and Freddy Poly. Automated damage detection in operational vehicles using mask r-cnn. In Hari Vasudevan, Antonis Michalas, Narendra Shekokar, and Meera Narvekar, editors, *Advanced Computing Technologies and Applications*, pages 563–571, Singapore, 2020. Springer Singapore. 3

[42] Kalpesh Patil, Mandar Kulkarni, Anand Sriraman, and Shirish Karande. Deep learning based car damage classification. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 50–54, 2017. 1, 3

[43] Xingchao Peng, Baochen Sun, Karim Ali, and Kate Saenko. Learning deep object detectors from 3d models. In *ICCV*, 2015. 2

[44] Siyuan Qi, Yixin Zhu, Siyuan Huang, Chenfanfu Jiang, and Song-Chun Zhu. Human-centric indoor scene synthesis using stochastic grammar. *CoRR*, abs/1808.08473, 2018. 2

[45] Stephan R. Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. *CoRR*, abs/1608.02192, 2016. 2

[46] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio Lopez. The SYNTHIA Dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *CVPR*, 2016. 1, 2

[47] B. C. Russell, K. P. Murphy, and W. T. Freeman. LabelMe: a database and web-based tool for image annotation. *IJCV*, 2008. 1

[48] Oindrila Saha, Zezhou Cheng, and Subhransu Maji. Improving few-shot part segmentation using coarse supervision. In Shai Avidan, Gabriel Brostow, Moustapha Cissé, Giovanni Maria Farinella, and Tal Hassner, editors, *ECCV*, 2022. 2

[49] Nithya Sambasivan, Shivani Kapania, Hannah Highfill, Diana Akrong, Praveen Paritosh, and Lora M Aroyo. "everyone wants to do the model work, not the data work": Data cascades in high-stakes ai. In *proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–15, 2021. 1

[50] Swami Sankaranarayanan, Yogesh Balaji, Arpit Jain, Ser Nam Lim, and Rama Chellappa. Learning from synthetic data: Addressing domain shift for semantic segmentation. In *CVPR*, 2018. 1

[51] Manolis Savva, Fisher Yu, Hao Su, M Aono, B Chen, D Cohen-Or, W Deng, Hang Su, Song Bai, Xiang Bai, et al. Shrec16 track: largescale 3d shape retrieval from shapenet core55. In *Proceedings of the eurographics workshop on 3D object retrieval*, 2016. 3

[52] Noor Shaker, Julian Togelius, and Mark J Nelson. Procedural content generation in games. 2016. 2

[53] Vaishaal Shankar, Rebecca Roelofs, Horia Mania, Alex Fang, Benjamin Recht, and Ludwig Schmidt. Evaluating machine accuracy on imagenet. In *International Conference on Machine Learning*, pages 8634–8644. PMLR, 2020. 1

[54] Zhiqiang Shen, Zechun Liu, Jie Qin, Marios Savvides, and Kwang-Ting Cheng. Partial is better than all: Revisiting fine-tuning strategy for few-shot learning. *CoRR*, abs/2102.03983, 2021. 8

[55] Jian Shi, Yue Dong, Hao Su, and Stella X. Yu. Learning non-lambertian object intrinsics across shapenet categories. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5844–5853, 2017. 3

[56] Ranjodh Singh, Meghna P. Ayyar, Tata Sri Pavan, Sandeep Gosain, and Rajiv Ratn Shah. Automating car insurance claims using deep learning techniques. *2019 IEEE Fifth International Conference on Multimedia Big Data (BigMM)*, pages 199–207, 2019. 3

[57] Hao Tang, Zechao Li, Zhimao Peng, and Jinhui Tang. Blockmix: Meta regularization and self-calibrated inference for

metric-based meta-learning. In *Proceedings of the 28th ACM International Conference on Multimedia*, MM '20, page 610–618, New York, NY, USA, 2020. Association for Computing Machinery. 8

[58] Julian Togelius, Emil Kastbjerg, David Schedl, and Georgios Yannakakis. What is procedural content generation? mario on the borderline. 06 2011. 2

[59] 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 *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1082–10828, 2018. 2

[60] R.E. van Ruitenbeek and S. Bhulai. Convolutional neural networks for vehicle damage detection. *Machine Learning with Applications*, 9:100332, sep 2022. 3

[61] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The caltech-ucsd birds-200-2011 dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011. 2

[62] Niannian Wang, Xuefeng Zhao, Zheng Zou, Peng Zhao, and Fei Qi. Autonomous damage segmentation and measurement of glazed tiles in historic buildings via deep learning. *Computer-Aided Civil and Infrastructure Engineering*, 35(3):277–291, 2020. 2

[63] Xinkuang Wang, Wenjing Li, and Zhongcheng Wu. Cardd: A new dataset for vision-based car damage detection, 2022. 1, 3, 4, 6, 8

[64] Ethan Weber, Nuria Marzo, Dim P Papadopoulos, Aritro Biswas, Agata Lapedriza, Ferda Ofli, Muhammad Imran, and Antonio Torralba. Detecting natural disasters, damage, and incidents in the wild. In *ECCV*, 2020. 1, 3

[65] Ethan Weber, Dim P Papadopoulos, Agata Lapedriza, Ferda Ofli, Muhammad Imran, and Antonio Torralba. Incidents1m: a large-scale dataset of images with natural disasters, damage, and incidents. *IEEE transactions on pattern analysis and machine intelligence*, 2022. 1, 3

[66] Erroll Wood, Tadas Baltrušaitis, Charlie Hewitt, Sebastian Dziadzio, Thomas J Cashman, and Jamie Shotton. Fake it till you make it: face analysis in the wild using synthetic data alone. In *ICCV*, 2021. 2

[67] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M. Alvarez, and Ping Luo. Segformer: Simple and efficient design for semantic segmentation with transformers. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 12077–12090. Curran Associates, Inc., 2021. 7

[68] Lihe Yang, Wei Zhuo, Lei Qi, Yinghuan Shi, and Yang Gao. Mining latent classes for few-shot segmentation. *CoRR*, abs/2103.15402, 2021. 8

[69] Gengwei Zhang, Guoliang Kang, Yunchao Wei, and Yi Yang. Few-shot segmentation via cycle-consistent transformer. *CoRR*, abs/2106.02320, 2021. 8

[70] Qinghui Zhang, Xianing Chang, and Shanfeng Bian Bian. Vehicle-damage-detection segmentation algorithm based on improved mask rcnn. *IEEE Access*, 8:6997–7004, 2020. 3

[71] Wei Zhang, Yuan Cheng, Xin Guo, Qingpei Guo, Jian Wang, Qing Wang, Chen Jiang, Meng Wang, Furong Xu, and Wei Chu. Automatic car damage assessment system: Reading and understanding videos as professional insurance inspectors. In *AAAI*, 2020. 1

[72] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba. Scene parsing through ADE20K dataset. In *CVPR*, 2017. 1

[73] Yanzhao Zhou, Yi Zhu, Qixiang Ye, Qiang Qiu, and Jianbin Jiao. Weakly supervised instance segmentation using class peak response. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3791–3800, 2018. 2