Bent & Broken Bicycles: Leveraging synthetic data for damaged object re-identification - Supplementary Material

Luca Piano, Filippo Gabriele Prattico, Alessandro Sebastian Russo, Lorenzo Lanari, Lia Morra, Fabrizio Lamberti
Department of Control and Computer Engineering, Politecnico di Torino, Torino, Italy
{luca.piano,filippogabriele.prattico,alessandrosebastian.russo}@polito.it,
lorenzo.lanari@studenti.polito.it,{lia.morra,fabrizio.lamberti}@polito.it

A. Dataset

A.1. CGI Pipeline implementation details

In this section we provide additional details on the CGI pipeline used to implement the BBBicycles dataset, and in particular on the transformations that are randomly applied to generate "before" and "after" images.

A.1.1 Template rig.

A bike contains many movable elements which need to be positioned (to randomly change the bike pose) or deformed (to simulate damages). In order to randomly apply these transformations, we defined a *template rig* that needs to be adapted to match the given bike model. The template rig is composed of an "armature" (Figure 1a), "lattices" (Figure 1b), and "rail guides", placed as depicted in Figure 1c.

(a) (b) (c)

Figure 1: Template rig adaptation and skinning: (a) armature with bone groups and layers, (b) fitting lattice to wheel meshes, and (c) rail guides positioning.

A.1.2 Image rendering.

For each ID, multiple images are generated by applying the following transformations:

1. Mobile parts composition. This is accomplished by randomly performing one or more actions among: translating the seat and handlebar; rotating the seat, handlebar, pedals, and wheels. The allowed range of movement for each model is set during the rig adaptation.
2. Dirt. A custom shader is used to randomly apply dirt in the form of mud or rust, with a predetermined probability.
3. (Optional) Removing parts. The seat, pedals, handlebar and wheels can be removed with a predetermined probability.
4. (Optional) Damaging parts (frame excluded). Parts of the bikes can be damaged, by selecting a deformation either from the wheels' library or from the pose library.

5. (Optional) Damaging the frame. The frame can be damaged either by picking a deformation from the pose library (Bent Frame) and/or by breaking it using the rail guides-boolean system (Broken frame). Hence, four possible damage categories are possible: normal frame, bent frame, broken frame, or bent & broken frame.

6. Point of View selection. The virtual camera position is chosen randomly within a given boundary, by randomly switching the visible side of the bike, as well as randomly adjusting the camera focal length within a parametrized range.

7. Environment and Lighting. A combination of background and lighting setup is picked.

8. Segmentation. The bike is segmented in the following classes: "Front Wheel", "Rear Wheel", "Seat", "Crankset", and "Frame". Segmentation was implemented using the *bpycv* [1] library. An example of segmentation is shown in Figure 2

(a) Intact frame

(b) Broken frame

Figure 2: Examples of the CGI pipeline auxiliary outputs. From right to left: segmentation, rendered image, and depth map.

[1] https://github.com/DIYer22/bpycv

A.2. The BBBicycles dataset

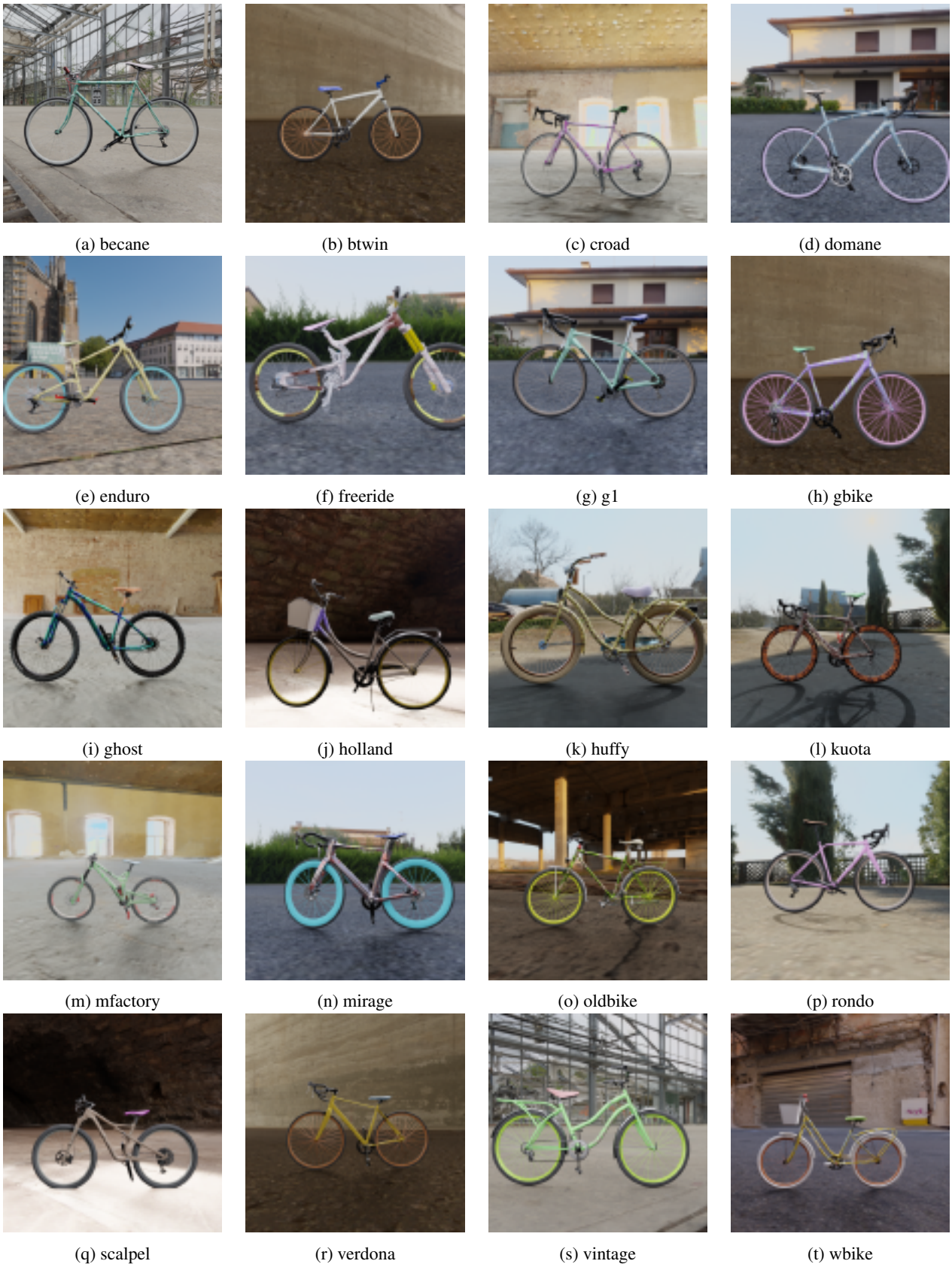
In this section we provide additional details on the generated dataset to better illustrate the variety of models and damages/deformations included in the BBBicycles dataset.

A.2.1 3D Bike models

BBBicycles contains images generated from 20 3D bike models retrieved from dedicated marketplaces. It includes several variants of popular bikes such as Road, Cruiser and MTB. In particular, it contains 6 MTB, 1 Enduro, 6 Road, 1 Circuit, 1 Gravel, and 5 Cruiser. The list of bike models per category is illustrated in Table 1: each bike model was assigned to either the training, validation, or (stress) test set. Examples of renderings from each model are shown in Figure 3.

Table 1: Bike model distribution across BBBicycles in the training, validation and test set. Models marked with (*) are shared between Train and Validation.

Category	Train	Validation	Test
MTB	mfactory ghost freeride* scalpel*	becane btwin	-
Road	rondo verdona ghost domane* g1* kuota*	croad	-
Cruiser	oldbike holland* huffy* vintage* wbike*		-
Enduro	-		enduro
Circuit	-		mirage
Gravel	-		gbike



(a) becane (b) btwin (c) croad (d) domane

(e) enduro (f) freeride (g) g1 (h) gbike

(i) ghost (j) holland (k) huffy (l) kuota

(m) mfactory (n) mirage (o) oldbike (p) rondo

(q) scalpel (r) verdona (s) vintage (t) wbike

Figure 3: Examples of each model used to generate the dataset.

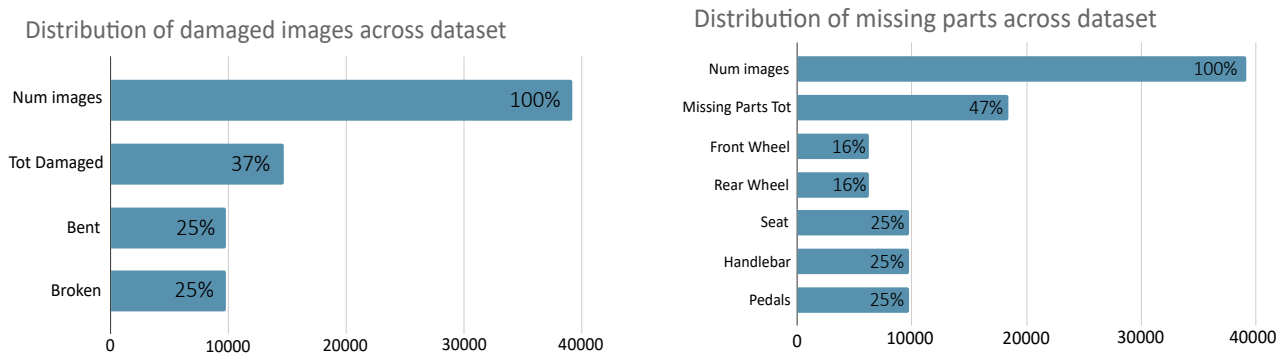


Figure 4: Distribution of damages and missing parts across the synthethic dataset.

A.2.2 Damage distribution and examples

As illustrated in Section 3, in BBBicycles 50% of the images are generated "before" and 50% "after" a damage occurs. "Before" bikes have a 25% probability of being dirty, while "after" bikes have a 50% chance. "After" bikes are further divided into 25% undamaged and 75% bent, broken or both. As a result, 37% of the total images are damaged (see Figure 4). Examples of damaged and undamaged synthetic bike renderings are shown in Figure 5, Figure 6, Figure 7 and Figure 8.

Figure 5: Examples of bike renderings without damages (to the frame).

Figure 6: Examples of bike renderings with bent frames.

Figure 7: Examples of bike renderings with broken frames.

Figure 8: Examples of bike renderings with bent and broken frames.

Moreover, we set additional labels for each image according to the missing parts of the bike, namely: Front Wheel, Rear Wheel, Seat, Handlebar, Pedals. In the annotations, missing parts are represented by a One-hot vector encoding, where each vector value indicates if the corresponding part is present (0) or not (1), as exemplified in Fig. 11.

Figure 9: One-hot encoding example: the illustrated image only misses the "Rear Wheel", "Seat" and "Handlebar" parts, hence it has been labeled as "01110".

A.3. Real dataset acquisition: additional details.

In this section, we provide additional details on how the real dataset was assembled and annotated.

A.3.1 Delft Bikes.

The DelftBikes dataset[2] was originally designed to study whether deep neural networks could hallucinate missing parts in objects. It contains 10,000 bike images with 22 densely annotated parts for each bike. All part locations and part states (i.e., missing, intact, damaged, occluded) are explicitly annotated.

Specifically, we retained only the images from the DelftBikes training set with complete annotations (for some images missing parts annotations were not available), for a total of 8,000 images. Then, we translated the Delftbikes annotations to be compatible with the synthetic dataset annotations, as follows:

- For Front Wheel, Rear Wheel and Seat, we labeled the part as missing if the corresponding part was labeled in the same way (i.e., object state class = missing) in the Delftbikes dataset.
- For Handlebar, we labeled the part as missing if all parts belonging to the group {back handle, front handle, back hand break, front hand break, steer} were also labeled in the same way (i.e., object state class = missing) in the Delftbikes dataset.

[2]available https://data.4tu.nl/articles/dataset/DelftBikes$_{d}ata_{u}nderlying_{t}he_{p}ublication_{H}allucination_{I}n_{O}bject_{D}etection - A_{S}tudy_{I}n_{V}isual_{P}art_{V}erification/14866116$.

- For Pedals, we labeled the part as missing if both parts in the group {front_pedal, back_pedal} were also labeled in the same way (i.e, object state class = 2) in the Delftbikes dataset.

None of the bike instances in the DelftBikes dataset presented damages to the frame.

A.3.2 Web scraping details

We collected samples of real damaged bikes by querying popular search engines (i.e., Google, Bing) and online forums (i.e., Reddit and other dedicated forums). We used different keywords (i.e., "damaged bike", "bici danneggiata", etc.) in different languages (i.e., English, Italian, Spanish, French, etc.) in order to increase the number of matches. In particular, we selected countries with higher bike usage like the Netherlands and Denmark. Synonyms of damage were searched to amplify the number of returned images (for instance, "broken bike" and "damaged bike" produce different search results). Additional images of normal bikes were retrieved from second-hand e-commerce sites.

For each scraped image, the origin URL has been serialized as a source reference. The results have then been pruned from unrelated (e.g., excluding images about bike helmets, cycling suits, etc.) and duplicated images by hand and by means of automatic de-duplication techniques, respectively. In particular, we chose a de-duplication technique based on pre-trained CNNs, which marks as duplicated images those with a pairwise similarity score above a given threshold value (experimentally set to 85%).

A.3.3 Labelling.

All images were manually labeled indicating the damage type and missing parts. Labels were assigned as uniformly as possible to the synthetic dataset. Concerning damage labeling, we set four different labels based on the type of damage present on the bike frame:

- normal: the bike frame is intact, regardless of the condition of the other parts of the bike (e.g., missing parts, damaged wheels, damaged seat).
- bent: the bike frame is bent or presents damage, but it is broken in multiple pieces.
- broken: the bike frame is broken and clearly divided in pieces, and each piece does not present any bending.
- bent & broken: the bike frame is broken and the frame pieces show signs of bending.

For missing parts, we follow the same convention of the synthetic dataset and set additional labels for the following parts: Front Wheel, Rear Wheel, Seat, Handlebar, Pedals. Missing parts are represented by a One-hot vector encoding, where each vector value indicates if the corresponding part is present (0) or not (1), as exemplified in Fig. 11.

B. Experimental setting

In this section, we provide full details on the TransReI3D architecture and training details .

B.1. Data preprocessing

We preprocessed each input image sample by applying the following transformations:

- image resized to 256×256
- image padding to 276×276
- image pixel values rescaling to the range $[0, 1]$
- image standardization using mean and standard deviation computed on the synthetic dataset

(a) Normal

(b) Bent

(c) Broken

(d) Bent & broken

Figure 10: Examples of real images with normal (a), bent (b), broken (c) and bent & broken (d) frames.

Figure 11: Example of real image with missing parts: the illustrated image only misses the “Front wheel” and “Seat” parts, hence it has been labeled with “10100”.

B.2. Data augmentation

We employed the following data augmentation transformations:

- random horizontal flip
- random crop of size 256×256 (i.e., the input size required by TransReI3D)
- random gaussian blur, using a (3, 3) kernel with a standard deviation of of (0.1, 2.0)

- random gaussian noise, with mean=0.1 and std=0.07

B.3. Architecture details

B.3.1 Hyper-parameters

The default configurations for training used are the following :

- Batch size: 32
- Number of epochs: 20
- Optimizer: SGD with momentum = 0.9
- Base Learning rate: 0.01
- LR Scheduler: Cosine scheduler with linear warmup
  - Initial Learning rate = 0.01 * Base Learning rate
  - Warmup time = 5 epochs
  - Maximum Learning rate = Base Learning rate
  - Minimum Learning rate = 0.002 * Base Learning rate
- Weight intialization:
  - The pretrained ImageNet weights suggested in the official TransReID GitHub code implementation[3] for the backbone layers
  - Normal distribution with mean=0 and std=0.001 and constant bias = 0 for Linear classifier layers
  - Constant weight = 1 and bias = 0 for BatchNorm Layers
  - Kaiming Normal initialization for the remaining Layers
- Weight Decay: 1e-4
- Losses:
  - Triplet loss with no margin for ReID (weight α=1)
  - Cross entropy loss for ID classification (weight β=1)
  - A combination of three cross entropy losses for total damage loss (weight γ=1):
    - Bend classification (λ=0.25)
    - Broken classification (μ=0.25)
    - Missing part classification (ν=0.5, mean average over all missing parts losses)

B.3.2 Components Details

Here we provide a brief overview of each component of the TransReI3D architecture:

- **(Input) patch embeddings**: analogously to ViT, each input image is split into N patches which are then linearly projected into N patch embeddings. In our case, we split each image into overlapping patches of size 16×16, with patch stride set to 12×12 (as in the original TransReID implementation). The embedding size M is set to 768.
- **Side Information Embedding (SIE)**: a SIE module creates a general SIE embedding containing the information regarding the camera focal c (=$\{20,..,38\}$) and view direction v (=$\{$"right", "left"$\}$) associated with the image I. The embedded value S (= "right", "left") is obtained by combining the two informations through the following formula:

$$\mathcal{S}_I = v_I \mathcal{N}_c + c_I \tag{1}$$

  where $\mathcal{N}_c$ is the cardinality of c. The information is then embedded into a vector of size $M = 768$. A similar approach is taken for the position embedding, which encodes the position of each patch relative to the others.

[3] downloaded from https://github.com/rwightman/pytorch-image-models/releases/download/v0.1-vitjx/jx_$vit_base_p16_224-80ecf9dd.pth$.

- **Patch tokens and $[cls]$ Token**: similarly to ViT, a *classification [cls] embedding*, which encodes image global features, is prepended to the sequence of N *patch tokens*. Each token is encoded by the combination (sum) of the corresponding patch embedding, learnable positional embedding, and SIE embeddings. The $N+1$ input tokens, inclusive of the [*cls*] token for a total size of $[1, N+1, 768]$, are then input into the transformer backbone.
- **Shared ViT Network**: a ViT-like structure including $L-1$ layers ($L=12$) is used as a shared backbone, whose output is then passed to each task-dedicated branch, each including an additional separate L^{th} transformer layer; each layer attention module has 12 attention heads.
- **ReID global branch**: the ReID task is performed based on the $[cls]$ token alone (which is a global representation of the image features). The token is first passed through a Batch Normalization (BN) layer, whose output is first used for the triplet loss calculation and then passed to a FC layer for performing the ID cross-entropy loss computation.
- **Jigsaw Branch and Jigsaw Patch Module**: in the Jigsaw branch, the Jigsaw Patch Module (JPM) module is applied on the output of the $L-1$ shared transformer layers: first the $[cls]$ token is separated from the output of the $L-1$ layers, while the remaining part of the output, consisting only of the patch tokens, is randomly rearranged into four equally $N/4$ sized groups. Then, the previously extracted $[cls]$ token is added to each group so obtained, and each group is finally passed to L^{th} transformer layer of the JPM branch; the output of the JPM branch is a set of classification tokens, one for each group. In the same way as in the global branch, each output [*cls*] token is passed through a corresponding BN layer, whose output is used for the triplet loss calculation and then passed to the corresponding FC layer for the ID cross-entropy loss. These loss components are added to the combined loss of the global branch to be minimized. In this way, the ReID model learns more discriminative parts and becomes more robust with respect to perturbations.
- **Damage branch**: like for the ReID task, the damage classification is performed on the [*cls*] token alone. The token is first passed through 7 different BN layers (one per head), and each output is passed to a corresponding FC layer, one for Bend frame classification, one for Broken frame classification, and one for each missing part classification. The scores and cross-entropy losses produced by each of these heads are then combined by weighted averaging for the final damage loss.

B.4. Domain adaptation

The resulting architecture configurations after the addition of DANN and PADA are depicted in Fig.12 and Fig.13, respectively.

Parameters used for domain adaptation are:

- $\theta = 1.0$ as weight for the domain discriminator loss $mathcalL_{dmn}$ (Equation 2).
- $\delta = 1.0$ as weight for the model classification loss $\mathcal{L}_{mdl}$, when PADA is active, otherwise 0 (Equation 2).
- Gradient Reversal Layer weight $\iota = 1.0$ in all DANN and PADA experiments except for the Base + Real + DANN experiment, in which $\iota = 10.0$ (Equation 2).

$$\mathcal{L}_{D\_tot} = \mathcal{L}_D + \theta\mathcal{L}_{dmn} + \delta\mathcal{L}_{mdl} - \iota\frac{\partial\mathcal{L}_{DMN}}{\partial f_g} \tag{2}$$

C. Additional results

C.1. Retrieval examples

Figure 14 depicts some example predictions of TransReI3D on the ReID task.

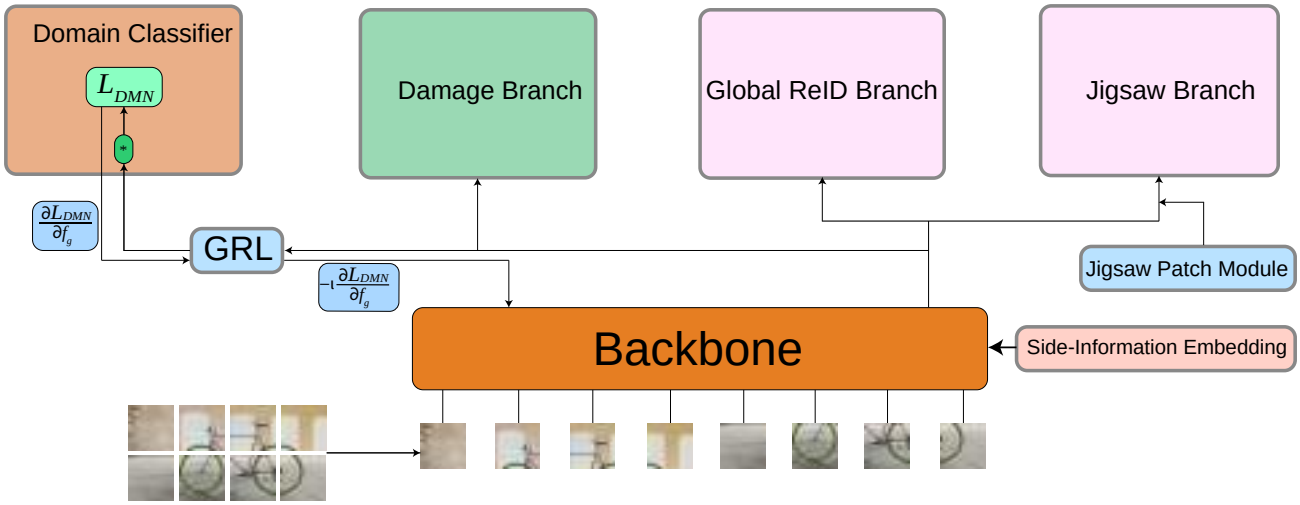


Figure 12: TransReI3D architecture with the addition of DANN components.

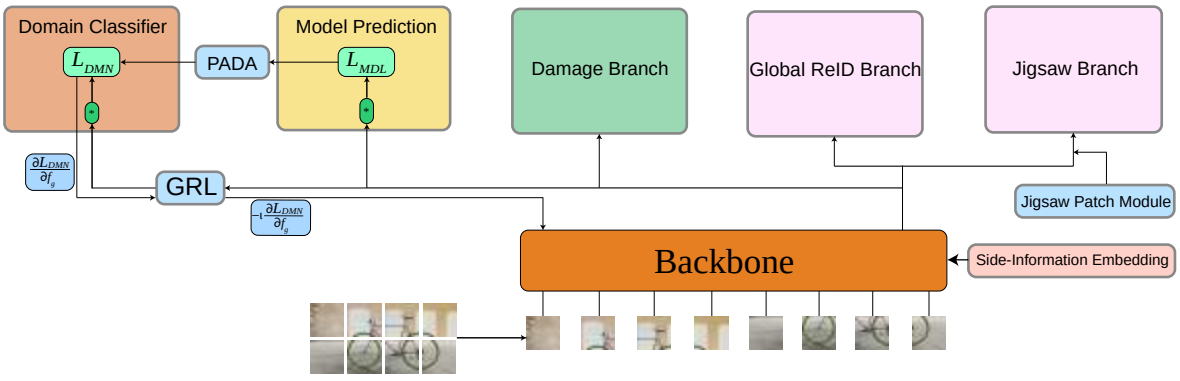


Figure 13: TransReI3D architecture with the addition of the PADA module.

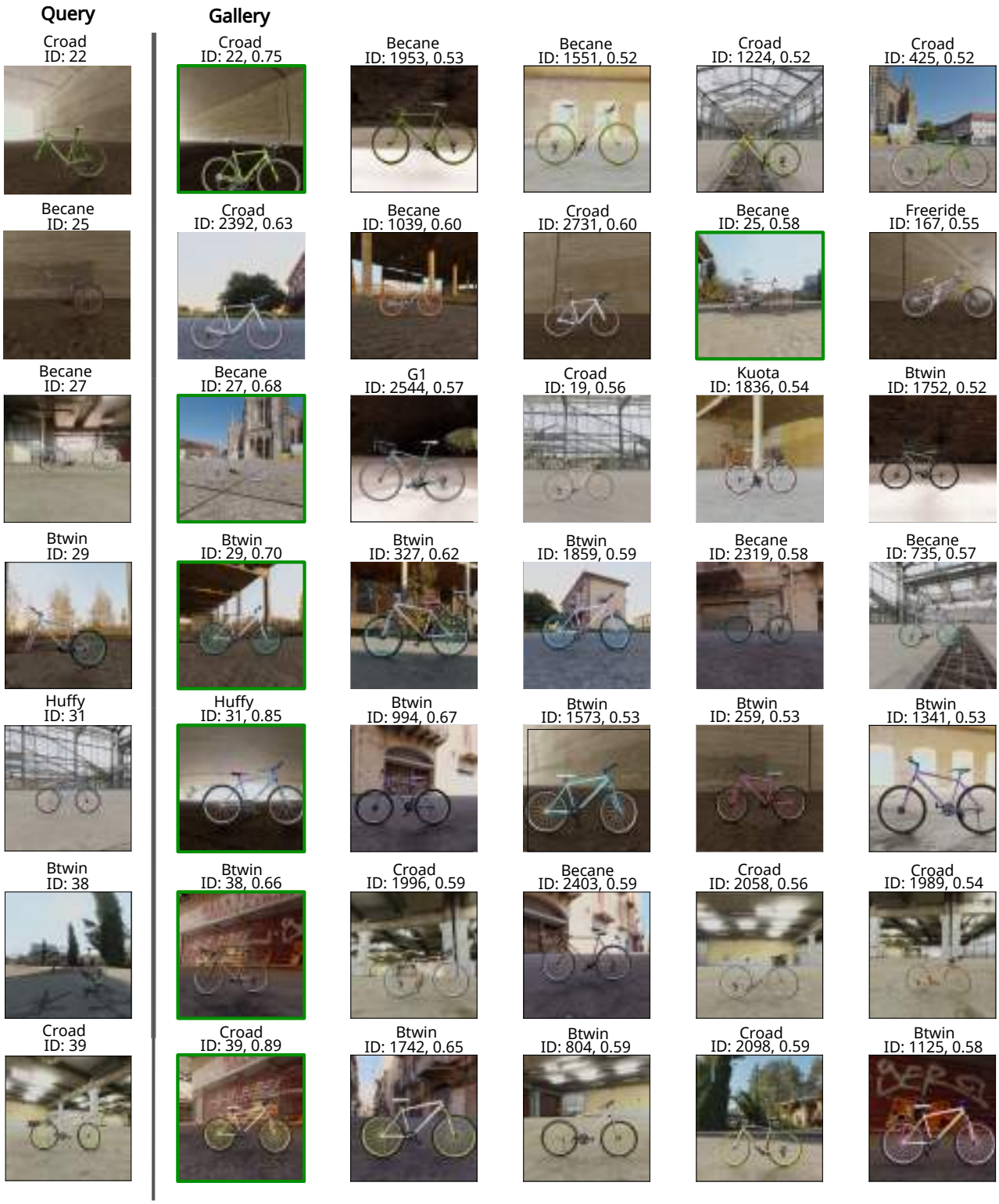


Figure 14: Retrieval results (Top-5 images) for the Baseline configuration, and corresponding model, bike ID, similarity scores. In most cases, the correct result (shown with a green border) is within the Top-5 predictions.

C.2. Effect of the background on ReID and DD tasks

In this section, we report additional experiments with different backgrounds to understand its effect on the ReID and DD tasks. Specifically, we compared three techniques: (i) the use of HDRI images, with random camera position, to generate the background, as detailed in Section 3; (ii) randomly picking an image from Places365 as background, and (iii) the use of a simple uniform background. To generate samples with Places365 images as backgrounds, we leveraged the original pipeline to render an auxiliary image with a transparent background and overlay it over the photo taken from the dataset. Examples are shown in Figure 15. It should be noticed that the proposed pipeline leverages a limited number of 360° HDRI maps, and even if the proposed pipeline can generate a virtually infinite number of backgrounds by varying the bike position, camera and illumination, the backgrounds will be visually correlated. On the other hand, Places365 contains a much wider range of scenes, but since the bike is randomly positioned, the resulting blend is not always realistic, and the foreground and background are not as consistent as with HDRI maps.

In Table 2, we evaluate the ability of TransReI3D to generalize to the real domain in the following transfer scenarios: HDRI $\rightarrow$ Real, Places365 $\rightarrow$ Real and Uniform $\rightarrow$ Real. We found moderately better results when employing the HDRI results, although when real images are available at training time, with HDRI yielding a marginal improvement over Places365.

We further assess the ability to transfer across synthetic domains, specifically we evaluate the following scenarios: Places 365 $\rightarrow$ HDRI and Uniform $\rightarrow$ HDRI. We found that the network generalizes quite well across different strategies to insert the background, as long as it is not uniform. In the latter case, the performance significantly drops as the network is no longer able to separate the bike from the background.

Based on these results, we conclude that the proposed pipeline contains sufficiently varied backgrounds, and the higher consistency improves the generalization capabilities.

Table 2: TransReI3D performance on the validation set with different strategies to generate the background. The network was trained on synthetic data except for † (labeled real images available at training time) and ‡ (unlabelled real images available at training time).

	[HTML]EFEFEF**Validation**					
	Damage Detection		**Re-identification (Synthetic)**			
	Real AUC	**Synthetic AUC**	**mAP**	**CMC-1**	**CMC-5**	**CMC-10**
BG HDRI + Real†	**97.3 ± 2.2**	**91.4 ± 0.2**	**85.3 ± 0.2**	**79.4 ± 0.1**	**92.9 ± 0.4**	**96.6 ± 0.4**
BG Places365 + Real †	96.3 ± 1.9	90.4 ± 0.2	85 ± 0	79.0 ± 0.4	92.8 ± 0.3	96.3 ± 0.2
BG Uniform + Real †	95.2 ± 3.4	87.4 ± 1.5	48.5 ± 3.4	39.2 ± 1.9	59.4 ± 5.6	66.0 ± 5.7

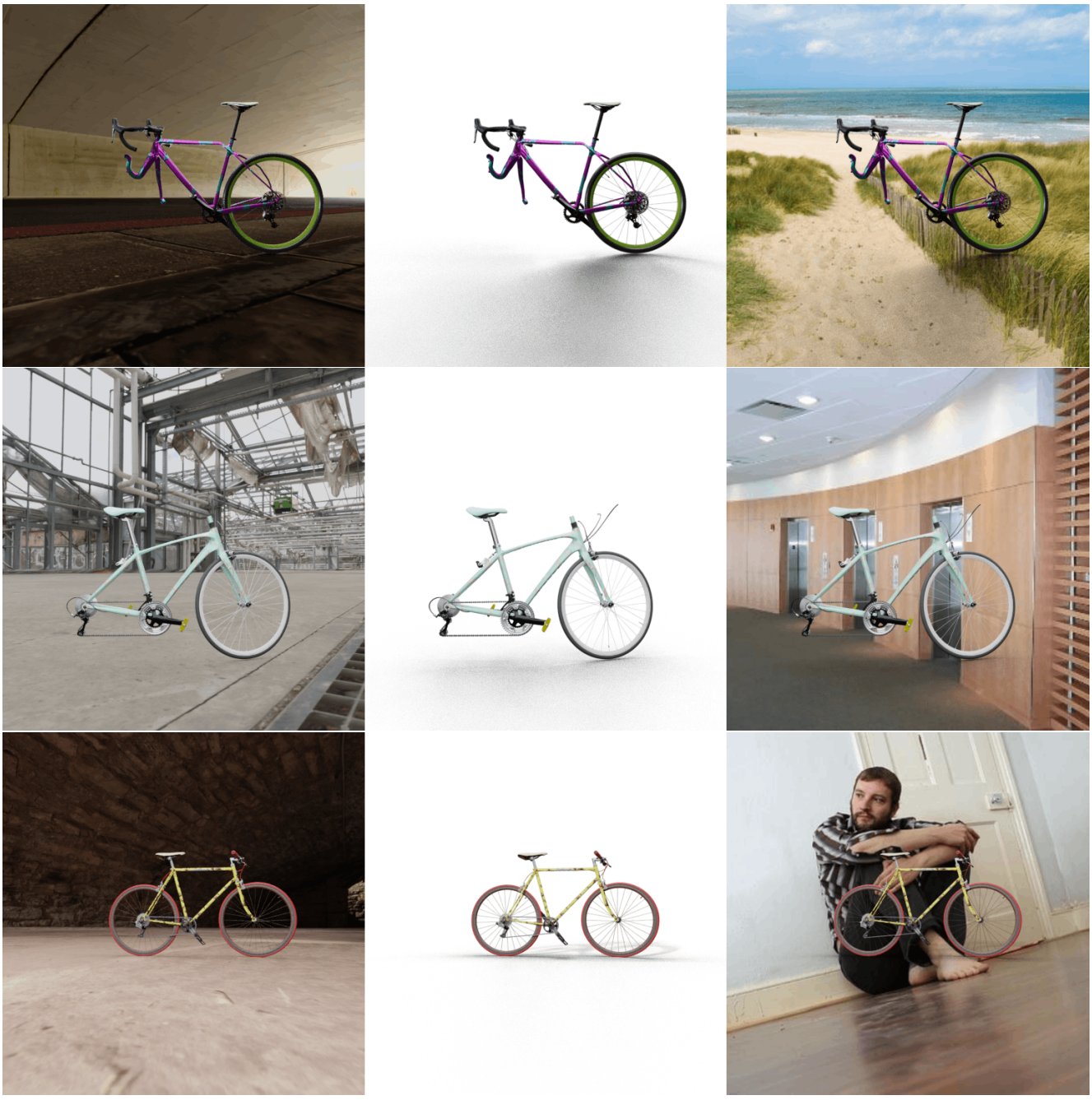
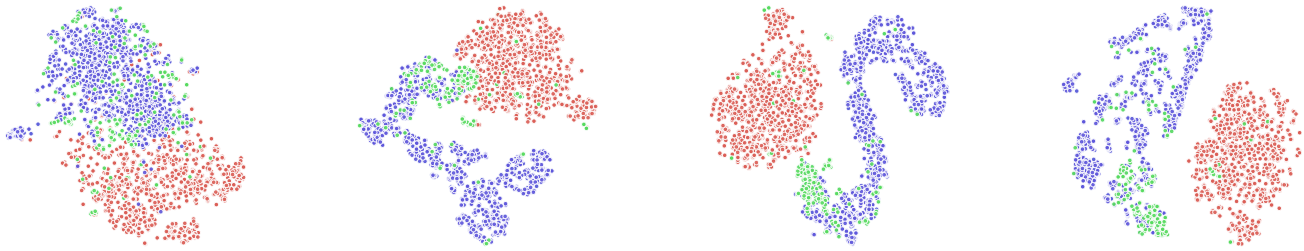


Figure 15: Examples of synthetic bike rendering placed against a 360 HDR background, a uniform background, and random scene from the Places365 dataset.

C.3. Additional explainability and t-SNE plot

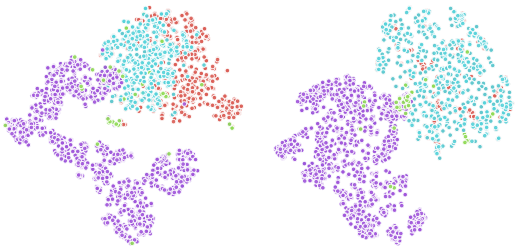
The t-SNE plots of the [*cls*] token extracted from the backbone (Figure 16) show partial overlap between the real and synthetic domains, and highlight how real images from various sources yield very different distributions.



(a) Baseline (b) BL + Real (c) BL + R + DANN (d) BL + R + PADA

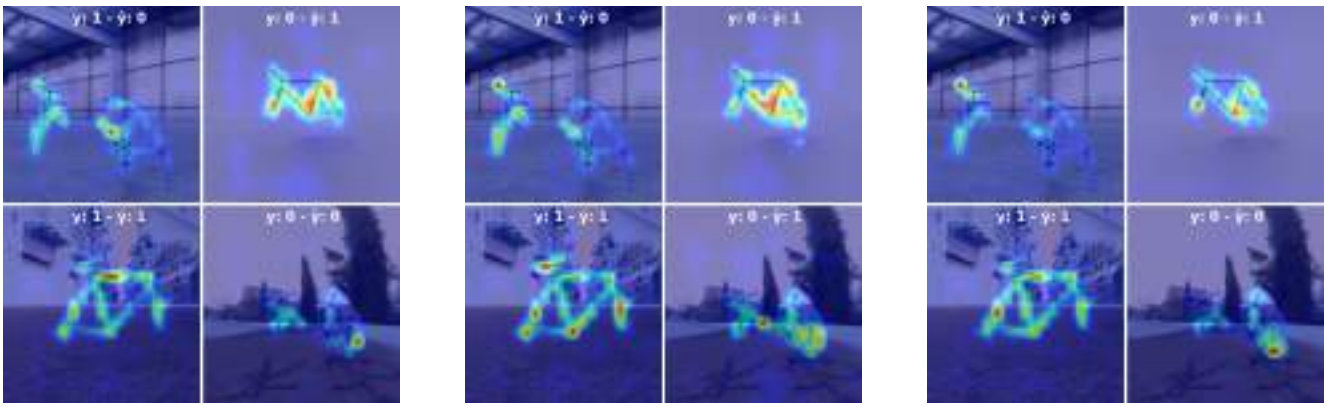
Figure 16: t-SNE plot of the [*cls*] token extracted from the DD branch under different training regimes. • DelftBikes (real) • Web scraping (real) • BBBicycles (synth)

The t-SNE plots in Figure 17 illustrate the distribution of the [*cls*] tokens from the DD branch and the backbone. By comparing synthetic damaged (red) and undamaged (cyan) examples, it can be seen how the backbone captures features related to the bike model and invariant to the presence of damage, whereas the DD branch clearly distinguishes damaged vs. non-damaged bikes.



(a) Damage branch (b) Backbone

Figure 17: t-SNE plot of the [*cls*] token extracted from DD branch (a) and backbone (b) for damaged and non-damaged bike instances (BL + Real setting). • Synthetic no damage • Synthetic damaged • Real no damage • Real damaged



(a) BL + Real (b) BL + Real + DANN (c) BL + Real + PADA

Figure 18: Attention maps of TransReI3D under different training regimes. Values for Bent frame labels (y) and predictions ($\hat{y}$) are superimposed on the image.