

Learning to Detect Label Errors by Making Them: A Method for Segmentation and Object Detection Datasets

1st Sarina Penquitt
Department of Mathematics
University of Wuppertal
 Wuppertal, Germany
 penquitt@uni-wuppertal.de

2nd Tobias Riedlinger
Department of Mathematics
Technical University of Berlin
 Berlin, Germany
 riedlinger@math.tu-berlin.de

3rd Timo Heller
Department of Mathematics
University of Wuppertal
 Wuppertal, Germany
 timo.heller@uni-wuppertal.de

4th Markus Reischl
Institute for Automation and Applied Informatics
Karlsruhe Institute of Technology (KIT)
 Karlsruhe, Germany
 markus.reischl@kit.edu

5th Matthias Rottmann
Institute of Computer Science
Osnabrück University
 Osnabrück, Germany
 matthias.rottman@uos.de

Abstract—Recently, detection of label errors and improvement of label quality in datasets for supervised learning tasks has become an increasingly important goal in both research and industry. The consequences of incorrectly annotated data include reduced model performance, biased benchmark results, and lower overall accuracy. Current state-of-the-art label error detection methods often focus on a single computer vision task and, consequently, a specific type of dataset, containing, for example, either bounding boxes or pixel-wise annotations. Furthermore, previous methods are not learning-based. In this work, we overcome this research gap. We present a unified method for detecting label errors in object detection, semantic segmentation, and instance segmentation datasets. In a nutshell, our approach – learning to detect label errors by making them – works as follows: we inject different kinds of label errors into the ground truth. Then, the detection of label errors, across all mentioned primary tasks, is framed as an instance segmentation problem based on a composite input. In our experiments, we compare the label error detection performance of our method with various baselines and state-of-the-art approaches of each task’s domain on simulated label errors across multiple tasks, datasets, and base models. This is complemented by a generalization study on real-world label errors. Additionally, we release 459 real label errors identified in the Cityscapes dataset and provide a benchmark for real label error detection in Cityscapes.

I. INTRODUCTION

Deep learning thrives on data: the more complex the task, the more data is required. In computer vision, larger training datasets consistently improve model performance [1], driving demand for large-scale, high-quality annotations. However, creating such datasets, especially for tasks like object detection or semantic segmentation, is expensive and labor-intensive [2]–[7]. The more tedious the annotation process, the higher likelier is human error [3], [4], [8]–[10]. Despite this, large amounts of high-quality labeled data remain the gold standard in critical domains like autonomous driving or

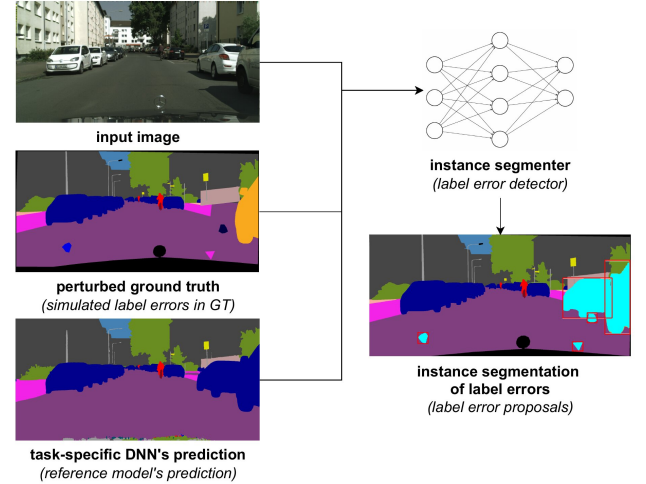


Fig. 1: Learning task of our label error method exemplary for Cityscapes in semantic segmentation.

medical imaging [2], [3], [11]. Accordingly, recent research has increasingly focused on label error detection in image classification [4], [8], [12], object detection [13]–[15], and semantic segmentation [9], [16], [17]. Label errors can substantially degrade model performance [12], [18]–[21], causing poor generalization [10], [14], biased predictions [22], and unstable benchmarks [23]. While minor localization noise may be tolerable, missing or wrongly labeled objects can seriously disrupt learning [22]. For instance, larger models are more likely to memorize noisy data, leading to worse predictions [23]. In [14] a network for localization label refinement

was introduced, showing a performance drop in state-of-the-art object detectors trained with noisy labels. A semi-automated label error detection system could thus dramatically reduce the manual review burden and associated costs. Designing such systems is challenging due to diverse annotation formats, the absence of ground truth for error locations, and intrinsic ambiguity caused by aleatoric uncertainty. To our knowledge, no existing method detects label errors across object detection, semantic segmentation, and instance segmentation. In this paper, we propose a unified framework (see fig. 1) that detects three types of label errors, i.e., missing annotations, wrong classes, and spurious (invented) labels, across multiple datasets and dataset types: Pascal VOC [24], COCO [25], Cityscapes [26], ADE20K [27], and LIVECell [28]. We simulate label noise using a perturbation algorithm and train an instance segmentation model to detect these errors. As input, this instance segmenter receives an original image, the perturbed ground truth (GT) and for simplification more reference information to be specified later on. The predictions of our instance segmenter serve as potential label error proposals.

We complement our methodological contribution by a benchmark for label error detection using simulated and real label errors and show that our approach outperforms existing state-of-the-art methods, also in detecting real label errors. Finally, we release a revised version of the Cityscapes ground truth, informed by our method. We contribute:

- We propose the first unified method to detect label errors across object detection, semantic segmentation, and instance segmentation. Furthermore, we close the research gap of *learning* the detection of label errors.
- Inspired by typical real label errors, we simulate three types of label errors by randomly perturbing the ground truth of a given dataset and learn their detection. In a broad numerical study, we evaluate our method in terms of simulated errors in PascalVOC, COCO, Cityscapes, ADE20K and LIVECell, as well as real errors in Cityscapes. We provide multiple ablation studies, e.g., on the composition of the input to our label error detection method and perturbation frequency and rate.
- Finally, we introduce a benchmark for label error detection across diverse vision tasks and complement it with a revised version of the Cityscapes ground truth, where we found 459 label errors.

The code is available under GitHub.

II. RELATED WORK

In this section, we provide an overview of existing label error detection methods for the computer vision tasks image classification, object detection, semantic segmentation, and instance segmentation. These methods differ in terms of their objectives and approach. The effects of label errors should not be underestimated as pointed out by [29]. They provide an annotation tool that simulates label noise in instance segmentation datasets and demonstrate that the performance of instance segmentation models suffers when label noise is present in the training data. To reduce the influence of label

noise in training, [30] introduced a method for training an instance segmentation model, using different losses such that the model becomes robust under noisy labels. Some works jointly consider the cost of labeling and reviewing, aiming at reducing overall annotation cost. In [31] a basic label error detection method is combined with active learning and a trade-off policy between querying new labels and reviewing already available ones. This approach maximizes model performance while minimizing annotation costs. The same goal is pursued by [32], designing a method that suppresses mislabeling between foreground and background, as well as noise from mislabeled instances. In [14] a localization label refinement network is developed to reduce label localization noise in object detection datasets and thus improve the performance of the object detector that suffers from this type of noise. The reduction in labeling costs is closely linked to the correction of label errors. In [17] a method is proposed for detecting label errors and actively correcting them in semantic segmentation datasets. The authors of [33] propose a method for the automated inspection of biomedical instance segmentation datasets with respect to noisy annotations. Domain experts can use the tool to review their annotated dataset and subsequently improve the quality of the annotations using the integrated annotation tool. There are various methods for label error detection, which take differing methodological approaches. One line of work focuses on assigning label quality scores to dataset annotations. These scores indicate the quality of the annotation of an image and induces a priority ranking for a potential review process. In the field of label error detection scoring methods, an empirical study was published by [34] that focuses on classification datasets. In [15] the framework *ObjectLab* estimates which images in object detection datasets are correctly annotated by assigning a label quality score. In [9], different quality scores for semantic segmentation datasets are studied, such as the proportion of correctly classified pixels, using the model’s predicted probabilities or the softmax score. Another approach is to consider the loss of a neural network in the decision about mislabeled data. In [35] a label error detection algorithm was developed that applies clustering algorithms to the training loss distribution for the distinction of clean and noisy labels in image classification datasets. For object detection datasets, in [13] label errors are detected by examination of instance-wise object detection losses. The use of uncertainty quantification is also widespread in image classification, object detection, as well as in semantic and instance segmentation. For image classification datasets, so-called confident learning for label noise estimation and label error detection was introduced in [12]. [8] developed more sophisticated model uncertainty measures to detect label errors and to reflect model’s predictive uncertainty accurately. The authors of [36], [37] presents a post-processing method that uses uncertainty quantification to detect label errors in LiDAR object detection datasets, by estimating the prediction quality of a LiDAR object detector. In [16] a method for semantic segmentation datasets was developed that detects label errors based on connected components and uncertainty quantification. In [38] the authors

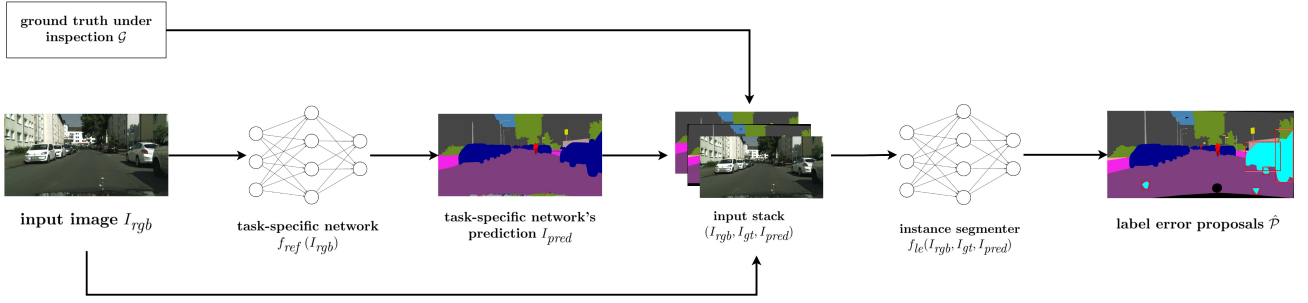


Fig. 2: Visualization of our label error detection method.

built on [16] and integrate additionally trustworthiness and hardness indices for the detection of incorrect labels in large Earth observation semantic segmentation datasets.

As opposed to all previously mentioned works, our contribution is a learning based method. Instead of focusing on a single type of annotation, we apply our method to object detection, semantic and instance segmentation annotations.

III. LABEL ERROR DETECTION METHOD

In this section, we introduce our method for detecting label errors in object detection, semantic segmentation and instance segmentation datasets. We proceed as follows: First, we present the label error detection model, followed by its training procedure. Thereafter, we describe our label error simulation and perturbation algorithm, before finally explaining how we evaluate label error detection methods.

Label Error Detection Model. Our label error detection model receives an input image I_{rgb} as well as a corresponding “ground truth” $\mathcal{G} = \{p_1, \dots, p_S\}$, $S \in \mathbb{N}$. Each p_i is comprised of a polygon representing an object (potentially contained in I_{rgb}) and a class label c_i for all $i = 1, \dots, S$. With I_{gt} we denote an image or mask representation of \mathcal{G} . Suppose that we have a label error ground truth (LEGT) $\mathcal{G}_{legt} = \{p_1^{legt}, \dots, p_M^{legt}\}$, $M \in \mathbb{N}$, with p_i^{legt} defined analogously to p_i , where \mathcal{G}_{legt} describes the set of all incorrectly annotated objects in \mathcal{G} . The goal is that the label error detector (here an instance segmentation model) identifies label errors in \mathcal{G} . As output our model provides an instance segmentation $\hat{\mathcal{P}}$ of possible label errors. Figure 2 provides an overview of our model. For learning label error detection in a given ground truth \mathcal{G} , the knowledge acquired by deep neural networks (DNNs) that learned the original task (associated with \mathcal{G}) of the given dataset are utilized. We call such a DNN a *reference DNN* f_{ref} , that provides a *reference prediction* $I_{pred} = f_{ref}(I_{rgb})$. If a given dataset’s annotation consists of semantic segmentation masks, the reference DNN predicts such masks as well, analogously for object detection and instance segmentation. The input image I_{rgb} , the ground truth mask I_{gt} and the reference prediction I_{pred} stacked together serve as input for the downstream instance segmenter, that learns the label error detection. That is, $f_{le} : (I_{rgb}, I_{gt}, I_{pred}) \mapsto \hat{\mathcal{P}}$, where $\hat{\mathcal{P}}$ is supposed to estimate \mathcal{G}_{legt} . Depending on the dataset, we

introduce a fourth input which is an additional difference mask I_{diff} explicitly containing differences between I_{gt} and I_{pred} .

Learning to Detect Label Errors. We now specify the learning process, that is summarized in Figure 3. We start with a given input image I_{rgb} with its corresponding ground truth \mathcal{G}_{org} and ground truth mask I_{org} . Then we apply a label perturbation algorithm that randomly perturbs labels of \mathcal{G}_{org} at a chosen rate q . Therefrom, we obtain a perturbed ground truth \mathcal{G}_{pert} (taking the role of the ground truth under inspection \mathcal{G}), a perturbed ground truth mask I_{pert} and a corresponding label error ground truth \mathcal{G}_{legt} . As we algorithmically perturb labels, we have the knowledge which of our labels deviate from the original ones in \mathcal{G}_{org} and this information is stored in \mathcal{G}_{legt} . The label perturbation algorithm is introduced in the next paragraph. Afterwards, a triplet $(I_{rgb}, \mathcal{G}_{pert}, \mathcal{G}_{legt})$ is given for each sample in the chosen dataset and this is used to train f_{le} using typical off-the-shelf instance segmentors. Losses \mathcal{L} are then computed as functions of $\hat{\mathcal{P}}$ and \mathcal{G}_{legt} .

Label Perturbation Algorithm. We developed a perturbation algorithm for the label error simulation. For an arbitrary dataset, let I_{rgb} be an RGB image with ground truth annotations $\mathcal{G}_{org} = \{p_1^{org}, \dots, p_N^{org}\}$, $N \in \mathbb{N}$. We assume that these annotations are clean and accurate. Therein, each p_i^{org} is comprised of a polygon covering a ground truth object and a class label c_i^{org} for all $i = 1, \dots, N$. Such a polygon can be a bounding box or a segment. Drawing all p_i^{org} onto the image plane, e.g., via a coloring scheme, yields a ground truth mask which we refer to as I_{org} . In our perturbation algorithm, we randomly generate a perturbed ground truth by modifying \mathcal{G}_{org} . The algorithm iterates over each ground truth polygon p_i^{org} , for $i = 1, \dots, N$, generating a perturbed ground truth $\mathcal{G}_{pert} = \{p_1^{pert}, \dots, p_{N'}^{pert}\}$, $N' \in \mathbb{N}$, as well as a label error ground truth (LEGT) $\mathcal{G}_{legt} = \{p_1^{legt}, \dots, p_M^{legt}\}$, $M \in \mathbb{N}$. Within these two sets, p_i^{pert} and p_i^{legt} are defined analogously to p_i^{org} . Corresponding to the perturbed ground truth \mathcal{G}_{pert} (the original ground truth consisting of simulated label errors) we define the perturbed ground truth mask as I_{pert} . The label error ground truth \mathcal{G}_{legt} includes all simulated label error objects. To perform data augmentation, we apply the label perturbation algorithm multiple times to a given dataset, obtaining a number of copies of the dataset with different label errors. We denote the number of copies by t . The algorithms initially starts with $\mathcal{G}_{pert} = \mathcal{G}_{legt} = \emptyset$. In each iteration i we perform a

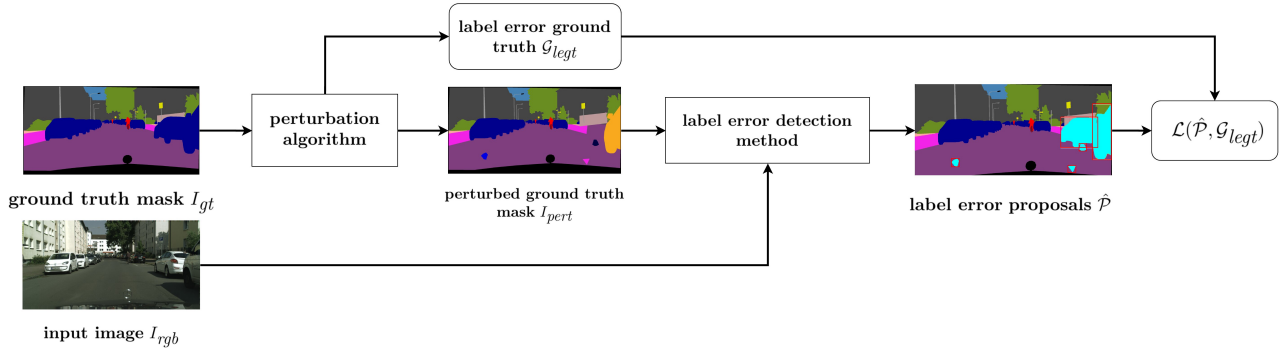


Fig. 3: Visualization of the learning process in our label error detection method.

Bernoulli experiment such that with probability q we add a perturbation to the dataset and with probability $1-q$ we do not. If we add a perturbation, then we uniformly choose a random perturbation from three different types of label errors. *Drop*: p_i^{org} is deleted and is not part of the perturbed ground truth \mathcal{G}_{pert} , but therefore added to \mathcal{G}_{legt} . *Flip*: the class affiliation of p_i^{org} is altered (drawing uniformly a label from the remaining ones), resulting in a p_i^{pert} with altered $c_i^{pert} \neq c_i^{org}$, which is added to \mathcal{G}_{pert} and \mathcal{G}_{legt} . *Spawn*: p_i^{org} is also part of the perturbed ground truth \mathcal{G}_{pert} , i.e., $p_i^{org} = p_i^{pert}$. However, we add another spawned ground truth polygon to \mathcal{G}_{pert} and \mathcal{G}_{legt} , which does not represent any object in I_{rgb} and is generated randomly. The above description is for a single image-ground-truth pair. Performing the iteration for all image-ground-truth pairs of a given dataset yields an perturbed ground truth for the whole dataset.

Evaluation. We evaluate the predictions of our label error detector by means of label error ground truth. We have two cases: evaluation on 1) simulated label errors $\hat{\mathcal{P}}_{pert} = f_{le}(I_{rgb}, I_{pert}, I_{pred})$ and 2) real label errors $\hat{\mathcal{P}}_{real} = f_{le}(I_{rgb}, I_{gt}, I_{pred})$. In the first case, we compare $\hat{\mathcal{P}}_{pert}$ with $\mathcal{G}_{legt}^{pert}$ on test data not used for training f_{le} , where the label error ground truth $\mathcal{G}_{legt}^{pert}$ is obtained by the perturbation algorithm. In the second case, $\mathcal{G}_{legt}^{real}$ is provided by human review of the given dataset’s annotations and we compare predictions $\hat{\mathcal{P}}_{real}$ of our method (wherein the labels under inspection are the original ones from a given dataset) with the dataset’s annotations $\mathcal{G}_{legt}^{real}$. Following the usual approach to image detection and segmentation methods, our aim is to identify as many label errors as possible from the label error ground truth, while minimizing false predictions of label errors. For each prediction of a given label error detection method, we decide whether it is a label error (true positive, TP_{le}) or no label error (false positive, FP_{le}). A prediction is a TP_{le} , if the Intersection over Union (IoU) between the prediction under consideration and a label error (from the label error ground truth) is greater than or equal to a chosen threshold $\tau_{iou} \in (0, 1]$. Otherwise it is an FP_{le} . Label errors from the label error ground truth that do not receive a match during this procedure are referred to as false negatives (FN_{le}). In both of the above cases (inspecting

methods in simulated and real label errors), the evaluation of $\hat{\mathcal{P}}$ via \mathcal{G}_{legt} is performed using evaluation protocol introduced by the COCO benchmark [25]. We consider the metrics Average Precision (AP), Precision, Recall and F1-Score. Here the AP is defined as the precision averaged over 10 Intersection over Union (IoU) values from 0.5 to 0.95 [25].

IV. NUMERICAL EXPERIMENTS

In this section, we apply our label error detection method to the semantic segmentation datasets Cityscapes and ADE20K, the object detection datasets PascalVOC and COCO, and the instance segmentation dataset LIVECell. We compare our results with existing state-of-the-art label error detection methods. After that, we present the results of ablation studies on the Cityscapes dataset and examine the detection of real label errors.

Implementation Details. We implemented our method and benchmark in Detectron2 library [39]. We used the instance segmentation model Cascade Mask-RCNN [40] as a label error detector, training it on only one class. This means that, even though we simulated three different types of label errors (*drop*, *flip* and *spawn*), all of these were included in the training as class *error*. In the semantic segmentation datasets, we limited the perturbation by setting minimum and maximum sizes for the objects being perturbed. In Cityscapes, objects required an area of at least 800 pixels, and in LIVECell and ADE20K, 200 pixels. The maximum size was set at 10,000 pixels. We introduced these constraints as there are many small objects in Cityscapes, however, our aim is to create a training dataset that contains visible but not unrealistically large label errors for training. For LIVECell and ADE20K, we opted for a smaller segment size due to the smaller image sizes. The creation of ground truth and perturbed ground truth masks differs in the respective computer vision task. In the semantic segmentation datasets, we use existing color coding to assign a color to each class. For instance segmentation, we randomly generate a color for each object to distinguish the individual segments. For object detection, we generate an HSV color model to reflect overlapping regions between two bounding boxes in the color map. This enables us to display each object in full without losing any information about it. An overview of the

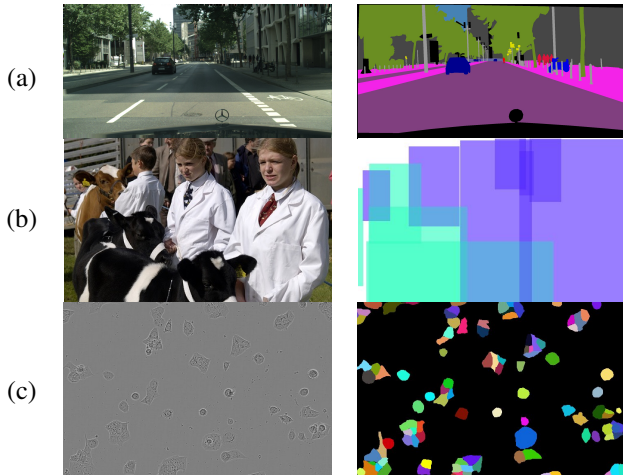


Fig. 4: Overview of ground truth masks for semantic segmentation (a), object detection (b) and instance segmentation (c) datasets.

different types of masks can be found in figure 4. As our label error detector is an instance segmenter, label errors in object detection datasets are learned and predicted in rectangular bounding box shape.

Baselines. The *naive baseline* investigates the predictions of the reference DNN. Those predictions that are false positive w.r.t. to ground truth \mathcal{G} under inspection serve as label error proposals, since they represent differences between the ground truth under inspection and the reference prediction. A prediction is false positive if it overlaps with an object of the same class in the ground truth by less than 0.3 IoU. For object detection and instance segmentation we also consider a *score baseline*, which differs from the naive baseline by taking into account the predicted score of the reference DNN, which is used for thresholding in the evaluation. For object detection, we also consider the *Loss inspection* method by [13]. This method is based on object detection loss at the instance level. The method depends on the architecture in which it was implemented, with the specific architecture determining the way in which it functions. Thus, we only apply it to Cascade R-CNN predictions as in the original work. The final baseline for semantic segmentation datasets is the DNN + uncertainty quantification (*DNN+UQ*) method developed by [16]. This method detects label errors based on connected components from a predicted semantic segmentation and UQ on component-level. A minimum object size of 100 pixels has been set for this method. In the following comparisons, where we used this baseline, we made sure that all predictions included were also at least 100 pixels in size in order to make the methods more comparable.

Results for Simulated Label Errors. In this section, we present results for five datasets across the tasks semantic and instance segmentation (table I) and object detection (table II). With one exception, our method outperforms state-of-the-art baselines across all datasets and evaluation metrics. The only

exception is the precision value on the COCO dataset, where the score baseline achieves slightly superior results. However, in that case our method is clearly superior in terms of recall, F1 and AP. Taking a closer look, we see that our method sometimes overproduces false positives. Precision gives an indication of how much reviewing work needs to be invested to find a label error. A precision of 50% means that every second proposal of our method is indeed a label error. The more important metric is probably recall, which is tightly related to the achievable annotation quality by any of the considered methods. A recall of 90% means that 90% of the label errors that we injected by our label perturbation algorithm can be detected. Note that TP, FP and FN metrics can still be traded. E.g. to achieve a higher recall, we can lower the score threshold, which of course increases the amount of reviewing work or in other words decreases the precision. In some cases we outperform the previous state-of-the-art by enormous margins. E.g. in all segmentation tasks, the recall of our method exceeds the one of the DNN+UQ baseline by 38–62 percent points. Note, however, that the DNN+UQ baseline is not able to detect spawns, which means a default offset of 33 percent points between DNN+UQ and our method. Even when taking this offset into account, we outperform the DNN+UQ baseline. We note that learning on a distribution that underwent the same perturbation algorithm as the test distribution might give our distribution a conceptual edge over the competition. That is why we evaluate on real label errors in the Cityscapes dataset in the next section.

Ablation Studies. In this section, we examine the performance results of trained label error detectors on simulated data and conduct ablation experiments on the Cityscapes dataset. To this end, we selected various training parameters, including altering the perturbation rate and the reference DNNs for the predicted masks, and varying the input, as broken down in table III. We performed evaluations of each model and baseline on the same perturbed validation dataset, created with a perturbation rate of $q = 0.2$, inducing 3,106 artificial label errors. Using our method, we outperformed the baselines in every training configuration and metric, except when using Mask-RCNN [41] as label error detector. We argue that Cascade Mask-RCNN achieves higher precision, whereas the Mask-RCNN prioritizes higher recall. This is also reflected in the higher number of true positives (TPs) of the Mask-RCNN compared to the Cascade Mask-RCNN. The results also demonstrate that the choice of the reference DNN can influence the performance of the label error detector. The DeepLabV3+ [42] network is often characterized by its high precision, which is shown in the results. Label error detection with DeepLabV3+ predictions achieves almost 10% higher precision than segformer [43] when used as a reference network. Segformer on the other hand, has superior recall in direct comparison to the other two reference networks, achieving highest recall marginally in front of VLTseg [44]. When the number of copies t of the perturbed dataset in the training is varied, two things become apparent. Firstly, the number of false positives increases, resulting in lower precision and

TABLE I: Performance of the trained label error detector on perturbed datasets on Cityscapes ($q = 0.2, t = 10$), ADE20K ($q = 0.2, t = 2$) and LIVECell ($q = 0.1, t = 1$). For evaluation we chose a score threshold of 0.5 and an IoU threshold of 0.1.

	AP	TPs	FPS	FNs	Precision	Recall	F1-Score
<i>Cityscapes - Semantic Segmentation</i>							
naive baseline	-	1336	9554	1770	12.27	43.01	10.09
DNN+UQ	4.47	1029	4640	2077	18.15	33.13	23.45
$f_{le}(I_{rgb}, I_{pert}, I_{pred}, I_{diff})$	61.74	2975	4423	131	40.21	95.78	56.65
<i>ADE20K - Semantic Segmentation</i>							
naive baseline	-	1967	13,019	2169	13.13	47.56	20.57
DNN+UQ	3.42	1292	4029	2844	24.28	31.24	27.32
$f_{le}(I_{rgb}, I_{pert}, I_{pred}, I_{diff})$	46.58	3405	6223	731	35.37	82.33	49.48
<i>LIVECell - Instance Segmentation</i>							
naive baseline	-	21,888	91,254	22,229	19.35	49.61	27.84
score baseline	11.79	15,138	16,382	28,979	48.03	34.31	40.03
$f_{le}(I_{rgb}, I_{pert}, I_{pred})$	52.19	32,196	4657	11,921	87.36	72.98	79.53

TABLE II: Performance of the trained label error detector on perturbed datasets on PascalVOC ($q = 0.2, t = 10$) and COCO ($q = 0.2, t = 2$). For evaluation we chose a score threshold of 0.5 and an IoU threshold of 0.1.

	AP	TPs	FPS	FNs	Precision	Recall	F1-Score
<i>PascalVOC - Object Detection</i>							
naive baseline	-	1998	8251	968	19.49	67.36	30.24
score baseline	24.23	1545	2547	1421	37.76	52.09	43.78
loss inspection	1.35	720	2707	2246	21.01	24.28	22.52
$f_{le}(I_{rgb}, I_{pert}, I_{pred}, I_{diff})$	59.56	2533	3439	433	42.41	85.40	56.68
<i>COCO - Object Detection</i>							
naive baseline	-	4744	15,252	2763	23.72	63.19	34.50
score baseline	29.13	3272	1962	4235	62.51	43.59	51.36
loss inspection	1.39	2667	10,531	4840	20.21	35.53	25.76
$f_{le}(I_{rgb}, I_{pert}, I_{pred}, I_{diff})$	55.82	5722	5465	1785	51.15	76.22	61.22

F1-Score. However, by perturbing the dataset several times and thus increasing diversity of the training dataset (at fixed perturbation rate $q = 0.2$), we can achieve a higher AP and recall. An increasing perturbation rate q with constant $t = 5$ has a positive effect on all metrics. Although the number of false positives grows, the simultaneous increase in TPs means that precision, recall and the F1-Score are still improved. The performance results are surprising when we change the composition of the input tensor used to train the label error detector. In fact, we achieve the best results when we do not use the RGB input image. Directly comparing training $f_{le}(I_{rgb}, I_{pert}, I_{pred})$ and $f_{le}(I_{pert}, I_{pred}, I_{diff})$ shows that the label error detector performs better when trained with a difference mask than with an RGB image, even if the differences are marginal. In summary, there are essentially no significant differences in performance, and all trained label detectors perform well regardless of training parameters. However, the influence of varying input and choice of reference network offer opportunities for finetuning to a specific use case.

Results for Real Label Errors. The evaluation of our label error method in detecting real label errors in Cityscapes, ADE20K, PascalVOC, COCO and LiveCell is the focus of the final section on numerical experiments. First, we created a label error ground truth of real label errors in the Cityscapes validation dataset by manually reviewing label error proposals from the DNN+UQ method [16] and our own method. We detected 228 label errors using DNN+UQ, and additional 231 using our method. At this point, we would like to emphasize that this is only a selection of real label errors in the Cityscapes validation dataset and is intended only as a

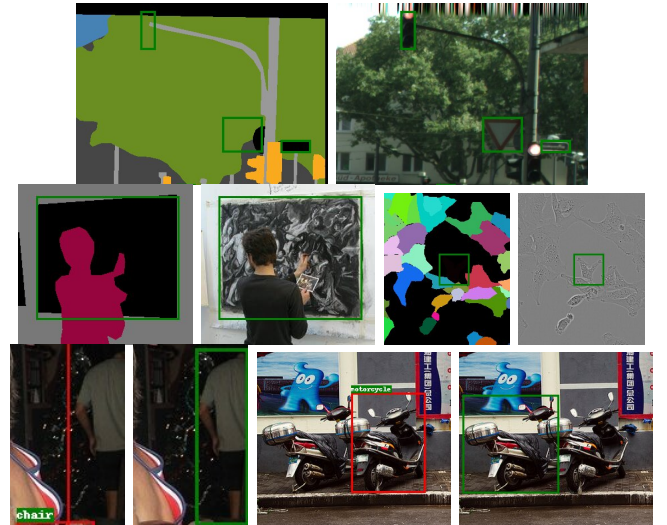


Fig. 5: Examples of found real label errors in Cityscapes, ADE20K, LIVECell, PascalVOC und COCO (left to right). The ground truth and RGB images are shown alternately. The segmentation masks and the RGB images with red boxes show the GT annotations. In green are the label error proposals of our trained label error detector.

basis for evaluation. We do not claim that all existing label errors have been detected. It is possible that more real label errors may be found when manually reviewing additional label error proposals. To get proposals for possible real label errors, we change the input from the trained label error detector

TABLE III: Ablation studies and performance comparison on different training parameters of label error detector on perturbed label errors. Evaluation of all experiments on the same perturbed validation dataset with perturbation rate $q = 0.2$.

	AP	TPs	FPs	FNs	Precision	Recall	F1-Score
naive baseline	-	1336	9554	1770	12.27	43.01	10.09
DNN+UQ	4.47	1029	4640	2077	18.15	33.13	23.45
t	<i>perturbation frequency</i>						
1	56.61	2869	3638	237	44.09	92.37	59.69
5	61.28	2969	4326	137	40.70	95.59	57.09
10	61.74	2975	4423	131	40.21	95.78	56.65
q	<i>perturbation rate</i>						
0.05	54.80	2534	4018	572	38.68	81.58	52.47
0.1	58.82	2920	4451	186	39.61	94.01	55.74
0.2	61.28	2969	4326	137	40.70	95.59	57.09
f_{ref}	<i>semantic segmentation reference network</i>						
segformer	61.74	2975	4423	131	40.21	95.78	56.65
DeepLabV3+	64.20	2950	3000	156	49.58	94.98	65.15
vltseg	62.65	2973	3397	133	46.67	95.72	62.75
f_{le}	<i>instance segmentation network (label error predictor)</i>						
Cascade Mask-RCNN	61.74	2975	4423	131	40.21	95.78	56.65
Mask-RCNN	61.64	3019	5911	87	33.81	97.20	50.17
<i>training inputs</i>							
$f_{le}(I_{rgb}, I_{pert}, I_{pred}, I_{diff})$	61.74	2975	4423	131	40.21	95.78	56.65
$f_{le}(I_{rgb}, I_{pert}, I_{pred})$	61.60	2972	4302	134	40.86	95.69	57.26
$f_{le}(I_{pert}, I_{pred}, I_{diff})$	62.89	2991	4285	115	41.11	96.30	57.62

TABLE IV: Evaluation on real label errors in Cityscapes. Our label error detector was trained on $t = 10$ copies of Cityscapes with perturbation rate $q = 0.2$.

	AP	TPs	FPs	FNs	Precision	Recall	F1-Score
naive baseline	-	294	8462	165	3.36	64.05	6.38
DNN+UQ	4.13	274	4038	185	6.35	59.69	11.49
$f_{le}(I_{rgb}, I_{org}, I_{pred}, I_{diff})$	5.46	323	4568	136	6.60	70.37	12.07

TABLE V: Found real label errors predicted by our label error detector with a score threshold of 0.75 of 200 randomly chosen images of validation dataset.

Dataset	#Label Errors	Precision	#Predictions
ADE20K	21	5.57	377
LIVECell	95	29.78	319
PascalVOC	13	18.57	70
COCO	35	41.12	85

from $f_{le}(I_{rgb}, I_{pert}, I_{pred}, I_{diff})$ to $f_{le}(I_{rgb}, I_{gt}, I_{pred}, I_{diff})$, where the difference mask I_{diff} in combination with I_{gt} shows now the differences between the ground truth mask I_{gt} and the prediction mask I_{pred} . In table IV the results of our label error method in comparison to the semantic segmentation baseline *naive baseline* and *DNN+UQ* are presented. Despite the higher number of false positives in our method, we can still outperform the baselines in particular in precision and recall. Real label errors were also found in the ADE20K, LIVECell, PascalVOC and COCO datasets. Examples are presented in figure 5. As a study, we randomly selected 200 images from validation set of each dataset. We then used our label error detector to predict label error proposals with a score threshold of 0.75, and counted how many of these were real ones. The results are shown in table V. Most of the label errors were found in the COCO and LIVECell datasets, whereas the PascalVOC dataset was well labeled from the very beginning. The ADE20K dataset was difficult to review due to its 150 different classes. However, at least 21 real label errors were

found, although there could possibly be more.

V. CONCLUSION

In this work, we proposed the first method to detect label errors in object detection, semantic and instance segmentation datasets by learning detection from simulated label errors. For the label error simulation, we developed a perturbation algorithm to simulate three different label error types. In numerical experiments on simulated and real label errors, our label error detector method outperforms common baselines and existing state-of-the-art label error detection methods. Moreover, we introduce a benchmark for label error detection with an updated version of the Cityscapes ground truth.

ACKNOWLEDGEMENTS

We thank Miriam Ackermann and Anna Hövermann for the label error review and we thank Marcel Schilling for discussion and useful advice. S.P. and M.R. acknowledge support by the German Federal Ministry of Research, Technology and Space (BMFTR) within the junior research group project “UnrEAL” (grant no. 01IS22069).

REFERENCES

- [1] C. Sun, A. Shrivastava, S. Singh, and A. Gupta, “Revisiting Unreasonable Effectiveness of Data in Deep Learning Era,” in *2017 IEEE International Conference on Computer Vision (ICCV)*. Los Alamitos, CA, USA: IEEE Computer Society, Oct. 2017, pp. 843–852. [Online]. Available: <https://doi.ieeecomputersociety.org/10.1109/ICCV.2017.97>

- [2] A. S. Lundervold and A. Lundervold, "An overview of deep learning in medical imaging focusing on mri," *Zeitschrift für Medizinische Physik*, vol. 29, no. 2, pp. 102–127, 2019, special Issue: Deep Learning in Medical Physics. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0939388918301181>
- [3] D. Feng, C. Haase-Schütz, L. Rosenbaum, H. Hertlein, C. Gläser, F. Timm, W. Wiesbeck, and K. Dietmayer, "Deep multi-modal object detection and semantic segmentation for autonomous driving: Datasets, methods, and challenges," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 3, pp. 1341–1360, 2021.
- [4] N. M. Muller and K. Markert, "Identifying mislabeled instances in classification datasets," in *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, Jul. 2019, p. 1–8. [Online]. Available: <http://dx.doi.org/10.1109/IJCNN.2019.8851920>
- [5] M. Haakman, L. Cruz, H. Huijgens, and A. van Deursen, "Ai lifecycle models need to be revised: An exploratory study in fintech," *Empirical Softw. Engg.*, vol. 26, no. 5, Sep. 2021. [Online]. Available: <https://doi.org/10.1007/s10664-021-09993-1>
- [6] F. A. Schmidt, "Crowdproduktion von trainingsdaten: Zur rolle von online-arbeit beim trainieren autonomer fahrzeuge," Düsseldorf, Study der Hans-Böckler-Stiftung 417, 2019. [Online]. Available: <https://hdl.handle.net/10419/201850>
- [7] N. Heller, J. Dean, and N. Papanikolopoulos, "Imperfect segmentation labels: How much do they matter?" in *CVII-STENT/LABELS@MICCAI*, 2018. [Online]. Available: <https://api.semanticscholar.org/CorpusID:48363025>
- [8] J. Jakubik, M. Vössing, M. Maskey, C. Wölflle, and G. Satzger, "Improving label error detection and elimination with uncertainty quantification," 2024. [Online]. Available: <https://arxiv.org/abs/2405.09602>
- [9] V. Lad and J. Mueller, "Estimating label quality and errors in semantic segmentation data via any model," in *ICML Workshop on Data-centric Machine Learning Research*, 2023.
- [10] G. Pleiss, T. Zhang, E. Elenberg, and K. Q. Weinberger, "Identifying mislabeled data using the area under the margin ranking," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS '20. Red Hook, NY, USA: Curran Associates Inc., 2020.
- [11] S. Kuutti, R. Bowden, Y. Jin, P. Barber, and S. Fallah, "A survey of deep learning applications to autonomous vehicle control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 2, pp. 712–733, 2021.
- [12] C. Northcutt, L. Jiang, and I. Chuang, "Confident learning: Estimating uncertainty in dataset labels," *J. Artif. Int. Res.*, vol. 70, p. 1373–1411, May 2021. [Online]. Available: <https://doi.org/10.1613/jair.1.12125>
- [13] M. Schubert, T. Riedlinger, K. Kahl, D. Kröll, S. Schoenen, S. Šegvič, and M. Rottmann, "Identifying label errors in object detection datasets by loss inspection," in *2024 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2024, pp. 4570–4579.
- [14] A. Bär, J. Uhrig, J. P. Umesh, M. Cordts, and T. Fingscheidt, "A novel benchmark for refinement of noisy localization labels in autolabeled datasets for object detection," in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2023, pp. 3851–3860.
- [15] U. Tkachenko, A. Thyagarajan, and J. Mueller, "Objectlab: Automated diagnosis of mislabeled images in object detection data," in *ICML Workshop on Data-centric Machine Learning Research*, 2023.
- [16] M. Rottmann and M. Reese, "Automated detection of label errors in semantic segmentation datasets via deep learning and uncertainty quantification," in *2023 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2023, pp. 3213–3222.
- [17] H. Kim, S. Hwang, S. Kwak, and J. Ok, "Active label correction for semantic segmentation with foundation models," in *Proceedings of the 41st International Conference on Machine Learning*, ser. ICML'24. JMLR.org, 2024.
- [18] C. Agnew, C. Eising, P. Denny, A. Scanlan, P. Van De Ven, and E. M. Grua, "Quantifying the effects of ground truth annotation quality on object detection and instance segmentation performance," *IEEE Access*, vol. 11, pp. 25 174–25 188, 2023.
- [19] Y. He, C. Zhu, J. Wang, M. Savvides, and X. Zhang, "Bounding box regression with uncertainty for accurate object detection," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.
- [20] C. Brodley and M. Friedl, "Identifying mislabeled training data," *Journal of Artificial Intelligence Research - JAIR*, vol. 11, 06 2011.
- [21] M. P. Schilling, T. Scherr, F. R. Münke, O. Neumann, M. Schutera, R. Mikut, and M. Reischl, "Automated annotator variability inspection for biomedical image segmentation," *IEEE Access*, vol. 10, pp. 2753–2765, 2022.
- [22] C. Agnew, A. Scanlan, P. Denny, E. M. Grua, P. van de Ven, and C. Eising, "Annotation quality versus quantity for object detection and instance segmentation," *IEEE Access*, vol. 12, pp. 140 958–140 977, 2024.
- [23] C. G. Northcutt, A. Athalye, and J. Mueller, "Pervasive label errors in test sets destabilize machine learning benchmarks," in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021. [Online]. Available: <https://openreview.net/forum?id=XccDXrDNLeK>
- [24] M. Everingham, L. Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *Int. J. Comput. Vision*, vol. 88, no. 2, p. 303–338, Jun. 2010. [Online]. Available: <https://doi.org/10.1007/s11263-009-0275-4>
- [25] 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 *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds. Cham: Springer International Publishing, 2014, p. 740–755.
- [26] 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 *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
- [27] B. Zhou, H. Zhao, X. Puig, T. Xiao, S. Fidler, A. Barriuso, and A. Torralba, "Semantic understanding of scenes through the ade20k dataset," *International Journal of Computer Vision*, vol. 127, no. 3, pp. 302–321, 2019.
- [28] C. Edlund, T. R. Jackson, N. Khalid, N. Bevan, T. Dale, A. Dengel, S. Ahmed, J. Trygg, and R. Sjögren, "Livecell - a large-scale dataset for label-free live cell segmentation," *Nature Methods*, vol. 18, no. 9, pp. 1038–1045, 2021. [Online]. Available: <https://doi.org/10.1038/s41592-021-01249-6>
- [29] E. Grad, M. Kimhi, L. Halika, and C. Baskin, "Benchmarking label noise in instance segmentation: Spatial noise matters," 2024. [Online]. Available: <https://arxiv.org/abs/2406.10891>
- [30] L. Yang, F. Meng, H. Li, Q. Wu, and Q. Cheng, "Learning with noisy class labels for instance segmentation," in *Computer Vision – ECCV 2020*, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds. Cham: Springer International Publishing, 2020, p. 38–53.
- [31] M. Schubert, T. Riedlinger, K. Kahl, and M. Rottmann, "Deep active learning with noisy oracle in object detection," in *Proceedings of the 19th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 2: VISAPP, INSTICC*. SciTePress, 2024, pp. 375–384.
- [32] L. Chen, Y. Fu, S. You, and H. Liu, "Hybrid supervised instance segmentation by learning label noise suppression," *Neurocomputing*, vol. 496, pp. 131–146, 2022. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0925231222005574>
- [33] M. P. Schilling, L. Klinger, U. Schumacher, S. Schmelzer, M. B. López, B. Nestler, and M. Reischl, "Ai2seg: A method and tool for ai-based annotation inspection of biomedical instance segmentation datasets," in *2023 45th Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, 2023, pp. 1–6.
- [34] J. Kuan and J. Mueller, "Model-agnostic label quality scoring to detect real-world label errors," in *ICML DataPerf Workshop*, 2022.
- [35] C. Yue and N. K. Jha, "CTRL: Clustering Training Losses for Label Error Detection," *IEEE Transactions on Artificial Intelligence*, vol. 5, no. 08, pp. 4121–4135, Aug. 2024. [Online]. Available: <https://doi.ieeecomputersociety.org/10.1109/TAI.2024.3365093>
- [36] T. Riedlinger, M. Schubert, S. Penquitt, J.-M. Kezmann, P. Colling, K. Kahl, L. Roesse-Koerner, M. Arnold, U. Zimmermann, and M. Rottmann, "Lmd: Light-weight prediction quality estimation for object detection in lidar point clouds," in *Pattern Recognition*. Cham: Springer Nature Switzerland, 2024, pp. 85–99.
- [37] —, "Lmd: Light-weight prediction quality estimation for object detection in lidar point clouds," *International Journal of Computer Vision*, vol. 133, pp. 4349–4365, 02 2025.
- [38] C. Aybar, D. Montero, G. Mateo-García, and L. Gómez-Chova, "Lessons learned from cloudsens12 dataset: Identifying incorrect annotations in cloud semantic segmentation datasets," in *IGARSS 2023 - 2023 IEEE*

International Geoscience and Remote Sensing Symposium, 2023, pp. 892–895.

- [39] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick, “Detectron2,” <https://github.com/facebookresearch/detectron2>, 2019.
- [40] Z. Cai and N. Vasconcelos, “Cascade r-cnn: High quality object detection and instance segmentation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 5, pp. 1483–1498, 2021.
- [41] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask r-cnn,” in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 2980–2988.
- [42] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, “Encoder-decoder with atrous separable convolution for semantic image segmentation,” in *Computer Vision – ECCV 2018: 15th European Conference, Munich, Germany, September 8–14, 2018, Proceedings, Part VII*. Berlin, Heidelberg: Springer-Verlag, 2018, p. 833–851. [Online]. Available: https://doi.org/10.1007/978-3-030-01234-2_49
- [43] E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez, and P. Luo, “Segformer: Simple and efficient design for semantic segmentation with transformers,” in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 12 077–12 090. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2021/file/64f1f27bf1b4ec22924fd0acb550c235-Paper.pdf
- [44] C. Hümmer, M. Schwonberg, L. Zhou, H. Cao, A. Knoll, and H. Gottschalk, “Strong but simple: A baseline for domain generalized dense perception by clip-based transfer learning,” in *ACCV*, 2024.
- [45] M. Ounsworth, “Answer on stack overflow: *Algorithm to generate random 2D polygon*,” Stack Overflow, 2014, online; ID 25276331. [Online]. Available: <https://stackoverflow.com/a/25276331>

Algorithm 1 Perturbation Algorithm Pseudocode

Input: ground truth annotations $\mathcal{G}_{org} = \{p_1^{org}, \dots, p_N^{org}\}$, perturbation rate q , polygon type k , I_{org} size (w, h) , radius range $\bar{r} = (r_1, r_2)$ with $r_1 < r_2$, spikeness $s \in (0, 1)$, irregularity $i \in (0, 1)$, maximal number vertices v_m

Output: perturbed ground truth annotations $\mathcal{G}_{pert} = \{p_1^{pert}, \dots, p_{N'}^{pert}\}$, label error ground truth $\mathcal{G}_{legt} = \{p_1^{legt}, \dots, p_M^{legt}\}$
 $\mathcal{G}_{pert} = \mathcal{G}_{legt} = \emptyset$

for $i = 1$ to N **do**
 # Bernoulli experiment
 Sample $X \sim \text{Bernoulli}(q)$
 if $X = \text{true}$ **then**
 # uniformly choose label error type
 Sample $T \sim \mathcal{U}(\{\text{drop}, \text{flip}, \text{spawn}\})$
 if $T = \text{drop}$ **then**
 $\mathcal{G}_{legt} = \mathcal{G}_{legt} \cup \{p_i^{org}\}$
 else if $T = \text{flip}$ **then**
 $p_i^{pert} = \text{ChangeClass}(p_i^{org})$
 $\mathcal{G}_{pert} = \mathcal{G}_{pert} \cup \{p_i^{pert}\}$
 $\mathcal{G}_{legt} = \mathcal{G}_{legt} \cup \{p_i^{pert}\}$
 else if $T = \text{spawn}$ **then**
 if $k = \text{rectangle}$ **then**
 $p_{spawn} = \text{GenerateSpawnRectangle}(w, h)$
 else
 $p_{spawn} = \text{GenerateSpawn}(w, h, \bar{r}, s, i, v_m)$
 end if
 $\mathcal{G}_{pert} = \mathcal{G}_{pert} \cup \{p_i^{org}, p_{spawn}\}$
 $\mathcal{G}_{legt} = \mathcal{G}_{legt} \cup \{p_{spawn}\}$
 else
 $\mathcal{G}_{pert} = \mathcal{G}_{pert} \cup \{p_i^{org}\}$
 end if
 end if
end for

APPENDIX

A. Details on Perturbation Algorithm

In this section, we discuss the label perturbation algorithm in greater detail. The algorithm has been outlined in the main part of the manuscript. The pseudocode in algorithm 1 of the label perturbation algorithm is provided for the sake of completeness and contains additional sub-routines (`ChangeClass` performing label flips, `GenerateSpawnRectangle`, as well as `GenerateSpawn` performing label spawns) which have not been discussed in the main part of the paper.

Flip. If the label error type *flip* is chosen, then the class of the respective polygon is changed. To do this, we randomly select a new class from the set of all classes, excluding the current class of this polygon. After assigning a new class, we obtain our perturbed polygon.

Spawn. If the label error type sampled in the label perturbation algorithm is of type *spawn*, we follow two different approaches to generate spawns. If the respective dataset is

Function *GeneratePolygon()*

function *GeneratePolygon*(($(x_c, y_c), r, i, s, v_{numb}$))
 $i = (i \cdot 2 \cdot \pi) / v_{numb}$
 $s = s \cdot r$
 # generate list of random angles
 $steps = \{\}$
 $l = (2 \cdot \pi / v_{numb}) - i$
 $u = (2 \cdot \pi / v_{numb}) + i$
 $c = 0$
 for $m = 0$ in v_{numb} **do**
 # uniformly choose angles
 $a \sim \mathcal{U}([l, u])$
 $steps = steps \cup \{a\}$
 $c = c + a$
 end for
 $c = c / 2\pi$
 for a in $steps$ **do**
 $a = a / c$
 end for
 # uniformly choose start angle
 $a_{start} \sim \mathcal{U}([0, 2\pi])$
 # collect edges of polygon
 $e = \{\}$
 for $j = 0$ to v_{numb} **do**
 $g \sim \mathcal{N}((r, s))$
 $r_{new} = \min(2 \cdot r, \max(g, 0))$
 $e = e \cup \{[x_c + r_{new} \cdot \cos(a), y_c + r_{new} \cdot \sin(a)]\}$
 $a_{start} = a_{start} + steps[j]$
 end for
 return e
end function

from object detection, the generated spawns has to be rectangular. Therefore, we uniformly sample the width and height of the rectangle, and depending on these values, we uniformly select the coordinates of the bottom left edge of the spawn. Using these coordinates, we can create a randomly generated polygon for object detection datasets.

If the dataset is from a segmentation dataset, the form of this spawn has to have more vertices. The generation is fully random here as well. The code for spawn generation in segmentation datasets was copied in its entirety from [45]. For this polygon, we select fixed parameters for the radius range, spikiness and irregularity, as well as the maximum number of vertices. Irregularity describes the variance in the spacing of the angles between consecutive vertices. Spikeness refers to the variance in the distance of each vertex from the centre of the circumference. We uniformly sample a radius from the radius range, and depending on this parameter, as well as the image width and height, we uniformly sample the coordinates of the spawn's centre. Based on these parameters, we can generate the polygon. Starting from the centre of the polygon, we create it by sampling points on a circle around the centre. Random noise is added by varying the angular

Function *GenerateSpawn()*

```

function GenerateSpawn((w, h, (r1, r2), s, i, vm))
  # generate random polygon with multiple vertices
  sample r ~  $\mathcal{U}(\{r_1, r_1 + 1, \dots, r_2 - 1, r_2\})$ 
  # uniformly choose center of polygon
  sample xc ~  $\mathcal{U}(\{r, r + 1, \dots, w - r - 1, w - r\})$ 
  sample yc ~  $\mathcal{U}(\{r, r + 1, \dots, h - r - 1, h - r\})$ 
  # uniformly choose number of vertices
  # polygon requires at least 4 coordinates
  vnumb ~  $\mathcal{U}([4, 5, \dots, v_m])$ 
  pspawn = GeneratePolygon((xc, yc), r, i, s, vnumb)
  # uniformly choose a class
  cs ~  $\mathcal{U}(\{0, \dots, c - 1\})$ 
  pspawn ← cs
  return pspawn
end function

```

function *GenerateSpawnRectangle*((w, h))

```

  # generate quadratically spawn
  Sample ws ~  $\mathcal{U}([0, w])$ 
  Sample hs ~  $\mathcal{U}([0, h])$ 
  Sample xs ~  $\mathcal{U}([0, w - w_s])$ 
  Sample ys ~  $\mathcal{U}([0, h - h_s])$ 
  e1 = [xs, ys]
  e2 = [xs + ws, ys]
  e3 = [xs + ws, ys + hs]
  e4 = [xs, ys + hs]
  pspawn = [e1, e2, e3, e4]
  # uniformly choose a class
  cs ~  $\mathcal{U}(\{0, \dots, c - 1\})$ 
  pspawn ← cs
  return pspawn
end function

```

```

function ChangeClass(piorg)
  # piorg with class ciorg
  # uniformly choose a new class
  Sample  $\bar{c} \sim \mathcal{U}(\{0, \dots, c - 1\}) \setminus \{c_i^{org}\}$ 
  piorg ←  $\bar{c}$ 
  pipert ← piorg
  return pipert
end function

```

spacing between sequential points and the radial distance of each point from the centre. The resulting edges form the spawned polygon.

For label error detection, we apply the label perturbation algorithm multiple times to a given dataset as additional data augmentation for training data. This results in a number of copies of the dataset with different label errors. An overview of possible results are shown in fig. 6.

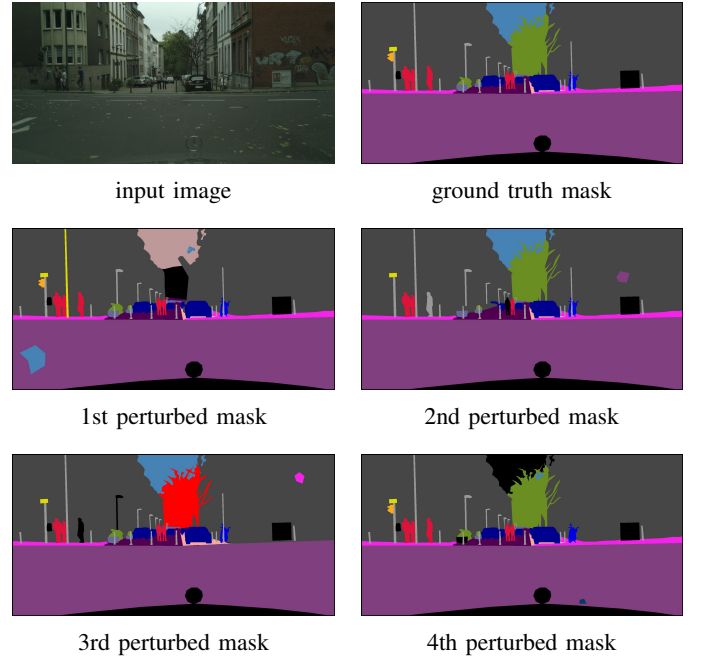


Fig. 6: Results of multiple application of perturbation algorithm

B. Implementation Details

TABLE VI: An overview of the dataset implementations in relation to the perturbed dataset and the reference networks used.

Dataset	q	t	no. of label errors		reference DNNs
			train	val	
Cityscapes	0.2	10	194,635	3558	segformer
ADE20K	0.2	2	79,077	4136	mask2former
PascalVOC	0.2	10	94,202	2911	GroundingDino
COCO	0.2	2	344,709	7507	Co-DETR
LIVECell	0.1	1	117,799	44,331	ResNeSt

1) Datasets:

a) *PascalVOC*.: The PascalVOC dataset contains images of everyday scenes with annotations for 20 classes and is used among other as object detection dataset. We used the *train+val2007* and *train+val2012* datasets, comprising a total of 16,551 images, for training, and the *test2007* dataset, comprising 4,952 images, for validation. We chose a perturbation rate of $q = 0.2$ and created $t = 10$ copies of the training datasets, resulting in approximately 97,000 simulated label errors for training and validation. Moreover we did not set any size restriction for objects due to the partly limited number of boxes per image. As reference network we opted for GroundingDino with a box threshold of 0.35 and a text threshold of 0.3. These parameters were chosen to prevent predictions with duplicate labels. Additionally, we applied Non-Maximum Suppression (NMS) to delete predicted boxes of the same class, as well as boxes of different classes that overlap significantly with an intersection over union (IoU) of

at least 0.7. Of the overlapping boxes of different classes, the box with the higher score was chosen.

b) *COCO*.: The object detection dataset COCO is characterized by its large number of images representing everyday scenes and categories. There are 80 different classes annotated on 118,287 images in the training set and 5,000 images in the validation set. The perturbed dataset was created using $t = 2$ copies and a perturbation rate of $q = 0.2$. In contrast to PascalVOC, we applied the perturbation algorithm only twice, given the already high number of images. The Co-DETR object detector was used for reference COCO predictions. It is based on a CoDETR backbone, a CoDINOHead as query head and a CoATSSHead as bounding box head. The pre-trained checkpoint we used achieves a box AP of 66.0 and we collected all predictions with a score of at least 0.3.

c) *Cityscapes*.: The Cityscapes semantic segmentation dataset contains high-resolution images of street scenes with 19 annotated classes. For training and validation, we use the 2,975 training images and 500 validation images, as well as generating different perturbed datasets for the ablation studies. The comparisons with baselines were performed with a perturbation rate of $q = 0.2$ and $t = 10$ copies of the training set. This dataset consists of images containing many annotated objects. The more objects in an image, the more likely label errors are to appear. Thus, we opted for a perturbation rate of 20%, which is a realistic number of possible label errors. For the spawn generation, we decided for a radius range of $\bar{r} = (10, 80)$, an irregularity $i = 0.35$ and a spikeness of $s = 0.2$. For the ablation studies, we selected segformer, DeeplabV3+ and VLTseg as reference networks. The pre-trained segformer checkpoint achieves an mIoU of 82.25%, and the DeepLabV3+ checkpoint achieves a value of 79.88%. The VLTseg checkpoint achieves an mIoU of 86.30% on Cityscapes.

d) *ADE20K*.: The ADE20K semantic segmentation dataset of everyday scenes contains annotated objects from 150 different classes. We simulated label errors in 20,210 training and 2,000 validation images by applying the perturbation algorithm with a perturbation rate of $q = 0.2$, a number of copies of $t = 2$, a radius range of $\bar{r} = (10, 30)$, irregularity of $i = 0.35$ and spikeness $s = 0.05$. The reduced number of copies compared to Cityscapes is again due to the large number of training images already available. All 150 classes were incorporated into the label perturbation algorithm. The pre-trained Mask R-CNN checkpoint, which we used for reference predictions, consists of a D2SwinTransformer backbone model and achieves an mIoU of 53.9%.

e) *LIVECell*.: We chose the LIVECell dataset, consisting of images with eight different cell types that are all categorized to class *cell*, as the instance segmentation dataset for our experiments. For training, we combined the training and validation data to create a single set of 3,727 images, and for validation, we used the 1,512 test images. We applied the label perturbation algorithm with perturbation rate of $q = 0.1$ and applying the algorithm only once $t = 1$ with the same parameters for spawn generation as in ADE20K. The reasons

for this are twofold: firstly, due to the high number of objects, applying the algorithm more frequently would take a long time, and secondly, due to the similarity of the images, where more copies do not introduce any variance. Due to the dataset only containing a the single class, we only simulated *drops* and *spawns*. The reference predictions are based on the anchor-based model from the Detectron2-ResNeSt architecture. It achieves a mask mAP of 47.89%. We collected all predictions with a score threshold of at least 0.1.

2) *Training Details*: We opted for the Cascade Mask-RCNN for training on simulated label errors. Training all the datasets took 150,000 iterations, with a batch size of 10, and the solver made steps after 80,000 and 120,000 iterations reducing the learning rate by a factor of 0.1. In addition to creating multiple perturbed copies of the original dataset by the label perturbation algorithm, we applied further data augmentations such as random flip and rotation, random crop and resizing of the shortest edge. Training of all datasets was performed with a learning rate of 0.01, except for Cityscapes where a learning rate of 0.001 was chosen. LIVECell is the only dataset for which we trained without a difference mask. For all the other datasets, we opted to train with an input image I_{rgb} , a perturbed mask I_{pert} , a predicted mask I_{pred} and a difference mask I_{diff} . This choice turned out most beneficial in our experiments.

Another configuration that we adjusted before training was the input image normalization. For training purposes, the input images' pixel values were normalized by subtracting the mean and dividing by the standard deviation of the images. Since we have nine or twelve channels, respectively, we perform the normalization of each channel individually. For training our label error detector, we computed the mean and standard deviation of the input image I_{rgb} and the perturbed mask I_{pert} . Then, we set the same values for the perturbed mask I_{pert} , the prediction mask I_{pred} and, if used, the difference mask I_{diff} as well. The exact computations are shown below based on the case of four input images, i.e., when including I_{diff} . The computations are analogously for input $(I_{rgb}, I_{pert}, I_{pred})$.

Given are the RGB image I_{rgb} , the perturbed mask I_{pert} , the prediction mask I_{pred} and the difference mask I_{diff} with $I_{rgb}, I_{pert}, I_{pred}, I_{diff} \in \{0, 1, \dots, 255\}^{H \times W \times 3}$. The input tensor is a stack / concatenation of all four images

$$I_{in} = [I_{rgb}, I_{pert}, I_{pred}, I_{diff}] \in \{0, 1, \dots, 255\}^{H \times W \times 12},$$

where W is the width and H the height in pixels of the images. For the normalization of each input tensor I_{in} , we compute for each channel in I_{rgb} the mean and standard deviation by

$$\mu_c^{rgb} = \frac{1}{H \cdot W} \sum_{h=1}^H \sum_{w=1}^W i_{h,w,c}^{rgb}$$

$$\sigma_c^{rgb} = \sqrt{\frac{1}{H \cdot W} \sum_{h=1}^H \sum_{w=1}^W (i_{h,w,c}^{rgb} - \mu_c^{rgb})^2}$$

for all channels $c = 1, 2, 3$. For the perturbed mask the computations are performed analogously. The results for I_{rgb} are

$$\mu_{rgb} = (\mu_1^{rgb}, \mu_2^{rgb}, \mu_3^{rgb}) \quad \text{and} \quad \sigma_{rgb} = (\sigma_1^{rgb}, \sigma_2^{rgb}, \sigma_3^{rgb})$$

and mean and standard deviation of I_{pert} are

$$\mu_{pert} = (\mu_1^{pert}, \mu_2^{pert}, \mu_3^{pert}) \quad \text{and} \quad \sigma_{pert} = (\sigma_1^{pert}, \sigma_2^{pert}, \sigma_3^{pert}).$$

The RGB image I_{rgb} is normalized using its computed mean and standard deviation. The perturbed mask I_{pert} , predicted mask I_{pred} and difference mask I_{diff} are normalized by μ_{pert} and σ_{pert} . The result is

$$I_{in}^{norm} = \frac{I_{in} - (\mu_{rgb}, \mu_{pert}, \mu_{pert}, \mu_{pert})}{(\sigma_{rgb}, \sigma_{pert}, \sigma_{pert}, \sigma_{pert})} \in \mathbb{R}^{H \times W \times 12}.$$

C. Details on Difference Mask Generation

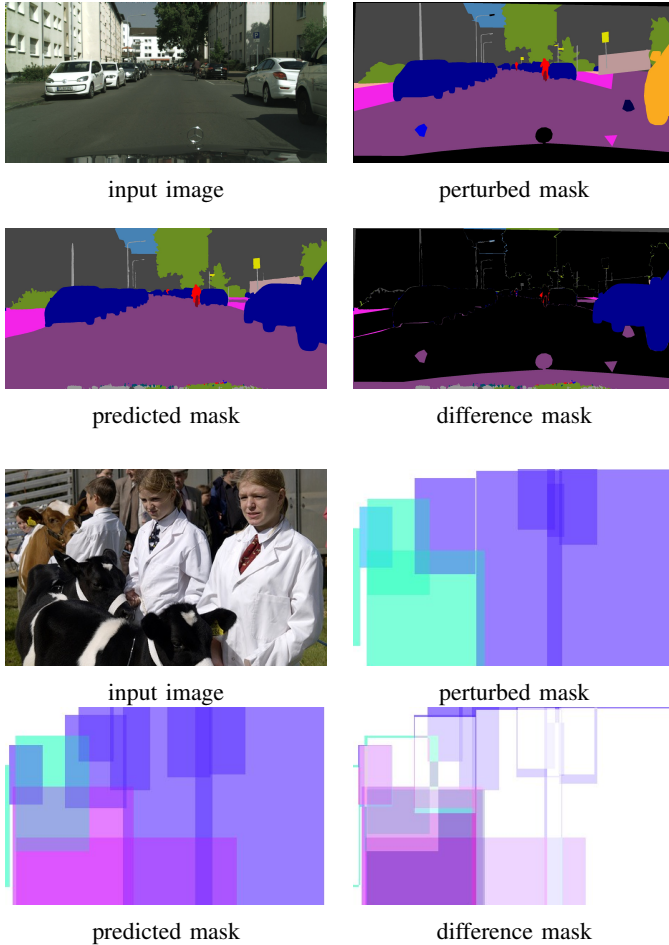


Fig. 7: Overview of input image, perturbed mask, prediction mask and difference mask for semantic segmentation (first four images) and object detection datasets.

The difference masks illustrate the differences between the perturbed ground truth and the prediction mask in the semantic

segmentation and object detection datasets (see fig. 7). For Cityscapes and ADE20K, the difference mask is created by comparing the perturbed and predicted masks pixel by pixel. Each pixel that differs is given the same color as the corresponding pixel in the predicted mask. For object detection datasets, only the differences between the HSV-colored perturbed and predicted masks are computed. For instance segmentation datasets, especially in the case of LIVECell, each object is assigned a random color, making it difficult to find corresponding instances between the perturbed and predicted masks. Since training the label error detector on perturbed LIVECell data was successful without a difference mask, we decided to proceed without one for instance segmentation.

D. Ablation Studies

In addition to the ablation studies in the main part of the manuscript, we performed a study training on three separate classes: *drop*, *flip* and *spawn*. To train the label error detector, we used the Cascade Mask-RCNN as the instance segmentor and a perturbed dataset with a perturbation rate of $q = 0.2$ and a number of copies of $t = 10$ as training data. Instead of using one class *error* for the instance segmentor’s training feedback (as done in all previous experiments), we used the classes *drop*, *flip* and *spawn* for training the instance segmentor. Training was performed with a learning rate of 0.001, using the difference mask I_{diff} as additional input alongside the input image I_{rgb} , the perturbed mask I_{pert} and the predicted mask I_{pred} from the reference network Segformer. The results are shown in table VII. We evaluated the trained label error detector on a validation dataset with a perturbation rate of $q = 0.2$ and a number of copies of $t = 1$. In the evaluation, we take all three classes into consideration. It can be seen that the label error objects known as *spawns* are the easiest to learn and also easiest to find giving rise to the highest AP, precision, recall and F1-score out of all classes. By comparing the results of *drop* and *flip*, we observe that the latter performs even better in terms of 14% higher AP and 8% higher recall. As seen in the main part of this paper, training on class *error* achieves a higher AP of 61.74% and a higher recall of 95.78%.

E. A Collection of Real Label Errors

Below, we present a collection of real label errors that we have found using our label error detector. Each collage contains 18 pairs of images, each of which represents one or more real label errors. These five figures show label errors from PascalVOC (fig. 9), COCO (fig. 10), ADE20K (fig. 11), Cityscapes (fig. 12) and LIVECell (fig. 13).

F. Limitations

We performed a study of the false positives of our label error detection method in fig. 8. In particular in the case of object detection annotations, we noticed that our method (as well as the baselines considered in our work) is influenced by false predictions of the reference model. This represent a gap that will be closed by stronger object detectors in future. Furthermore, although we demonstrated the generalization of

TABLE VII: Performance results on validation data of a trained label error detector $f_{le}(I_{rgb}, I_{pert}, I_{pred}, I_{diff})$ on classes *drop*, *flip* and *spawn* on a perturbed Cityscapes validation dataset with perturbation rate $q = 0.2$ and $t = 1$). For evaluation we chose a score threshold of 0.5 and an IoU threshold of 0.1.

Class	AP	TPs	FPs	FNs	Precision	Recall	F1-Score
drop	32.03	723	1463	159	33.07	81.97	47.13
flip	46.07	909	1741	124	34.30	88.00	49.36
spawn	88.77	1153	199	38	85.28	96.81	90.68
drop, flip, spawn	55.62	2785	3403	321	45.01	89.67	59.93

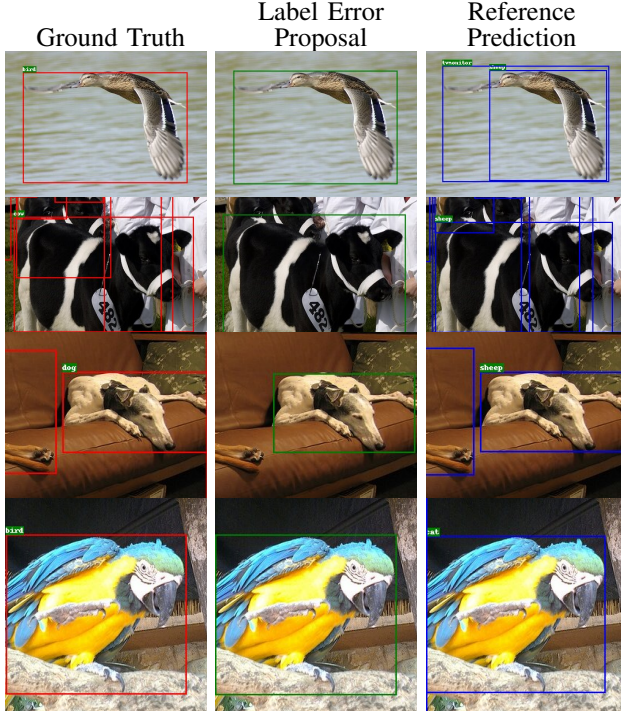


Fig. 8: Examples of false positive label error proposals of our label error detector on PascalVOC dataset. In red are the ground truth bounding boxes, in blue are the predicted bounding boxes from reference network GroundingDino and in green are the label error proposals of our label error detector for real label errors.

our method to the detection of real label errors, it is to be expected that the label errors that we simulate via our label perturbation algorithm do not cover the entire distribution of potential label errors.

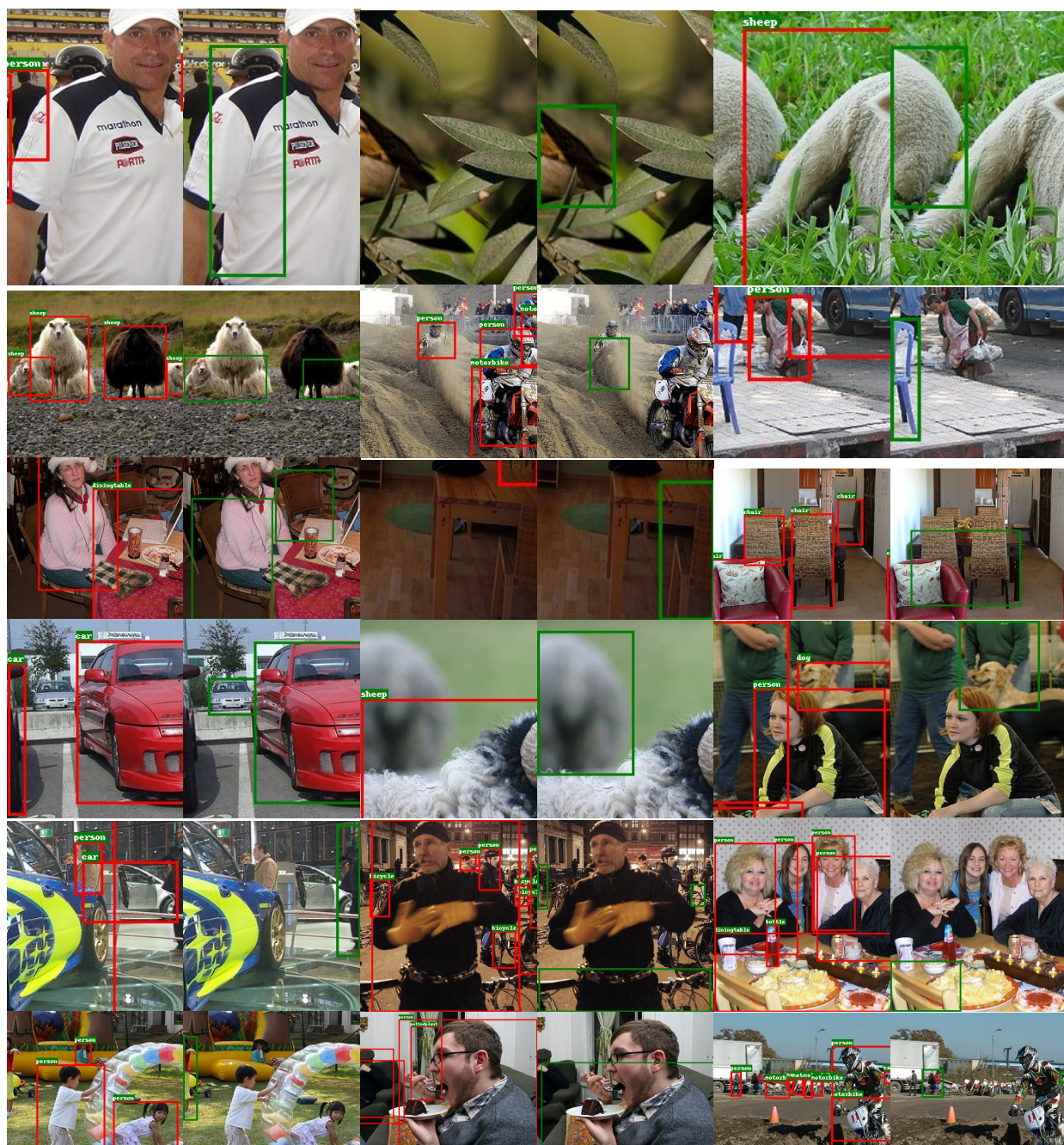


Fig. 9: A collection of label errors present in PascalVOC. In red are the ground truth boxes and in green the predictions of our label error detection method.



Fig. 10: A collection of label errors present in COCO. In red are the ground truth boxes and in green the predictions of our label error detection method.



Fig. 11: A collection of label errors present in ADE20K. The segmentation masks show the ground truth annotations and in green are the prediction of our label error detection method.

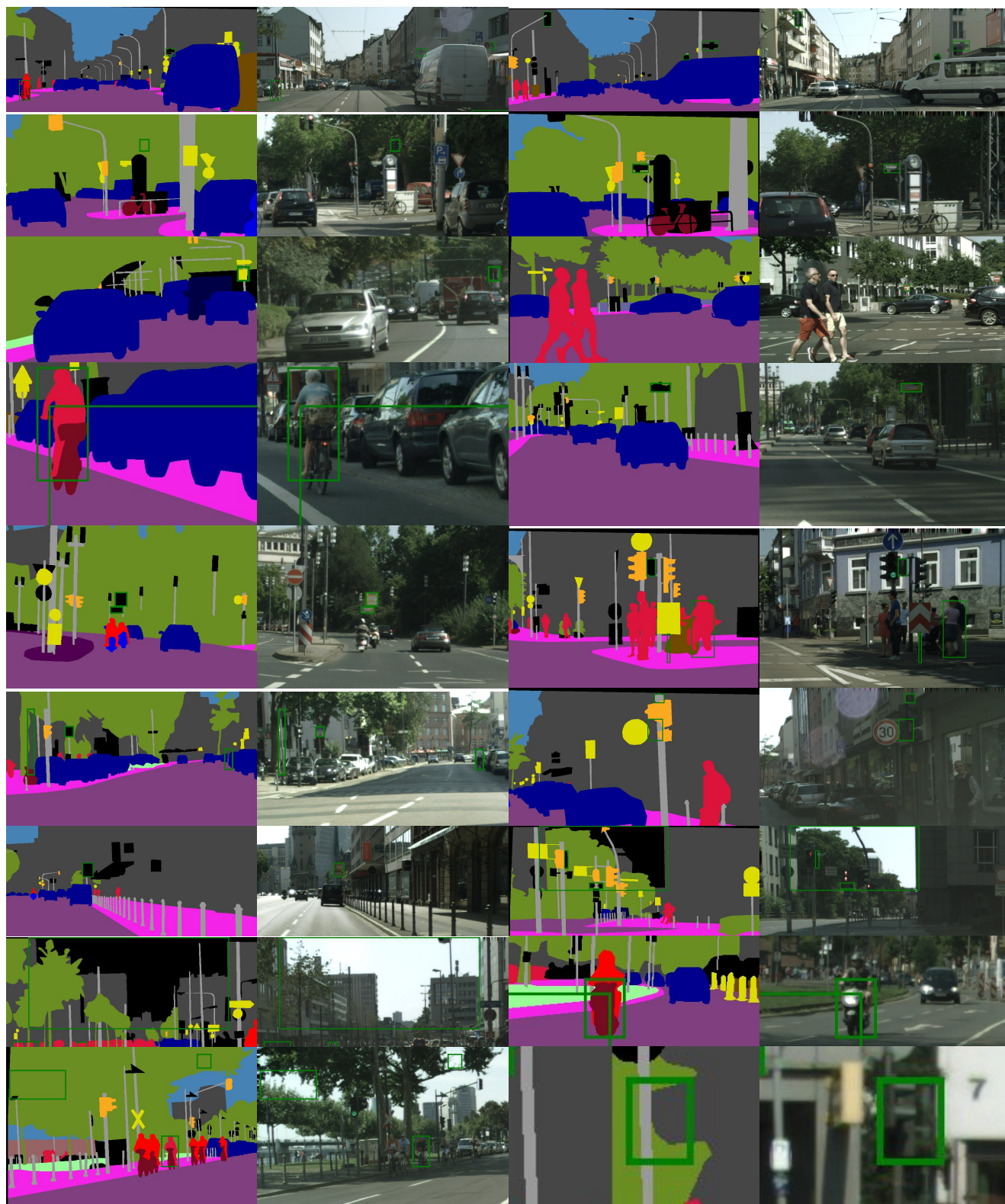


Fig. 12: A collection of label errors present in Cityscapes. The segmentation masks show the ground truth annotations and the green boxes are the predictions of our label error detection method.

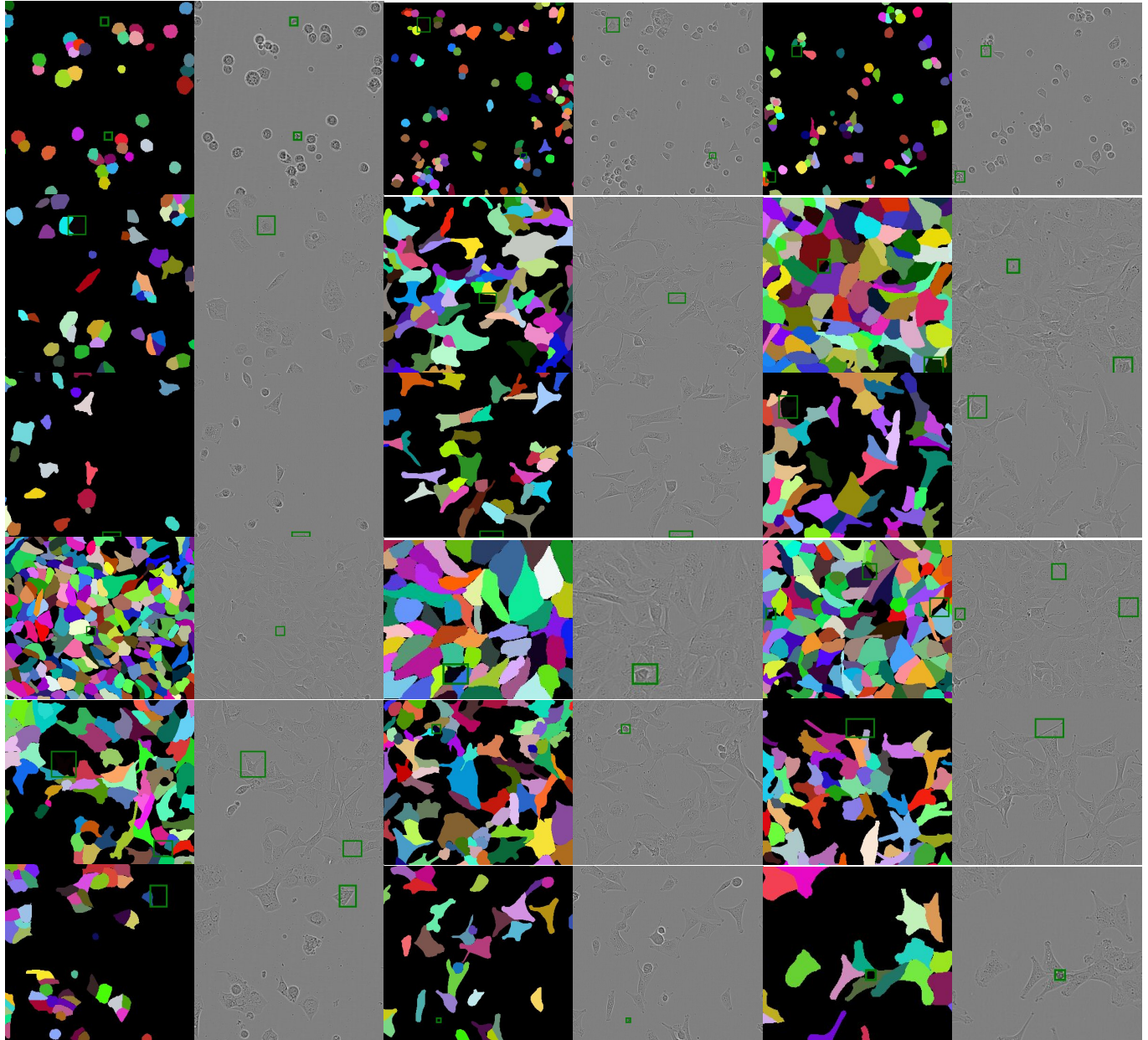


Fig. 13: A collection of label errors present in LIVECell. The segmentation masks show the ground truth annotations and the green boxes are the predictions of our label error detection method.