Per-frame mAP Prediction for Continuous Performance Monitoring of Object Detection During Deployment

Quazi Marufur Rahman, Niko Sünderhauf, Feras Dayoub
Australian Centre for Robotic Vision, Queensland University of Technology
Brisbane, Australia
{quazi.rahman, niko.suenderhauf, feras.dayoub}@qut.edu.au

Abstract

Performance monitoring of object detection is crucial for safety-critical applications such as autonomous vehicles that operate under varying and complex environmental conditions. Currently, object detectors are evaluated using summary metrics based on a single dataset that is assumed to be representative of all future deployment conditions. In practice, this assumption does not hold, and the performance fluctuates as a function of the deployment conditions. To address this issue, we propose an introspection approach to performance monitoring during deployment without the need for ground truth data. We do so by predicting when the per-frame mean average precision drops below a critical threshold using the detector's internal features. We quantitatively evaluate and demonstrate our method's ability to reduce risk by trading off making an incorrect decision by raising the alarm and absenting from detection.

1. Introduction

Object detection is a crucial part of many safety-critical applications such as robotics and autonomous systems. For safe operation, an autonomous vehicle (AV), for example, needs to accurately locate and identify critical objects like other vehicles and pedestrians on the road. To achieve this goal, there is ongoing research [37, 35, 25, 10, 1, 15, 3, 22, 23, 4] to improve the speed and accuracy of object detection models. However, due to the discrepancy between training data and deployment environments (i.e., dataset shift [31]) and many other unavoidable factors like sensor failure or degraded image quality, a consistent deployment performance can not be guaranteed. Hence, object detection accuracy can fluctuate without any prior notification while deployed on an autonomous vehicle. A silent failure like this in the object detection model is a significant concern. Due to this failure, the AV can cause catastrophic damage if it operates based on erroneous object detection. Undetected performance drops are a significant bottleneck for the widespread deployment of autonomous vehicles in our everyday lives. Hence, for safety, robustness, and reliability, the importance of performance monitoring of a deployed object detection model is paramount.

The standard practice to prepare an object detection model for deployment is to train and evaluate the model using training and evaluation split of some dataset to measure the accuracy and generalization capacity. Here, the assumption is the training and evaluation data are representative of the real operating environment. However, this assumption does not hold in the context of autonomous vehicles where the operating environment is continuously evolving and might change unexpectedly. Consequently, object detection performance fluctuates without any prior notification. Moreover, the performance might drop below any critical threshold, which can cause a fatal incident. See Figure 1 for an overview.

One possible solution is to develop an exceptionally accurate and domain adaptive object detection system for autonomous vehicles. However, it is impossible in most practical circumstances to account for all imaginable future deployment conditions during training. Another approach is to identify when the performance of the deployed object detector drops below a critical threshold. So without the need to increase the detection accuracy directly, a performance drop identifier can protect the autonomous vehicle by providing crucial alerts during periods of silent failure. However, measuring the performance drop directly during deployment is impractical due to the absence of ground-truth data in this phase. Therefore, we advocate equipping object detectors with self-assessment capability to detect instances of performance drop during deployment.

There are several factors such as unknown environments,

The authors are with the Australian Centre for Robotic Vision at Queensland University of Technology (QUT), Brisbane, QLD 4001, Australia. This research has been conducted by the Australian Research Council (ARC) Centre of Excellence for Robotic Vision (Grant CE140100016). The authors acknowledge continued support from the QUT Centre for Robotics.

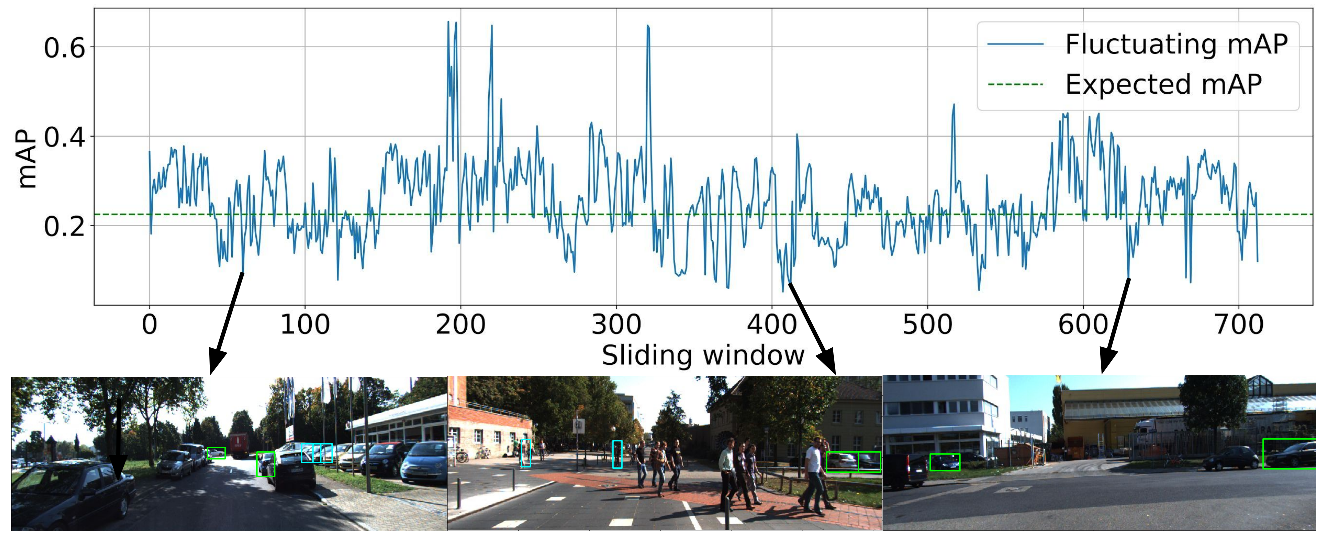

Figure 1: First row, an example of object detection performance fluctuation during the deployment, tracked using a sliding window of ten frames. mAP is computed for the ten frames at a time. The dashed line represents a predefined critical threshold. We can see that mAP drops below this threshold from time to time. The second row shows some samples from the low mAP regions. Green and Cyan boxes represent false negative and false positive errors made by the object detector.

degraded input image quality and sensor failure that can deteriorate the performance of an object detector during the deployment phase. Instead of identifying the reason behind this performance drop, we are focusing on raising a warning when the drop happens. Besides, our assumption is the object detection model weight is fixed, and the deployment environment is previously unknown.

Self-assessment is becoming a prerequisite for any vision-based efficient, safe, and robust robotic system. This capability is often referred to as introspective perception [7, 29]. For autonomous driving, an introspective object detection system can hand over the control to human drivers when it can predict inconsistency in its operation. There are several works [13, 18, 32, 28] towards addressing the requirement of providing self-assessment in a deep learning based robotic vision system. However, there are very few works towards introspective systems for object detection. To this end, our paper makes the following contributions:

1. We propose an introspective approach to performance monitoring of object detection during deployment without access to ground truth labels.
2. We propose an internal integrated feature based on the mean, max and statistics pooling techniques for performance monitoring.
3. We introduce the use of per-frame mAP prediction for continuous performance monitoring of object detectors.

The rest of the paper is organized as follows: In Section 2, we review the related works on introspective perception systems. In Section 3, we introduce our framework to find the performance drop for an object detection system. Section 4 outlines our experimental setup. Section 5 presents the results and finally in Section 6 we conclude and suggest areas for future work.

2. Related Work

In robotics, the idea of self-assessment was introduced by [29] to achieve reliable performance in a real environment. They described this self-assessment as to the introspection capability of a mobile robot while operating in an unknown environment. Later [13] and [38] adopted the idea of introspection for classification and semantic mapping respectively in the context of robotics. These works examine the output of the underlying system to predict the likelihood of failure on any given input.

Another line of research is to predict the perception system performance from the input itself. In this paradigm, [20] introduced an evaluator algorithm to predict the failure of a human pose estimation model. Zhang *et al.* [42] introduced the terminology *basesys* and *alert* in failure prediction context. They proposed a general framework where *alert* is used to raise a warning when the underlying system *basesys* fails to make a correct decision from an input. Daftry *et al.* [7] proposed an introspective framework to predict an image classifier model failure deployed on a micro aerial vehicle. Following a similar methodology, [39] proposed a model to predict how hard an image is for an underlying classifier. Using a probabilistic model, [14] predicted the probable performance of a robot's perception system based on past experience in the same location.

Recently, confidence estimation and Bayesian approaches for uncertainty estimation have gained popularity to detect how well the underlying model has performed on the input. TrustScore [21], Maximum Class Probability [17] and True Class Probability [6] are some of the works that measure the confidence of the underlying model for a given task using the confidence estimation paradigm. Using a Bayesian approach, [11] proposed to use dropout as a Bayesian approximation technique to represent model uncertainty. Following their work, [9, 19] have used dropout sampling to identify the quality of image and video segmentation network. Here, all of these works focus on predicting model failure using different approaches for classification and segmentation tasks.

In the context of object detection, [27, 5] have used several approaches to identify the failure of an object detection system. Both of these works are suitable to identify false positive errors made by an object detector. Whereas, [33] and [34] have proposed different procedures to detect false negative errors made by an object detection model. Our proposed approach differs from these methods and focuses on addressing the issue of predicting low per-frame mAP, which covers both false positive and false negative errors as well as poor object localization.

Another line of research focus to propose various summary metrics other than mAP to evaluate object detection performance. nuScenes detection score [2], delay metric [26] and planner-centric metric [30] are some recent works of this paradigm. These works rely on the ground-truth data to evaluate the object detection performance, whereas our work does not require the ground-truth to predict the performance drop during the deployment phase.

3. Approach Overview

In this section, we describe our proposed framework to predict the performance drop of an object detection system during deployment without using any ground-truth data. We assume that the deployed object detection model weight remains frozen during this phase. First of all, we will define the problem.

Assuming we have an object detector O with backbone deep neural network B, O is trained to detect a set of objects T from a training dataset, D_t. We also have an evaluation dataset D_e, similarly distributed as D_t. D_e contains a set of images $I = \{I_1, I_2 \dots I_n\}$ and corresponding annotations $L = \{L_1, L_2 \dots L_n\}$ per image. After the object detection training phase, O is applied on D_e to detect all the objects from T. Thus, we get a set of predictions per image, $P = \{P_1, P_2 \dots P_n\}$. Using the pairs of annotations and predictions per image (L_i, P_i), we compute the per-frame mAP, $M = \{M_1, M_2 \dots M_n\}$ following the procedure at [24]. Here, per-frame mAP quantifies O's performance to detect all the existing objects in each image.

We assign each image of D_e into *success* and *failure* classes using the Equation 1. Here λ is chosen to be the k^{th} percentile of M. The *failure* class contains the $k\%$ image frames from the D_e where O was not accurate enough to detect the available objects. The choice of k here is application specific. We want to train the introspective perception system *alert* to predict the images similar to the *failure* class where per-frame mAP will be lower than λ.

$$\mathcal{L}(I) = \begin{cases} failure, & mAP_{\texttt{per-frame}} < \lambda \\ success, & \text{otherwise.} \end{cases} \tag{1}$$

To train the *alert* we use features $F = \{F_1, F_2 \dots F_n\}$ for each image from D_e. Following the failure prediction network proposed by [39] and [6], the final convolutional layer of backbone B is used to extract all the necessary features. Assuming that, there are N channels at the last layer of B and each activation map is of size $W \times H$, we apply Equation 2 on the last layer to extract the mean pooling feature F_{mean}. Here, $f(x, y)$ represents the spatial unit of each activation map.

$$F_{mean} = \frac{\sum_{x=1}^{H} \sum_{y=1}^{W} f(x, y)}{W \cdot H}. \tag{2}$$

Applying Equation 3 on the last layer of B, we generate the max pooling feature F_{max},

$$F_{max} = \max_{x \in [1,H]} \max_{y \in [1,W]} f(x, y). \tag{3}$$

Inspired by [36], we calculate the standard deviation from each activation map to generate the statistics pooling feature F_{std} following the Equation 4. Here $std(f_i)$ calculates the standard deviation of i^{th} feature map.

$$F_{std} = std(f_1) \oplus std(f_2) \oplus \dots std(f_N). \tag{4}$$

All the features described above are concatenated together to generate the feature $F_{mean\_max\_std}$ for the *alert* system. Equation 5 formulates this process.

$$F_{mean\_max\_std} = F_{mean} \oplus F_{max} \oplus F_{std}. \tag{5}$$

We train a binary classifier using the $F_{mean\_max\_std}$ feature and the corresponding labels from Equation 5 and Equation 1 respectively. The classifier is trained to predict the probability of an image feature to be in the *failure* class. Following [42], we will refer to the object detection model and its corresponding binary classifier as *basesys* and *alert* respectively. Figure 2 shows the incorporation between the *basesys* and *alert* system.

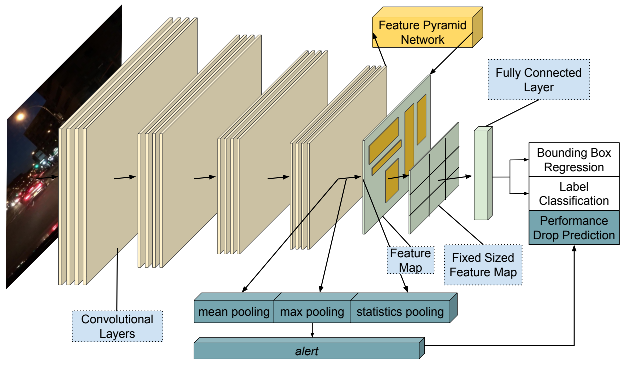

Figure 2: The architecture of our proposed *basesys* and *alert* system together. The last convolutional feature of the backbone is pooled using the mean, max and statistical pooling layer to generate the feature for *alert*. *Alert* consists of a binary classifier that predicts the performance drop of the *basesys*.

4. Experimental Setup

In this section we will describe the settings that we have used to train the *basesys* and *alert* system.

Datasets: We used all images from *kitti* [12] dataset and one frame per video from *bdd* [40] dataset to train both *basesys* and corresponding *alert* system. Randomly selected 60%, 20% and 20% images from both dataset have been used for *basesys* training, evaluation and testing purposes. As person and car classes are available in both of these dataset, we used these two objects as *basesys* target class. After the object detection training, *basesys* is used to detect person and car from the 20% evaluation split. Based on *basesys* performance on the evaluation split, we collect image features and labels for the *alert* system following the procedure described in Section 3. Thus the features and labels collected from *basesys* evaluation split works as a training dataset for the corresponding *alert* system. To test the *alert*, we first apply *basesys* on the testing split and measure its per-frame performance drop, which works as the testing data for the *alert* system. In some of our experiments, we will train and test *basesys* and *alert* using training and testing split of a single dataset. We will refer to these settings as *similar dataset*. In the rest of the experiments, *basesys* and *alert* will be trained using a training split of one dataset and tested using a testing split of another dataset. These arrangements will be referred to as *cross dataset* settings.

Basesys Training: We have used Faster RCNN object detection network [35] as the *basesys* in all of our experiments. *Basesys* has been trained using transfer learning to detect person and car objects from both *kitti* and *bdd* dataset. Two different versions of Residual Neural Network [16], ResNet18 and ResNet50 have been used as the *basesys* backbone. In our experiments, the *basesys*, trained using RestNet50 backbone has performed better than the ResNet18 backbone. Table 1 shows comparative performance using the mean average precision (mAP) for all different *basesys* and dataset combinations. We used ResNet18 and ResNet50 to demonstrate how well the proposed *alert* system performs for weaker and stronger *basesys* respectively.

Feature Collection: We experimented with multiple features to find the most suitable one for the proposed *alert* system. The first set of features are collected from *basesys* bounding box proposals.

- *mean_conf_score*: This feature exploits object proposal confidence score to determine *basesys* performance drop. As *basesys* proposes multiple bounding boxes with corresponding confidence scores and labels during object detection, we use the mean of confidence scores which are greater than 0.5 to build the first feature. Here, a lower mean confidence score indicates a potential performance drop in the *basesys*.

- *n_proposals*: We assume that a crowded environment might be a factor for *basesys* performance drop. To evaluate this assumption, we used the number of proposals having a confidence score greater than 0.5 as a performance drop indicator.

The second set of features are collected from two external deep convolutional neural networks.

- *classifier:* Two different versions of Residual neural network, Resnet18 and ResNet50 have been used to extract image features to train the *alert* system. Both of these networks are pre-trained on ImageNet [8] dataset.

Table 1: *Basesys* mean average precision (mAP) using ResNet18 and ResNet50 backbone. Here *basesys* is trained and tested using *similar dataset* and *cross dataset* settings. *Basesys* accuracy drops when trained and tested on different dataset.

ResNet18			ResNet50		
training dataset	testing dataset	mAP	training dataset	testing dataset	mAP
kitti	kitti	0.292	kitti	kitti	0.377
kitti	bdd	0.130	kitti	bdd	0.182
bdd	kitti	0.200	bdd	kitti	0.259
bdd	bdd	0.331	bdd	bdd	0.499

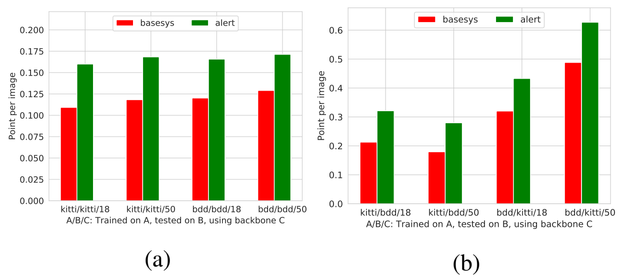

Figure 3: Risk-Averse Metric for the proposed *alert* system. (a) Point per image earned by *basesys* with and without considering *alert* warning when trained and tested on similar dataset. (b) Point per image for *basesys* with and without *alert* system when trained and tested on different dataset. In both cases, *basesys* earns more points per image when associated with *alert*.

- *places365:* We used ResNet18 and ResNet50 network pre-trained on Places365 dataset to extract features to train the *alert* system.

  In both cases, average pooling has been used at the final convolutional layer to extract the necessary image features.

We use the *basesys* backbone to extract the third set of features. These will be referred to as the internal features.

- *layer:* We applied the mean-pooling operation in all of the convolutional layers of the backbone and concatenated them to create this feature.

- *mean*, *max* and *std*: Applying the mean, max and statistics pooling technique described in Section 3 at the last convolutional layer of *basesys* backbone, we extracted the *mean*, *max* and *std* features.

- *mean_std* and *mean_max:* Using the concatenation operation and following the feature generation technique proposed in [36] and [41], we generate two new features *mean_std* and *mean_max* using the *mean*, *max* and *std* feature.

- *mean_max_std:* This feature is generated by applying the Equation 5 at the last convolutional layer of *basesys* backbone.

***Alert* Training:** We used a multi layer fully connected binary classifier with 50% dropout rate to train all the *alert* systems. Besides, we used binary cross entropy loss with balanced sampling to train the *alert* network.

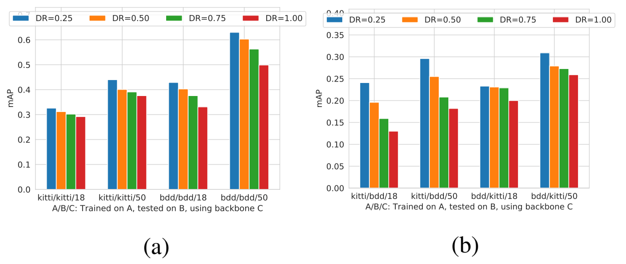

Figure 4: mAP vs declaration rate metric for the proposed *alert* system. We used four different declaration rates to calculate the corresponding mAP metric. An increasing declaration rate shows a gradual drop in the mean average precision. (a) shows the mAP vs DR metric for the similar dataset settings (b) mAP vs DR metric for cross dataset settings.

5. Evaluation and Results

5.1. AUPRC and AUROC Metrics

This section summarizes the *alert* accuracy using Area Under the Precision Recall Curve (AUPRC) and Area Under the ROC Curve (AUROC) metric. Here, we will refer to all our experimental settings using the notation *A/B/C*. It means the *basesys* and *alert* are trained on dataset *A* using backbone *C* and *alert* is used to identify *basesys* performance drop on dataset *B*. Here *C* can be 18 or 50, resembling the ResNet18 and ResNet50 backbone for the *basesys*.

Table 2 summarizes the *alert* accuracy for similar dataset settings. Our proposed *mean_max_std* feature achieves 0.781 and 0.902 as AUPRC and AUROC score, and outperforms all other features in the case of *kitti/kitti/18*. For *kitti/kitti/50*, *bdd/bdd/18* and *bdd/bdd/50* experimental settings, features collected from the *basesys* performs better than all other features in terms of AUPRC and AUROC score.

The proposed *alert* system is beneficial for cross dataset settings too. Table 3 shows the AUPRC and AUROC scores for *alert* when it is used to identify *basesys* performance when deployed on an unknown environment. For *bdd/kitti/18* settings, *alert* achieves 0.790 and 0.696 as AUPRC and AUROC score respectively when used with the *mean_max_std* feature. In all cross dataset experimental settings, *mean_max_std* features outperforms all other features for identifying *basesys* performance drop.

5.2. True Warning Rate

Using the best performing feature, *mean_max_std*, we use the true warning rate metric to determine the quality of the *alert* system in raising a warning against *basesys* performance drop. Here, the warning rate is the ratio of correctly raised warning vs the total number of frames with per-frame

Table 2: Area Under the Precision Recall Curve (AUPRC) and Area Under the ROC Curve (AUROC) score for *alert* system in the similar dataset settings. Here *alert* is used to identify *basesys* performance drop in a known environment. The notation *A/B/C* denotes that *basesys* and *alert* is trained on dataset *A* using backbone *C* and *alert* is used to identify *basesys* performance drop on dataset*B*.

	kitti/kitti/18		*kitti/kitti/50*		*bdd/bdd/18*		*bdd/bdd/50*	
Feature	AUPRC	AUROC	AUPRC	AUROC	AUPRC	AUROC	AUPRC	AUROC
n_proposals	0.180	0.128	0.186	0.110	0.205	0.368	0.197	0.363
mean_conf_score	0.205	0.320	0.192	0.358	0.452	0.653	0.463	0.665
classifier	0.728	0.851	0.689	0.831	0.498	0.744	0.488	0.734
places365	0.654	0.799	0.670	0.823	0.516	0.753	0.507	0.744
layer	0.760	0.890	0.480	0.764	0.622	0.814	0.587	**0.798**
mean	0.738	0.876	0.602	0.822	0.587	0.800	0.549	0.777
max	0.756	0.887	0.673	0.819	0.621	0.811	0.587	0.790
mean_std	0.747	0.879	0.689	**0.855**	0.609	0.815	0.577	0.791
mean_max	0.777	0.898	0.708	0.841	0.627	0.818	0.587	0.793
mean_max_std	**0.781**	**0.902**	**0.712**	0.846	**0.633**	**0.820**	**0.595**	0.795

Table 3: Area Under the Precision Recall Curve (AUPRC) and Area Under the ROC Curve (AUROC) for *alert* in the cross dataset settings. Here *alert* is identifying *basesys* performance drop in an unknown environment. The notation *A/B/C* denotes that *basesys* and *alert* is trained on dataset *A* using backbone *C* and *alert* is used to identify *basesys* performance drop on dataset *B*.

	bdd/kitti/18		*bdd/kitti/50*		*kitti/bdd/18*		*kitti/bdd/50*	
Feature	AUPRC	AUROC	AUPRC	AUROC	AUPRC	AUROC	AUPRC	AUROC
n_proposals	0.558	0.381	0.736	0.447	0.531	0.465	0.538	0.497
mean_conf_score	0.641	0.507	0.786	0.566	0.675	0.623	0.730	0.710
classifier	0.783	0.629	0.818	0.483	0.672	0.663	0.557	0.578
places365	0.781	0.624	0.821	0.493	0.857	0.837	0.630	0.637
layer	0.780	0.684	0.815	0.580	0.868	0.836	0.662	0.670
mean	0.754	0.665	0.809	0.563	0.858	0.831	0.733	0.753
max	0.778	0.682	0.813	0.582	0.647	0.605	0.661	0.686
mean_std	0.759	0.672	0.815	0.569	0.855	0.823	0.751	0.768
mean_max	0.786	0.692	0.822	0.586	0.826	0.786	0.701	0.726
mean_max_std	**0.790**	**0.696**	**0.822**	**0.587**	**0.883**	**0.856**	**0.833**	**0.825**

mAP below the critical threshold. Table 4 shows the true warning rate raised by the *alert* system.

The results in Table 4 show that in *cross dataset* settings the true warning rate is higher than the *similar dataset* settings. As *basesys* accuracy drops in cross dataset settings (Table 1), *alert* becomes more useful in these cases by identifying the critical cases. When the detector with ResNet50 backbone is trained on BDD and tested on Kitti, *alert* can identify 81.4% of the frames where *basesys* per-frame mAP is lower than the critical threshold. Figure 5 displays multiple samples of *alert* raising the alarm and flagging frames where *basesys* performance drops below a critical threshold of 0.5. The frames show conditions such as night, rain, cluttered environments.

Table 4: The true warning rate of the *alert* system to identify *basesys* performance drop.

ResNet18			ResNet50		
training dataset	testing dataset	mAP	training dataset	testing dataset	mAP
kitti	kitti	59.1%	kitti	kitti	66.0%
kitti	bdd	81.8%	kitti	bdd	78.7%
bdd	kitti	79.3%	bdd	kitti	81.4%
bdd	bdd	55.4%	bdd	bdd	52.4%

5.3. Risk-Averse Metric

In Risk-Averse Metric (RAM) [42], we evaluate *alert*'s capability to trade-off the risk of making an incorrect decision without making a decision at all. RAM gives *basesys* $+1.0$ and -0.5 respectively for a correct and incorrect prediction. *basesys* will get 0 point if it does not make any decision considering the warning raised by *alert*. For crucial

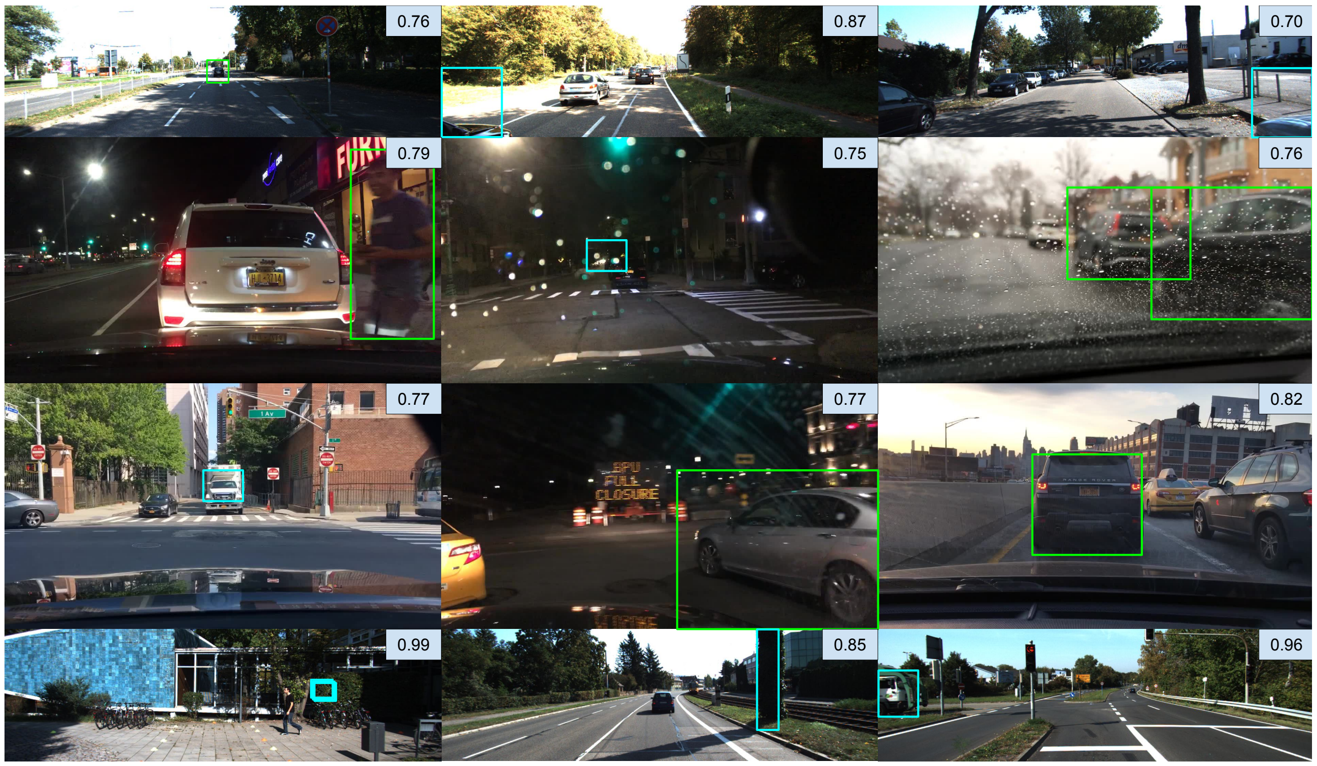

Figure 5: Examples of *alert* prediction to identify *basesys* performance drop. Here the Green and Cyan bounding boxes show the false negative and false positive errors respectively made by an object detector. *Alert* prediction is shown at the upper right corner of each image. The first row shows samples from the *kitti/kitti/50* experimental settings. The second, third and fourth row show samples from the *bdd/bdd/18*, *kitti/bdd/18* and *bdd/kitti/50* experiments.

systems like self-driving car's object detection, we expect *basesys* to trade-off incorrect decision for no decision. In such case, *basesys* can handover its control to some more competent systems. Figure 3a shows the point per image earned by *basesys* when it operates with and without considering the warning raised by *alert* in similar dataset settings. In all cases, *basesys* earns more points per image if it abstains from making an incorrect decision taking *alert*'s warning in consideration. Figure 3b shows the RAM metric for cross dataset settings. These experiments show that the warning raised by *alert* is crucial to mitigate the incorrect detection made by an object detector.

5.4. mAP vs Declaration Rate Metric

In this section we evaluate the *basesys* accuracy score for different declaration rates (DR) [42]. This metric shows the correlation between *alert* confidence and *basesys* performance. Here, the declaration rate is the proportion of images on which *basesys* operates. The rest of the images are discarded assuming that *basesys* per-frame mAP will be lower than the critical threshold on those images. To calculate this metric, we first sort the images in the ascending order of *alert* confidence. Next, the mAP of the top DR percentage of images is computed to plot the mAP vs DR metric. For a perfect *alert* the mAP for low DR images would be very high and decrease gracefully as DR approaches to 1.0. In Figure 4a we show the mAP score for four different declaration rates in similar dataset settings. The mAP score drops gradually with the increasing declaration rate. Figure 4b shows the mAP vs DR metric for cross dataset settings. In both cases, we use *mean_max_std* features in *alert* to identify *basesys* performance drop.

6. Conclusion

Deep learning-based object detection is a critical component of a wide variety of robotic applications, from autonomous vehicle to warehouse automation due to its accuracy and efficiency. However, its performance is a function of the deployment conditions and could drop below a critical threshold leading to increased risk. Although there is always room to improve accuracy and speed, safety is still a significant concern that should not be overlooked. To this end, we presented an introspection approach to performance monitoring of deep learning based object detection. We showed that our approach can improve safety by raising an alarm when per-frame mean average precision is detected to drop below a critical. We also showed that internal features from the detector itself could be used to predict when per-frame mAP degrade. Our results showed quantitatively the ability of our method to reduce risk by trading off making an incorrect detection with raising the alarm and absenting from detection.

References

[1] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *ArXiv*, abs/2004.10934, 2020.

[2] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11621–11631, 2020.

[3] Zhaowei Cai and Nuno Vasconcelos. Cascade r-cnn: Delving into high quality object detection. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6154–6162, 2018.

[4] Nicolas Carion, F. Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander M Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. *ArXiv*, abs/2005.12872, 2020.

[5] Bowen Cheng, Yunchao Wei, Humphrey Shi, Rogério Schmidt Feris, Jinjun Xiong, and Thomas S. Huang. Decoupled classification refinement: Hard false positive suppression for object detection. *ArXiv*, abs/1810.04002, 2018.

[6] Charles Corbière, Nicolas Thome, Avner Bar-Hen, Matthieu Cord, and Patrick Pérez. Addressing failure prediction by learning model confidence. In *Advances in Neural Information Processing Systems*, pages 2902–2913, 2019.

[7] Shreyansh Daftry, Sam Zeng, J Andrew Bagnell, and Martial Hebert. Introspective perception: Learning to predict failures in vision systems. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1743–1750. IEEE, 2016.

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[9] Terrance Devries and Graham W. Taylor. Leveraging uncertainty estimates for predicting segmentation quality. *ArXiv*, 1807.00502, 2018.

[10] Kaiwen Duan, Song Bai, Lingxi Xie, Honggang Qi, Qingming Huang, and Qi Tian. Centernet: Keypoint triplets for object detection. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6568–6577, 2019.

[11] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *ICML*, 2016.

[12] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[13] Hugo Grimmett, Rudolph Triebel, Rohan Paul, and Ingmar Posner. Introspective classification for robot perception. *The International Journal of Robotics Research*, 35(7):743–762, 2016.

[14] Corina Gurau, Dushyant Rao, Chi Hay Tong, and Ingmar Posner. Learn from experience: Probabilistic prediction of perception performance to avoid failure. *The International Journal of Robotics Research*, 37:981 – 995, 2018.

[15] Kaiming He, Ross B. Girshick, and Piotr Dollár. Rethinking imagenet pre-training. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 4917–4926, 2019.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[17] Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *ArXiv*, abs/1610.02136, 2017.

[18] Humphrey Hu and George Kantor. Introspective evaluation of perception performance for parameter tuning without ground truth. In *Robotics: Science and Systems*, 2017.

[19] Po-Yu Huang, Wan Ting Hsu, Chun-Yueh Chiu, Ting-Fan Wu, and Min Sun. Efficient uncertainty estimation for semantic segmentation in videos. In *ECCV*, 2018.

[20] Nataraj Jammalamadaka, Andrew Zisserman, Marcin Eichner, Vittorio Ferrari, and C. V. Jawahar. Has my algorithm succeeded? an evaluator for human pose estimators. In *ECCV*, 2012.

[21] Heinrich Jiang, Been Kim, and Maya R. Gupta. To trust or not to trust a classifier. In *NeurIPS*, 2018.

[22] Zeming Li, Chao Peng, Gang Yu, Xiangyu Zhang, Yangdong Deng, and Jian Sun. Detnet: A backbone network for object detection. *ArXiv*, abs/1804.06215, 2018.

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

[24] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

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

[26] Huizi Mao, Xiaodong Yang, and William J Dally. A delay metric for video object detection: What average precision fails to tell. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 573–582, 2019.

[27] Dimity Miller, Lachlan Nicholson, Feras Dayoub, and Niko Sünderhauf. Dropout sampling for robust object detection in open-set conditions. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–7, 2018.

[28] Sina Mohseni, Mandar Pitale, Vasu Singh, and Zhangyang Wang. Practical solutions for machine learning safety in autonomous vehicles. In *SafeAI@AAAI*, 2020.

[29] Aaron Christopher Morris. *Robotic introspection for exploration and mapping of subterranean environments*. PhD thesis, Carnegie Mellon University, The Robotics Institute, 2007.

[30] Jonah Philion, Amlan Kar, and Sanja Fidler. Learning to evaluate perception models using planner-centric metrics. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[31] Joaquin Quionero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D Lawrence. *Dataset shift in machine learning*. The MIT Press, 2009.

[32] Sadegh Rabiee and Joydeep Biswas. Ivoa: Introspective vision for obstacle avoidance. *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1230–1235, 2019.

[33] Quazi Marufur Rahman, Niko Sünderhauf, and Feras Dayoub. Did you miss the sign? a false negative alarm system for traffic sign detectors. *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3748–3753, 2019.

[34] Manikandasriram Srinivasan Ramanagopal, Cyrus Anderson, Ram Vasudevan, and Matthew Johnson-Roberson. Failing to learn: Autonomously identifying perception failures for self-driving cars. *IEEE Robotics and Automation Letters*, 3:3860–3867, 2018.

[35] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[36] David Snyder, Daniel Garcia-Romero, Daniel Povey, and Sanjeev Khudanpur. Deep neural network embeddings for text-independent speaker verification. In *Interspeech*, pages 999–1003, 2017.

[37] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. Fcos: Fully convolutional one-stage object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 9627–9636, 2019.

[38] Rudolph Triebel, Hugo Grimmett, Rohan Paul, and Ingmar Posner. Driven learning for driving: How introspection improves semantic mapping. In *ISRR*, 2013.

[39] Pei Wang and Nuno Vasconcelos. Towards realistic predictors. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 36–51, 2018.

[40] Fisher Yu, Wenqi Xian, Yingying Chen, Fangchen Liu, Mike Liao, Vashisht Madhavan, and Trevor Darrell. Bdd100k: A diverse driving video database with scalable annotation tooling. *arXiv preprint arXiv:1805.04687*, 2(5):6, 2018.

[41] Longfei Zhang and Yanming Guo. Delving into fully convolutional networks activations for visual recognition. In *ICMIP 2018*, 2018.

[42] Peng Zhang, Jiuling Wang, Ali Farhadi, Martial Hebert, and Devi Parikh. Predicting failures of vision systems. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3566–3573, 2014.