Journal of Signal Processing Systems (2023) 95:1455–1468
https://doi.org/10.1007/s11265-023-01893-5

Efficient Performance Prediction of End-to-End Autonomous Driving Under Continuous Distribution Shifts Based on Anomaly Detection

Siyu Luan[1] · **Zonghua Gu**[1] · **Shaohua Wan**[2]

Received: 21 July 2023 / Revised: 25 August 2023 / Accepted: 3 September 2023 / Published online: 20 November 2023

Abstract
A Deep Neural Network (DNN)'s prediction may be unreliable outside of its training distribution despite high levels of accuracy obtained during model training. The DNN may experience different degrees of accuracy degradation for different levels of distribution shifts, hence it is important to predict its performance (accuracy) under distribution shifts. In this paper, we consider the end-to-end approach to autonomous driving of using a DNN to map from an input image to the control action such as the steering angle. For each input image with possible perturbations that cause distribution shifts, we design a Performance Prediction Module to compute its anomaly score, and use it to predict the DNN's expected prediction error, i.e., its expected deviation from the ground truth (optimal) control action, which is not available after deployment. If the expected prediction error is too large, then the DNN's prediction may no longer be trusted, and remedial actions should be taken to ensure safety. We consider different methods for computing the anomaly score for the input image, including using the reconstruction error of an Autoencoder, or applying an Anomaly Detection algorithm to a hidden layer of the DNN. We present performance evaluation of the different methods in terms of both prediction accuracy and execution time on different hardware platforms, in order to provide a useful reference for the designer to choose among the different methods.

Keywords Machine learning · Deep learning · Distribution shifts · Performance prediction · End-to-end autonomous driving

1 Introduction

Machine Learning, especially Deep Learning, has achieved revolutionary success in a variety of application domains. However, one significant challenge to the deployment of DNNs in safety-critical applications is the impact of distribution shifts on prediction accuracy. Distribution shifts refer to changes in the underlying data distributions that a model was trained on. They may occur due to different reasons, such as dataset sample selection bias, changes in the natural environment, or even adversarial attacks, and they can cause significant performance degradation when it is used to make predictions on new, unseen data. One approach to addressing the distribution shift problem is domain adaptation [1] or continual learning [2], where the DNN is re-trained (offline or online) to generalize to novel data, but re-training may not be practical for limited available data and large distribution shifts.

In this paper, we consider progressive distribution shifts [3], i.e., gradual and continuous distribution shifts. For example, the weather gradually transitions from clear skies to overcast conditions, and then to light, medium, and heavy rainfall over time. The application DNN may experience different degrees of performance degradation for different levels of distribution shifts. Instead of model re-training, we assume the DNN is pre-trained and frozen, and aim to predict its performance (in terms of accuracy) under different levels of progressive distribution shifts. We consider the end-to-end approach to autonomous driving [4] as the application context, where a DNN is used to directly map from an input image to the corresponding control action of steering angle command [5]. We present the design of a Performance Prediction Module (PPM) that outputs an estimation of the performance of the DNN, in terms of its expected prediction error relative to the ground truth, for each input image with possible perturbations, based on different methods for computing the anomaly score for the input image. We evaluate

✉ Zonghua Gu
zonghua.gu@umu.se

[1] Department of Applied Physics and Electronics, Umeå University, Umeå, Sweden

[2] Shenzhen Institute for Advanced Study, University of Electronic Science and Technology of China, Shenzhen, China

different methods in terms of both prediction accuracy and computation efficiency, measured by the PPM's execution time on two hardware platforms: a powerful workstation and an embedded device.

This paper is structured as follows: we present background and related work in Sect. 2; material and methods in Sect. 3; performance evaluation results in Sect. 4; and conclusions in Sect. 5. Additional detailed experimental results are included in the Appendix for the interested reader.

2 Background and Related Work

2.1 End-to-End Autonomous Driving

The classic processing pipeline of autonomous driving consists of 1) Perception stage, which perceives and interprets its surroundings using sensor data from cameras, Lidars, and Radars, including the presence of other vehicles, road conditions, pedestrians, street signs, and other relevant objects and features; 2) Behavior Prediction stage, which predicts the likely future actions of detected objects, e.g., if a pedestrian is about to cross the road, or if a vehicle is about to change lanes; 3) Planning stage, which establishes a safe and efficient trajectory for the autonomous vehicle to follow, taking into account the presence of obstacles; 4) Control stage, which issues control commands for the vehicle's actuators, including throttle, brake, and steering, guiding the vehicle along the planned trajectory. Machine Learning models, especially Deep Neural Networks (DNNs) trained with Deep Learning, may be deployed in any stage in the processing pipeline. The predominant industrial practice is to deploy DNNs primarily in the Perception stage [6]. An alternative approach is end-to-end control [4], where a DNN is trained to directly map from input images to driving actions using methods such as Imitation Learning or Reinforcement Learning, bypassing the need for explicit computer vision tasks. For instance, Nvidia Net [5] is a pioneering work that employs a Convolutional Neural Network (CNN) to map raw pixels from a single front-facing camera to steering commands. The CNN is trained using Imitation Learning, where it learns from the driving behaviors of expert human drivers to navigate effectively. (We use the term DNN in this paper, but we focus on CNNs in our experiments, instead of other DNN architectures such as Transformers.) Significant progress has been made since then, and interested readers are referred to the survey paper [4] for more details.

2.2 Out of Distribution (OOD) Detection

The purpose of OOD detection is to identify test samples that fall outside of the distribution of the training (In Distribution, ID) dataset, i.e., outliers/anomalies. OOD detection is a binary classification problem of ID vs. OOD. A large number of OOD detection algorithms [7] have been developed in recent years, which can be categorized into classification-based, density-based, distance-based, and reconstruction-based methods. Our work [8] presents a framework for OOD detection based on outlier detection in one or more hidden layers of a DNN based on either Isolation Forest (IF) or Local Outlier Factor (LOF). For real-time safety-critical applications on resource-constrained hardware platforms, timing performance is also of great importance in addition to accuracy. Our recent work [9] presents a comprehensive benchmark study of well-known OOD detection algorithms in terms of both accuracy and execution time on different hardware platforms, in order to provide a useful reference for their practical deployment.

While most works on OOD detection focus on classification problems, some authors address regression problems based on Autoencoder (AE) or Variational Autoencoder (VAE), an unsupervised approach to learning a low-dimensional feature representation from unlabelled training data. Stocco et al. present SelfOracle [10] for detection of misbehaviors in autonomous driving systems, using a VAE to reconstruct a set of consecutive input images and compute the reconstruction errors. The VAE is trained on normal data, and fits a probability distribution for the observed reconstruction errors via maximum likelihood estimation, which is then used to determine a threshold value for OOD detection. Cai and Koutsoukos [11] adopt VAE and Deep Support Vector Data Description (DeepSVDD) [12] to learn models for computing the nonconformity score of input samples relative to the training set, use Inductive Conformal Anomaly Detection [13] to take into account multiple samples to improve the robustness of OOD detection.

OOD detection can be an effective safety assurance mechanism for safety–critical applications such as autonomous driving. Instead of blindly trusting the prediction results, a system equipped with an OOD detector should fail loudly upon detecting OOD input, so that remedial actions may be taken subsequently to ensure safety. However, OOD detection gives a binary classification result of ID vs. OOD for each test sample, which may be too coarse-grained. More fine-grained predictions of DNN performance may be more appropriate for continuous distribution shifts, as discussed next.

2.3 DNN Performance Prediction for Autonomous Driving

One general approach to estimating/predicting the performance of a DNN is uncertainty estimation [14], e.g., Bayesian Neural Networks, Monte-Carlo dropout, and Deep Ensembles. While uncertainty estimation techniques offer the advantage of providing quantitative uncertainty values,

their main drawback is high runtime overhead, which may render them unsuitable for deployment in real-time safety-critical applications such as autonomous driving.

Several recent papers address performance prediction of DNNs used for perception tasks in autonomous driving, e.g., semantic segmentation. Lohdefink et al. [15] use an Autoencoder, which runs in parallel with a trained DNN for semantic segmentation to estimate domain shifts during inference, by first computing the autoencoder's image reconstruction performance, measured by the Peak-Signal-to-Noise Ratio (PSNR), then computing a domain mismatch metric as the earth mover's distance between a pre-stored PSNR distribution on training (source) data, and an online-acquired PSNR distribution on any inference (target) data. However, it does not give per-image estimates. Rottmann et al. [16] propose a method for meta-classifying whether segments predicted by a semantic segmentation DNN intersect with the ground truth, by aggregating measures of dispersion for predicted pixel-wise class probability distributions, like classification entropy, that yield heat maps of the input image size. Klingner et al. [17, 18] present a multi-task learning approach with monocular depth estimation as a secondary task, which enables prediction of the DNN's performance for the primary task of semantic segmentation by evaluating the depth estimation task with a physical depth measurement provided, e.g., by a Lidar sensor. Bär et al. [19] present an approach to per-image performance prediction for semantic segmentation with no need for additional (Lidar) sensors as in [17, 18], by extending a trained semantic segmentation DNN with fixed parameters with an image reconstruction decoder, and using the image reconstruction performance as input to a regression model to predict the semantic segmentation performance.

In this paper, we address the problem of performance prediction for end-to-end autonomous driving [4] based on the input image quality (i.e., the degree/level of perturbation), which has not been considered in related works.

3 Material and Methods

3.1 Design Objective

We consider a DNN for end-to-end autonomous driving, consisting of the Convolutional Encoder (ENC), with multiple convolutional layers, followed by the prediction head (Head), containing one or more fully connected layers, and finally a single regression output neuron that outputs the steering angle command. We assume the DNN is pre-trained and frozen, e.g., with Imitation Learning as in Nvidia Net [5] to mimic expert driver actions, with the optimization objective of minimizing the L2 loss, i.e., the Mean Squared Error (MSE) $\frac{1}{N}\sum_{i=1}^{N}\left(\widehat{y}_i(0)-y_i\right)^2$ between the control action (steering angle command) $\widehat{y}_i(0)$ output by the DNN for each input image $\boldsymbol{x_i}(0)$ in the training dataset for the DNN $\{\boldsymbol{x_i}(0)|1 \le i \le N\}$ with no distribution shift, and the corresponding ground truth action y_i for $\boldsymbol{x_i}(0)$, which is assumed to be known as part of the training dataset, e.g., from expert demonstrations used for Imitation Learning [5].

For each input image $\mathbf{x}(l)$ with possible distribution shift, simulated by adding some perturbation at level l, we design a PPM to compute its anomaly score $S(\mathbf{x}(l))$, and use it to predict the DNN's performance as measured by the expected prediction error $\left(\widehat{y}(l)-y\right)^2$, i.e., the expected deviation of the predicted action $\widehat{y}(l)$ from the ground truth action y. If the expected prediction error is too large, then the DNN's prediction can no longer be trusted, and remedial actions should be taken to ensure safety, e.g., switching to a simple, backup safety controller, or alerting the human operator to take corrective actions. Intuitively, there should be a positive correlation, i.e., for a given input image $\mathbf{x}(l)$, a higher anomaly score $S(\mathbf{x}(l))$ should lead to a larger expected prediction error $\left(\widehat{y}(l)-y\right)^2$, hence the former can be used to predict the latter. To measure the degree of correlation between them, we use the Pearson Correlation Coefficient (PCC) [20], a statistical metric that measures the strength and direction of the linear dependence between two random variables. Formally, the PCC of two random variables A and B is defined as the covariance of the two variables divided by the product of their standard deviations:

$$PCC(A,B) = \frac{cov(A,B)}{\sigma_A \sigma_B} = \frac{\mathbb{E}\left[\left(A-\mu_A\right)\left(B-\mu_B\right)\right]}{\sigma_A \sigma_B}$$

The PCC value varies between -1 and +1, with value of +1 indicating perfect positive correlation; value of -1 indicating perfect negative correlation; value of 0 indicating no correlation. As the degree of correlation is highly dependent on the Anomaly Detection algorithm used for computing the anomaly score $S(\mathbf{x}(l))$, we evaluate different Anomaly Detection algorithms to identify those that result in high PCC between $S(\mathbf{x}(l))$ and $\left(\widehat{y}(l)-y\right)^2$ for deployment.

3.2 Detailed Design of Our Framework

Figure 1 shows the detailed design of our proposed framework. (Either the PyOD module or the DEC module is present in the PPM as a design choice, but not both.) We choose a perturbation type, and a perturbation level $l \in [0, L]$ for each perturbation type ($l = 0$ denotes no perturbation, and $l = L$ denotes the highest level of perturbation), and construct a sequence of input images $\boldsymbol{X}(l) = \{\boldsymbol{x_i}(l)|i = 1 \ldots K\}$ with perturbation level l, based on the sequence of original input images $\boldsymbol{X}(0) = \{\boldsymbol{x_i}(0)|i = 1 \ldots K\}$ with no perturbation. For each input image $\boldsymbol{x_i}(l)$, we record the prediction error $\left(\widehat{y}_i(l)-y_i\right)^2$ between the DNN-predicted action $\widehat{y}_i(l)$

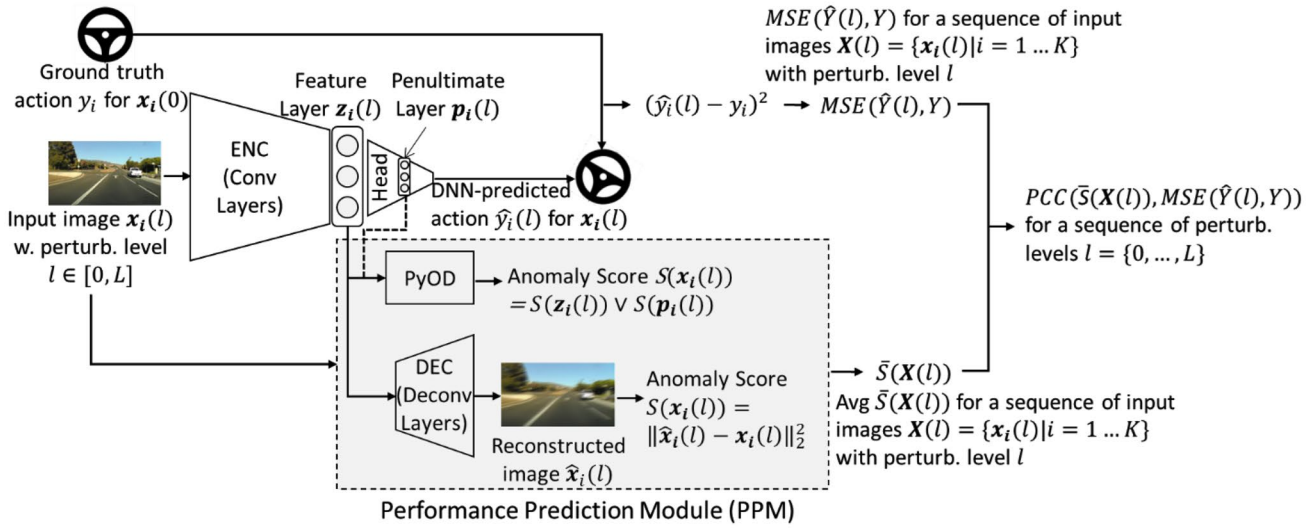


Figure 1 Our framework for performance prediction of the DNN for end-to-end autonomous driving.

and ground truth action y_i. The PPM (shown in the shaded box in Fig. 1) is used to compute the anomaly score $S(\boldsymbol{x_i}(l))$ for input image $\boldsymbol{x_i}(l)$ with one of two approaches:

1) by applying an Anomaly Detection algorithm (e.g., using the software package PyOD [21]) to either the Feature Layer (FL) $\boldsymbol{z_i}(l)$ (the last layer of ENC), or the Penultimate Layer (PL) $\boldsymbol{p_i}(l)$ of the prediction head, which contains the highest-level representation that is used to compute the final regression output, i.e., $S\left(\boldsymbol{x_i}(l)\right)=S(\boldsymbol{z_i}(l)) \vee S(\boldsymbol{p_i}(l))$; or
2) by using an Autoencoder, with the Decoder (DEC) consisting of multiple deconvolutional layers to reconstruct the input image based on the FL $\boldsymbol{z_i}(l)$, which is the bottleneck layer of the Autoencoder (ENC+DEC). The anomaly score is computed as the pixel-wise reconstruction error of the Autoencoder, i.e., $S\left(\boldsymbol{x_i}(l)\right) = \|\hat{\boldsymbol{x}}_i(l) - \boldsymbol{x_i}(l)\|_2^2$, where the operator $\|\|_2^2$ denotes the square of the 2-norm of the pixel-wise differences between the reconstructed image $\hat{\boldsymbol{x}}_i(l)$ and the original image $\boldsymbol{x_i}(l)$. The Autoencoder is pre-trained with this loss function and frozen, similar to the DNN (ENC+Head).

For the first approach, we may choose to use either the FL $\boldsymbol{z_i}(l)$ or the PL $\boldsymbol{p_i}(l)$ for extracting the low-dimensional latent representation from each input image as input to PyOD. Note that $\boldsymbol{z_i}(l)$ and $\boldsymbol{p_i}(l)$ may or may not be the same layer, e.g., they are different for Nvidia Net [5], whose prediction head consists of 3 fully connected layers plus an output neuron with dimensions 100-50-10-1; whereas they are the same for ResNet [22], whose prediction head consists of one fully connected layer plus an output neuron with dimensions 1000-1. For the second approach, we always use $\boldsymbol{z_i}(l)$ as the bottleneck layer of the Autoencoder, if $\boldsymbol{z_i}(l)$ and $\boldsymbol{p_i}(l)$ are different layers, as this results in higher-quality reconstructions than using $\boldsymbol{p_i}(l)$, which has a very low dimension.

During offline design and evaluation phase, for the sequence of K input images with perturbation level l, $\boldsymbol{X}(l) = \{\boldsymbol{x_i}(l) | i = 1 \dots K\}$, we compute the corresponding sequence of K DNN-predicted actions $\hat{Y}(l) = \{\hat{y}_i(l) | i = 1 \dots K\}$. We then compute:

1) $\bar{S}(\boldsymbol{X}(l)) = \frac{1}{K}\sum_{i=1}^{K} S(\boldsymbol{x_i}(l))$, the average anomaly score of the sequence of input images $\boldsymbol{X}(l)$, using one of the two approaches described earlier; and
2) $MSE\left(\hat{Y}(l), Y\right) = \frac{1}{K}\sum_{i=1}^{K}\left(\hat{y}_i(l) - y_i\right)^2$, the Mean Squared Error between the sequence of DNN-predicted actions $\hat{Y}(l)$ for the sequence of input images $\boldsymbol{X}(l)$, and the sequence of ground truth actions $Y = \{y_i | i = 1 \dots K\}$ for the sequence of original input images with no perturbation $\boldsymbol{X}(0) = \{\boldsymbol{x_i}(0) | i = 1 \dots K\}$.

Finally, we compute the PCC between $\bar{S}(\boldsymbol{X}(l))$ and $MSE(\hat{Y}(l), Y)$ for the sequence of $L + 1$ perturbation levels $l = \{0, \dots, L\}$ as a metric to evaluate the prediction accuracy of PPM, i.e., whether the anomaly score $S\left(\boldsymbol{x_i}(l)\right)$ is a good predictor of the DNN's performance for input image $\boldsymbol{x_i}(l)$ measured by the expected prediction error $\left(\hat{y}_i(l) - y_i\right)^2$, when the ground truth action y_i is unknown at test time. Based on the evaluation results of different methods for computing the anomaly score, a method with high accuracy (as measured by the PCC) should be chosen for final deployment. After deployment, at test time, the anomaly score $S(\boldsymbol{x}(l))$ for any given input image $\boldsymbol{x}(l)$ (with any possible type and level of perturbation) is computed and used to predict the DNN's performance for $\boldsymbol{x}(l)$.

3.3 Selected Anomaly Detection Algorithms in PyOD

For the design of PPM that applies an Anomaly Detection algorithm to either $\boldsymbol{z}_i(l)$ or $\boldsymbol{p}_i(l)$, as discussed in Sect. 3.2, we use the software package PyOD [21], an open-source Python toolbox for performing scalable Anomaly Detection on multivariate data that provides a wide range of Anomaly Detection algorithms. We consider 14 Anomaly Detection algorithms in PyOD in our experiments:

1) Principal Component Analysis (PCA) [23]. PCA is a linear dimensionality reduction technique, using Singular Value Decomposition to project the input sample to a lower dimensional space, and then use the reconstruction error as the anomaly score.
2) One-Class SVM (OCSVM) [24]. OCSVM maximizes the margin between the origin and the normal samples, and defines the decision boundary as the hyperplane that determines the margin.
3) Local Outlier Factor (LOF) [8, 25]. LOF measures the local deviation of density of a given sample w. r. t. its neighbors.
4) Clustering-Based Local Outlier Factor (CBLOF) [26]. CBLOF calculates the anomaly score by first assigning samples to clusters, and then using the distance among clusters as anomaly scores.
5) Connectivity-Based Outlier Factor (COF) [27]. COF uses the ratio of average chaining distance of data points and the average of average chaining distance of k-th nearest neighbor of the data point, as the anomaly score for observations.
6) Histogram-Based Outlier Detection (HBOS) [28]. HBOS assumes the feature independence and calculates the anomaly score by building histograms.
7) K-Nearest Neighbors (KNN) [29]. KNN views the anomaly score of the input instance as the distance to its k-th nearest neighbor.
8) Subspace Outlier Detection (SOD) [30]. SOD aims to detect outliers in varying subspaces of a high-dimensional feature space.
9) Copula Based Outlier Detector (COPOD) [31]. COPOD is a hyperparameter-free outlier detection algorithm based on empirical copula models.
10) Empirical-Cumulative-distribution-based Outlier Detection (ECOD) [32]. ECOD is a hyperparameter-free outlier detection algorithm based on Empirical CDF functions, which is used to estimate the density of each feature independently, assuming that outliers lie on the tails of the distribution.
11) Deep Support Vector Data Description (DeepSVDD) [12]. DeepSVDD trains a neural network while minimizing the volume of a hypersphere that encloses the network representations of the data, forcing the network to extract the common factors of variation.
12) Deep Autoencoding Gaussian Mixture Model (DAGMM) [33]. DAGMM utilizes a deep autoencoder to generate a low-dimensional representation and reconstruction error for each input data point, which is further fed into a Gaussian Mixture Model (GMM).
13) Lightweight Online Detector of Anomalies (LODA) [34]. LODA is an ensemble method and is particularly useful in domains where a large number of samples need to be processed in real-time or in domains where the data stream is subject to concept drift and the detector needs to be updated online.
14) Isolation Forest (IForest) [8, 35]. IForest isolates observations by randomly selecting a feature and then randomly selecting a split value between the maximum and minimum values of the selected feature.

4 Performance Evaluation

4.1 Experimental Setup

We consider a real-world driving dataset A2D2: Audi autonomous driving Dataset [36]. We adopt a similar approach to [37] for dataset construction. We use the image sequences recorded in Gaimersheim, which consist of ~15,000 images with 30 FPS, down-sampled to 15 FPS to reduce similarities between adjacent frames, and aligned with ground truth steering command labels. To simulate distribution shifts, we consider 9 basic types of perturbations, including Gaussian blur (referred to as blur), Gaussian noise (referred to as noise), radial distortion (referred to as distortion), three-color channels RGB (Red, Green, Blue), and HSV (Hue, Saturation, and Value). In addition, we also include 6 other types of perturbations from ImageNet-C [38], which are snow, fog, frost, motion blur, zoom blur, and jpeg compression, denoted with the prefix IMGC_, e.g., IMGC_snow, IMGC_fog, etc. For each type of perturbation, we generate $L = 5$ levels of varying intensity, with a total of 6 levels of perturbation intensity (from 0 to 5), and apply the same perturbation and level to all images within the A2D2 dataset. We set the length of the image sequence $K = N$ in the experiments, i.e., the entire training dataset for the DNN.

We consider two DNN architectures based on [37], trained with Imitation Learning to perform end-to-end driving by mapping from the input image to the steering angle: Nvidia Net [5] and ResNet-152 [22]. The Nvidia Net architecture consists of 9 layers, including a normalization layer, 5 convolutional layers, 3 fully connected layers, plus a single output neuron for regression of the steering angle. The ResNet-152 architecture is based on the backbone of the original ResNet-152 [22], but with a single output neuron for regression. For both DNNs, we adopt the same decoder (DEC) as a 5-layer deconvolutional network with a symmetric architecture to the encoder (ENC)

Table 1 Hardware configuration of two hardware platforms.

HW Platform	Configuration
DL Workstation (WS)	CPU: Intel(R) Core(TM) i9-13900KF 24-Core Processor. Clock speed: 3.0 GHz
	GPU: NVIDIA GeForce RTX 4090
	CPU memory: 128 GB, GPU memory: 24 GB
Nvidia Jetson Nano 4 GB (JSN)	CPU: Quad-core ARM A57 Processor. Clock speed: 1.43 GHz
	GPU: 128-core Maxwell GPU
	Unified memory: 4 GB

of Nvidia Net. (Note that the ToTensor() method in PyTorch DataLoader automatically converts all pixel values of input images into the range [0, 1], which can be compared with the output of the decoder, also in the range [0, 1].)

We consider two hardware platforms, one powerful Deep Learning workstation (abbreviated as WS), and one resource-constrained embedded device Nvidia Jetson Nano 4 GB (abbreviated as JSN), as shown in Table 1. The workstation has separate CPU memory of size 128 GB and GPU memory of size 24 GB; the Jetson Nano has a unified memory of size 4 GB shared between its CPU and GPU. We use the following abbreviations: JSNCPU for Jetson Nano CPU; JSNGPU for Jetson Nano GPU; WSCPU for Workstation CPU; WSGPU for Workstation GPU. For each timing measurement study, we measure the total execution time of the test dataset and divide it by the dataset size to obtain the per-instance execution time.

4.2 Performance Prediction Accuracy

We consider the following methods for computing the anomaly score $S\left(\boldsymbol{x}_i(l)\right)$ for input image $\boldsymbol{x}_i(l)$ at perturbation level l:

1. **AE**: Anomaly score computed as reconstruction error of the Autoencoder, consisting of encoder ENC and decoder DEC, with the FL $\boldsymbol{z}_i$(l) as the bottleneck layer between them.
2. **FL for PyOD**: Anomaly score computed with one of the 14 Anomaly Detection algorithms in PyOD, with the FL $\boldsymbol{z}_i(l)$ as input.
3. **PL for PyOD**: Anomaly score computed with one of the 14 Anomaly Detection algorithms in PyOD, with the PL $\boldsymbol{p}_i(l)$ as input. (Recall that $\boldsymbol{p}_i(l)$ and $\boldsymbol{z}_i(l)$ are different layers for Nvidia Net, but they are the same layer for ResNet-152.)

Figure 2 plots the PCC values averaged across all different types of perturbations and all levels of perturbations, for the following design choices:

1. Nvidia Net (AE, or FL for PyOD).
2. Nvidia Net (PL for PyOD).
3. ResNet-152 (AE, or FL/PL for PyOD).

The AE-based method is placed in the leftmost position in the figures, and the other PyOD-based methods are sorted (from left to right) by decreasing average PCC value across all three configurations Nvidia Net (FL for PyOD), Nvidia Net (PL for PyOD), and ResNet-152 (FL/PL for PyOD).

4.3 Timing Performance

In addition to accuracy, it is important to consider the timing performance on embedded systems platforms with limited hardware resources in terms of CPU speed and memory size, due to SWaP (Size, Weight, and Power) considerations [39]. We consider the following allocations of different components in our framework shown in Fig. 1 to CPU or GPU, For maximum efficiency, ENC, Head, and DEC are always kept together and assigned to either CPU or GPU. PyOD is always assigned to the CPU due to its requirement for certain libraries on the CPU. Table 2 illustrates the four components for timing measurements, including DNN inference and the different components of the PPM:

1. DNN Inference (ENC+Head) is assigned to either CPU or GPU.
2. DEC of AE is assigned to either CPU or GPU, and is always co-located with DNN Inference (ENC+Head).
3. PyOD, with either FL or PL as input, is always assigned to the CPU.

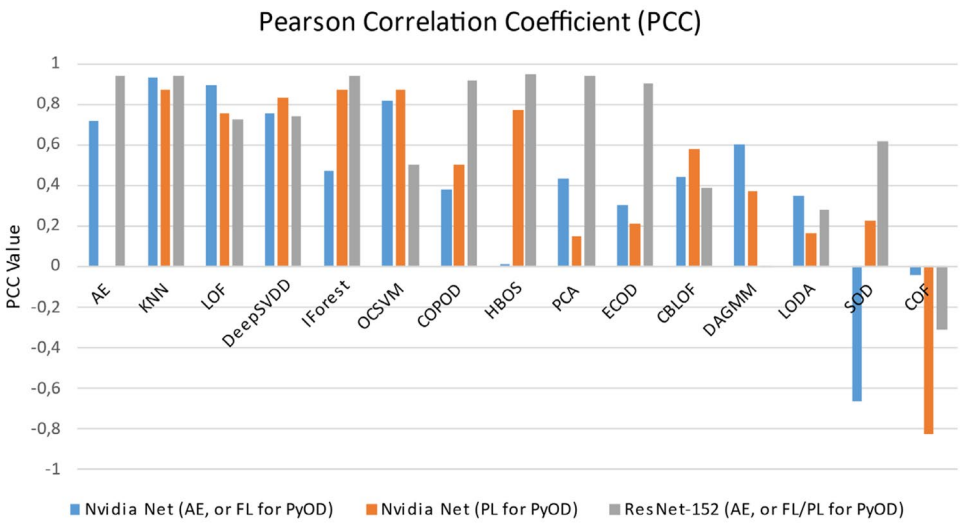


Figure 2 PCC of different performance prediction methods.

For each DNN, either Nvidia Net or ResNet-152, we measure both the total execution time and the DNN forward inference time (their difference is the runtime overhead of PPM):

1. *Total Execution Time (AE, or FL for PyOD):* for the AE-based method, it is the sum of execution times of (1) and (2) in Table 2; for each of the 14 PyOD-based methods, it is equal to the sum of execution times of (1) and (3) in Table 2.
2. *Total Execution Time (PL for PyOD):* for each of the 14 PyOD-based methods, it is the sum of execution times of (1) and (4) in Table 2.
3. *DNN Inference Time:* execution time of (1) only in Table 2.

Figure 3 shows the timing performance results for Nvidia Net, and Fig. 4 shows the results for ResNet-152, with DNN assigned to different hardware platforms. The DNN inference time is plotted as a horizontal line for comparison purposes. Since FL and PL are the same layer for ResNet-152, components (3) and (4) in Table 2 are the same for ResNet-152, hence Fig. 4 contains two timing measurement entries instead of three entries in Fig. 3 for Nvidia Net.

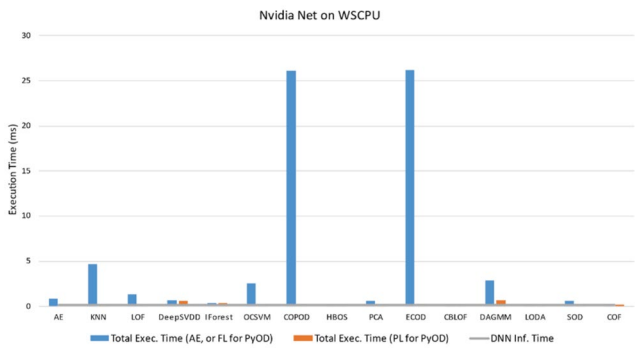
Table 2 Four components for timing measurements.

	DNN on CPU	DNN on GPU
(1) DNN Inference (ENC+Head)	CPU	GPU
(2) DEC of AE	CPU	GPU
(3) PyOD w. FL as input	CPU	CPU
(4) PyOD w. PL as input	CPU	CPU

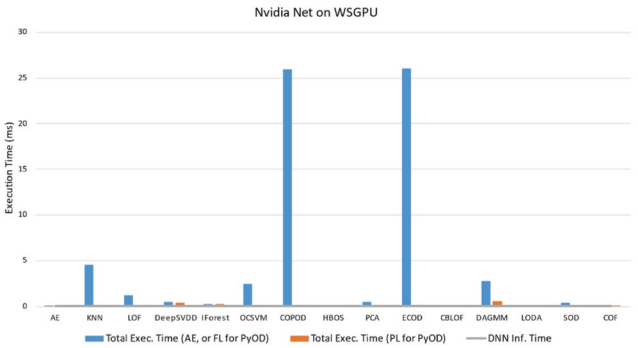
4.4 Results and Analysis

We make the following observations from Figs. 2, 3 and 4:

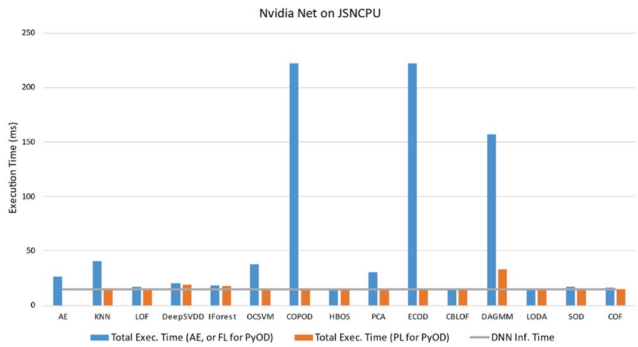
1) The most accurate methods are AE, KNN, LOF, and DeepSVDD, which achieve relatively high PCC values averaged across all types and levels of perturbations. Among them, AE achieves both high accuracy and relatively short execution time, especially on the GPU, hence it is the most competitive alternative overall. KNN has the highest accuracy on average, but it has significantly longer execution times than the other three methods for Nvidia Net. (It has comparable or smaller execution times for ResNet-152.) Two Anomaly Detection algorithms, SOD and COF, have the worst accuracy, with negative PCC values for some configurations.
2) Since Nvidia Net is a much smaller network than ResNet-152, its inference time is quite short, e.g., it is almost negligible relative to the PPM on WSCPU and WSGPU, and noticeable but insignificant relative to the PPM on JSNCPU and JSNGPU. On the other hand, the DNN inference time for ResNet-152 is quite significant. Hence the relative runtime overhead of PPM is higher for the smaller Nvidia Net than for the larger ResNet-152.
3) GPU helps to achieve significant acceleration of both DNN inference (ENC+Head) and decoder of the AE (DEC). The PyOD-based Anomaly Detection algorithms do not benefit from GPU acceleration since they can only execute on the CPU. The most time-consuming Anomaly Detection algorithms are COPOD and ECOD, which also have relatively poor prediction accuracy measured by PCC.



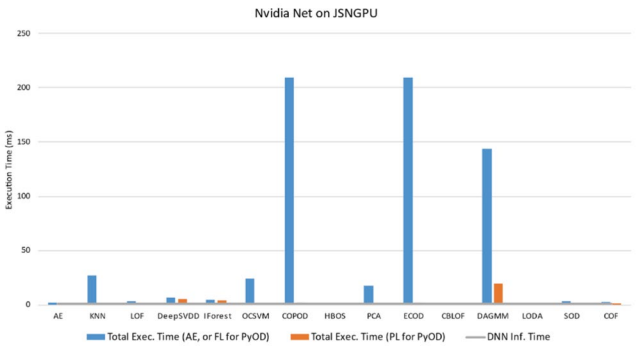
(a) Nvidia Net on WSCPU



(b) Nvidia Net on WSGPU

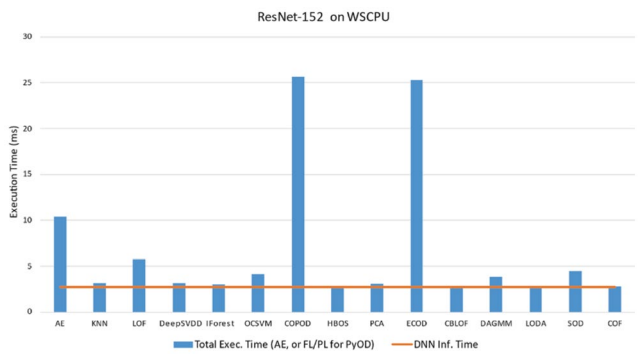


(c) Nvidia Net on JSNCPU

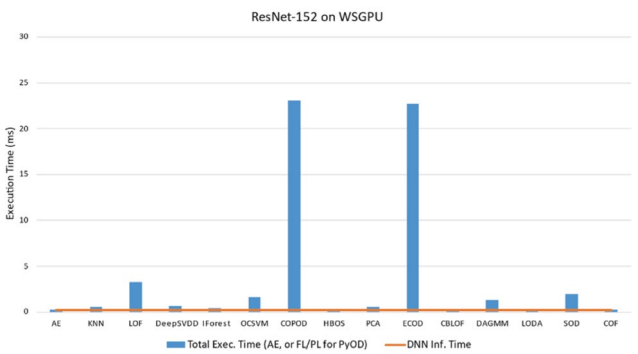


(d) Nvidia Net on JSNGPU

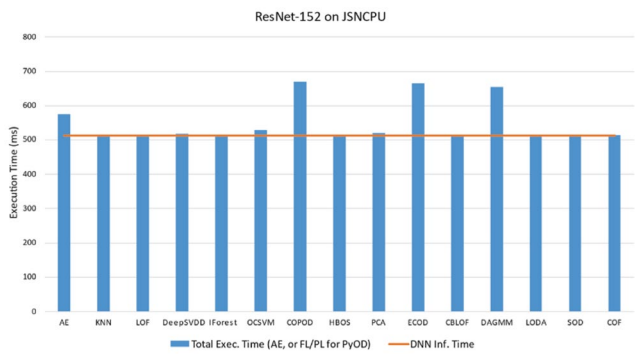
Figure 3 Execution time measurements for Nvidia Net.



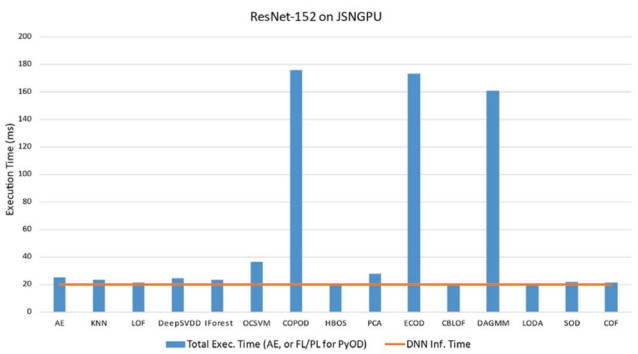
(a) ResNet-152 on WSCPU



(b) ResNet-152 on WSGPU



(c) ResNet-152 on JSNCPU



(d) ResNet-152 on JSNGPU

Figure 4 Execution time measurements for ResNet-152.

Considering both performance prediction accuracy and timing performance, we conclude that AE is the most competitive method overall, especially if the hardware platform is equipped with a GPU that can help achieve significant acceleration of AE, including both the encoder (ENC) and the decoder (DEC).

5 Conclusions

In this paper, we consider the end-to-end approach to autonomous driving of using a DNN to map from an input image to the corresponding control action of steering angle command, and present performance prediction methods that output an estimation of the accuracy of steering angle prediction relative to the ground truth, based on different methods for computing the anomaly score for the input image with possible perturbations. We evaluate the methods for both prediction accuracy and computation efficiency for two representative DNNs, a smaller-size DNN Nvidia Net and a more realistic, larger-size DNN ResNet-152. The results may provide a useful reference for the designer to choose among the different Anomaly Detection algorithms for DNN performance prediction.

Appendix

We present the detailed PCC values for different methods and input image disturbances, for Nvidia Net (Tables 3 and 4), and ResNet-152 (Table 5). The last row in each table denotes the average value across all the different types of perturbations, which are plotted in Fig. 2. Bold font denotes the highest value in each row. Different methods exhibit clear performance differences for different DNNs and different types of perturbations in terms of prediction accuracy measured by PCC, hence there is no one-size-fits-all method that works well for all perturbations. For example, the best-performing methods are KNN (with average PCC of 0.931) in Table 3, OCSVM (with average PCC of 0.873) in Table 4, and HBOS (with average PCC of 0.944) in Table 5. The worst-performing methods are SOD with PCC of -0.666 in Table 3, COF with PCC of -0.824 in Table 4, and COF with PCC of -0.312 in Table 5. However, the results support the general conclusions in Sect. 4.4 regarding the average accuracy of different methods for DNN performance prediction.

Table 3 PCC values for different methods and input image disturbances, for Nvidia Net (FL for PyOD).

	AE	KNN	LOF	DSVDD	IForest	OCSVM	COPOD	HBOS	PCA	ECOD	CBLOF	DAGMM	LODA	SOD	COF
B_darker	0.979	0.970	0.852	0.964	**0.999**	0.759	0.996	-0.439	0.997	0.998	0.995	0.987	0.996	-0.981	0.118
B_lighter	**0.997**	0.927	0.829	0.954	0.926	0.706	0.955	0.539	0.986	0.978	0.987	0.981	0.974	-0.986	-0.863
G_darker	0.994	0.954	0.790	0.082	0.999	0.727	**1.000**	-0.345	0.999	**1.000**	0.998	0.976	0.998	-0.968	0.949
G_lighter	0.975	0.969	0.984	0.989	0.983	0.861	0.997	0.861	0.999	**1.000**	0.998	0.997	0.999	-0.992	0.534
H_darker	0.953	0.907	0.904	0.836	0.994	0.805	0.998	0.300	**1.000**	0.989	0.995	0.977	0.999	-0.986	-0.702
H_lighter	0.935	0.972	0.935	0.916	0.950	0.853	0.984	0.334	0.983	0.973	0.984	0.297	**0.988**	-0.864	-0.796
IMGC_fog	**0.996**	0.874	0.922	0.907	-0.459	0.853	-0.578	-0.623	-0.533	-0.585	-0.409	0.717	-0.537	-0.062	-0.960
IMGC_frost	0.870	0.926	0.940	0.791	0.734	0.832	-0.306	0.904	0.463	-0.750	0.178	**0.992**	-0.306	-0.830	0.902
IMGC_jpeg	0.338	0.969	**0.995**	0.939	0.955	0.966	0.889	0.721	0.935	0.702	0.873	-0.955	0.828	0.330	-0.553
IMGC_motion_blur	-0.430	0.953	**0.977**	0.938	0.525	0.935	-0.547	0.736	-0.527	-0.626	-0.418	0.920	-0.899	0.909	-0.888
IMGC_snow	**0.922**	0.858	0.904	0.836	-0.992	0.697	-0.975	-0.991	-0.963	-0.964	-0.959	0.529	-0.968	-0.982	-0.609
IMGC_zoom_blur	-0.595	0.989	**0.999**	0.997	-0.966	0.983	-0.934	-0.917	-0.947	-0.940	-0.896	0.760	-0.944	-0.969	0.928
R_darker	0.949	0.956	0.954	0.800	0.975	0.782	0.995	0.682	0.995	0.995	0.994	**0.996**	0.991	-0.988	-0.610
R_lighter	0.961	0.968	0.840	0.211	0.998	0.768	0.998	-0.302	0.995	**1.000**	0.991	0.985	0.996	-0.967	0.730
S_darker	0.966	0.868	0.872	0.858	**0.991**	0.751	0.827	0.855	0.965	0.029	0.976	-0.829	0.767	-0.998	-0.793
S_lighter	0.923	0.980	0.964	0.914	0.999	0.806	**1.000**	-0.838	0.993	0.999	0.996	0.766	0.995	-0.812	0.923
V_darker	**0.953**	0.805	0.391	-0.284	0.879	0.668	0.732	-0.939	0.890	0.740	0.743	0.913	0.644	-0.931	-0.978
V_lighter	0.998	0.889	0.837	0.992	0.408	0.770	0.948	-0.867	0.870	**0.999**	0.984	-0.912	0.938	-0.902	0.907
Blur	-0.241	0.916	0.969	0.800	-0.946	0.900	-0.936	-0.983	-0.927	-0.945	-0.860	**0.986**	-0.833	-0.977	0.936
Distort	0.613	0.957	**0.974**	0.523	0.783	0.868	0.776	0.925	0.756	0.758	0.875	0.554	0.667	-0.953	0.721
Noise	0.922	0.934	0.960	0.913	-0.863	0.832	-0.918	0.694	-0.910	-0.923	-0.827	**0.984**	-0.921	0.930	-0.833
Average	0.713	**0.931**	0.895	0.756	0.470	0.815	0.376	0.015	0.429	0.306	0.438	0.601	0.351	-0.666	-0.045

Table 4 PCC values for different methods and input image disturbances, for Nvidia Net (PL for PyOD).

	AE	KNN	LOF	DSVDD	IForest	OCSVM	COPOD	HBOS	PCA	ECOD	CBLOF	DAGMM	LODA	SOD	COF
B_darker	0.979	0.972	0.493	0.962	0.986	0.949	0.998	0.996	0.995	0.995	**0.999**	-0.954	0.994	-0.537	-0.974
B_lighter	0.997	0.722	0.283	-0.662	0.912	0.851	0.939	0.649	0.995	0.980	**0.999**	0.992	0.993	-0.213	-0.771
G_darker	0.994	0.893	0.413	0.737	0.975	0.922	**1.000**	0.985	0.990	0.994	0.990	0.593	0.993	-0.928	-0.885
G_lighter	0.975	0.947	0.771	**0.995**	0.985	0.986	0.979	0.975	-0.884	-0.235	0.983	-0.973	-0.905	0.566	-0.943
H_darker	0.953	0.948	0.686	0.998	0.982	0.914	0.998	0.978	0.997	0.996	**0.999**	0.996	0.992	0.864	-0.911
H_lighter	0.935	0.961	0.873	**0.997**	0.975	0.956	0.980	0.964	0.988	0.965	0.977	0.941	0.973	0.906	-0.972
IMGC_fog	0.996	0.932	0.774	0.955	**0.998**	0.969	0.021	0.917	-0.451	-0.347	-0.151	0.913	-0.424	0.834	-0.541
IMGC_frost	0.870	0.917	0.712	0.947	0.961	**0.981**	-0.323	0.872	-0.965	-0.896	0.934	-0.738	-0.978	0.740	-0.942
IMGC_jpeg	0.338	0.930	**0.971**	0.949	0.869	0.927	-0.122	0.851	-0.851	-0.910	0.896	-0.832	-0.983	0.970	-0.106
IMGC_motion_blur	-0.430	0.960	0.974	0.978	**0.983**	0.978	-0.424	0.973	-0.954	-0.755	0.241	0.890	-0.954	0.906	-0.436
IMGC_snow	0.922	0.886	0.765	**0.978**	0.624	0.974	-0.916	-0.875	-0.947	-0.928	-0.765	-0.624	-0.971	-0.428	-0.960
IMGC_zoom_blur	-0.595	0.971	0.984	0.965	0.974	0.970	0.924	**0.993**	-0.617	-0.974	0.975	0.895	0.150	0.901	-0.601
R_darker	0.949	0.971	0.853	0.961	0.996	0.990	0.958	**0.997**	0.682	0.656	0.978	0.591	0.437	-0.136	-0.971
R_lighter	0.961	0.957	0.417	0.950	0.992	0.966	0.990	**0.993**	0.979	0.983	0.967	-0.799	0.970	-0.998	-0.931
S_darker	0.966	0.963	0.870	0.991	0.997	0.972	0.996	**1.000**	0.965	0.782	0.995	0.935	0.917	0.991	-0.992
S_lighter	0.923	0.989	0.966	0.972	0.998	0.980	0.997	**1.000**	0.967	0.988	0.987	0.980	0.982	0.993	-0.980
V_darker	0.953	0.755	0.815	0.349	0.927	0.859	0.983	0.989	**0.990**	0.961	-0.586	0.407	0.938	-0.968	-0.990
V_lighter	**0.998**	0.970	0.839	0.988	0.998	0.974	0.933	0.990	0.230	0.084	0.989	0.952	0.330	0.978	-0.988
Blur	-0.241	0.972	0.879	0.966	**0.989**	0.986	0.948	0.979	0.887	0.746	0.936	0.928	0.894	-0.800	-0.976
Distort	0.613	-0.241	0.634	0.472	-0.895	-0.744	-0.962	-0.946	-0.998	0.089	-0.968	**0.758**	-0.978	-0.918	-0.689
Noise	0.922	0.937	0.880	0.981	0.995	0.972	-0.328	0.987	-0.807	-0.791	-0.166	**0.998**	-0.839	0.990	-0.748
Average	0.713	0.872	0.755	0.830	0.868	**0.873**	0.503	0.775	0.152	0.209	0.581	0.374	0.168	0.224	-0.824

Table 5 PCC values for different methods and input image disturbances, for ResNet-152 (FL/PL for PyOD).

	AE	KNN	LOF	DSVDD	IForest	OCSVM	COPOD	HBOS	PCA	ECOD	CBLOF	DAGMM	LODA	SOD	COF
B_darker	0.950	0.941	-0.014	0.582	0.864	0.986	0.827	0.875	0.932	0.814	0.989	-0.403	**0.990**	0.510	-0.867
B_lighter	0.976	0.964	0.982	0.669	0.978	-0.963	0.973	0.982	**0.985**	0.948	-0.659	-0.800	-0.877	0.862	-0.757
G_darker	0.969	0.846	0.185	0.436	0.819	0.994	0.772	0.821	0.805	0.735	**0.995**	0.723	0.990	0.137	-0.852
G_lighter	0.851	0.969	0.809	**0.988**	0.965	0.846	0.962	0.971	0.968	0.953	0.971	-0.739	-0.971	-0.254	-0.887
H_darker	0.793	0.905	0.913	0.979	0.950	-0.830	**0.986**	0.971	0.971	0.961	-0.713	0.437	0.936	0.983	0.775
H_lighter	0.861	0.976	0.981	0.970	0.983	0.209	0.988	0.991	**0.992**	0.976	-0.722	-0.889	-0.333	0.964	0.741
IMGC_fog	0.980	0.981	0.956	0.913	0.981	**0.997**	0.980	0.979	0.978	0.981	0.996	0.957	0.997	0.972	-0.990
IMGC_frost	0.990	0.993	0.985	0.959	0.996	0.638	0.996	0.995	0.991	**0.997**	0.876	-0.469	-0.976	0.977	-0.909
IMGC_jpeg	0.980	0.966	0.985	0.943	0.990	0.843	0.984	0.990	0.988	0.986	-0.994	-0.958	0.021	**0.992**	0.968
IMGC_motion_blur	0.977	0.986	0.982	0.990	0.996	0.872	**0.998**	0.998	0.998	0.996	0.503	-0.811	-0.669	0.995	0.524
IMGC_snow	**0.975**	0.932	0.948	0.851	0.954	0.508	0.951	0.940	0.914	0.962	0.406	0.269	0.191	0.766	-0.659
IMGC_zoom_blur	**0.994**	0.993	0.968	0.934	0.985	0.900	0.960	0.984	0.978	0.990	0.638	-0.042	0.759	0.965	-0.426
R_darker	0.970	0.921	0.860	0.294	0.934	0.935	0.908	0.938	0.921	0.887	0.981	0.963	**0.982**	0.394	-0.966
R_lighter	0.903	0.912	-0.107	0.446	0.954	**0.985**	0.933	0.937	0.839	0.925	0.983	-0.714	0.936	0.512	-0.961
S_darker	0.855	0.879	0.883	**0.965**	0.932	-0.717	0.960	0.952	0.958	0.932	-0.890	-0.883	-0.917	0.898	0.916
S_lighter	**0.996**	0.938	0.648	0.583	0.842	0.986	0.819	0.876	0.919	0.820	0.990	0.994	0.982	0.786	-0.620
V_darker	0.850	0.858	-0.176	0.395	0.892	0.997	0.565	0.855	0.813	0.492	0.997	0.597	**0.999**	-0.500	-0.557
V_lighter	0.997	0.900	0.590	**1.000**	0.827	0.897	0.816	0.857	0.860	0.838	0.832	-0.014	0.832	0.818	-0.186
Blur	0.954	0.997	0.990	0.833	0.990	0.990	0.985	0.997	**0.998**	0.979	0.976	0.935	0.920	0.975	-0.904
Distort	**0.975**	0.900	0.956	-0.023	0.957	-0.968	0.929	0.936	0.930	0.870	-0.998	0.604	-0.924	-0.705	0.808
Noise	0.928	0.977	0.924	0.822	0.957	0.521	0.958	0.972	**0.987**	0.939	0.971	0.135	0.975	0.919	-0.741
Average	0.939	0.940	0.726	0.739	0.940	0.506	0.917	**0.944**	0.939	0.904	0.387	-0.005	0.278	0.617	-0.312

Author Contributions System implementation and evaluation (S. Luan); idea conception and paper writing (Z. Gu and S. Wan).

Funding Open access funding provided by Umea University. This work was partially supported by National Natural Science Foundation of China under Grant No. 62172438, Key Project of Shenzhen City Special Fund for Fundamental Research under Grant # 202208183000751, and the Kempe Foundation, Sweden.

Data Availability N/A.

Declarations

Ethics Approval This research involves no human participants and/or animals.

Competing Interests The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. Song, Y., Li, Y., Jia, L., & Qiu, M. (2019). Retraining strategy-based domain adaption network for intelligent fault diagnosis. *IEEE Transactions on Industrial Informatics, 16*(9), 6163–6171.
2. De Lange, M., et al. (2021). A continual learning survey: Defying forgetting in classification tasks. *IEEE transactions on pattern analysis and machine intelligence, 44*(7), 3366–3385.
3. Zhai, R., Schroedl, S., Galstyan, A., Kumar, A., Ver Steeg, G., & Natarajan, P. (2022). Online Continual Learning for Progressive Distribution Shift (OCL-PDS): A Practitioner's Perspective.
4. Tampuu, A., Matiisen, T., Semikin, M., Fishman, D., & Muhammad, N. (2020). A survey of end-to-end driving: Architectures and training methods. *IEEE Transactions on Neural Networks and Learning Systems, 33*(4), 1364–1384.
5. Bojarski, M., et al. (2016). End to end learning for self-driving cars. *arXiv preprint* arXiv:1604.07316
6. Wen, L. H., & Jo, K. H. (2022). Deep learning-based perception systems for autonomous driving: A comprehensive survey. *Neurocomputing, 489*, 255–270.
7. Yang, J., et al. (2022). OpenOOD: Benchmarking generalized out-of-distribution detection. *arXiv preprint* arXiv:2210.07242
8. Luan, S., Gu, Z., Freidovich, L. B., Jiang, L., & Zhao, Q. (2021). Out-of-distribution detection for deep neural networks with isolation forest and local outlier factor. *IEEE Access, 9*, 132980–132989.
9. Luan, S., Gu, Z., Saremi, A., Freidovich, L., Jiang, L., & Wan, S. (2023). Timing performance benchmarking of out-of-distribution detection algorithms. *Journal of Signal Processing Systems.*
10. Stocco, A., Weiss, M., Calzana, M., & Tonella, P. (2020). Misbehaviour prediction for autonomous driving systems. In *Proceedings of the ACM/IEEE 42nd international conference on software engineering* (pp. 359-371).
11. Cai, F., & Koutsoukos, X. (2020). Real-time out-of-distribution detection in learning-enabled cyber-physical systems. In *2020 ACM/IEEE 11th International Conference on Cyber-Physical Systems (ICCPS)* (pp. 174-183). IEEE.
12. Ruff, L., et al. (2018). Deep one-class classification. In *International conference on machine learning* (pp. 4393-4402). PMLR.
13. Laxhammar, R., & Falkman, G. (2015). Inductive conformal anomaly detection for sequential detection of anomalous sub-trajectories. *Annals of Mathematics and Artificial Intelligence, 74*, 67–94.
14. Henne, M., Schwaiger, A., Roscher, K., & Weiss, G. (2020). Benchmarking Uncertainty Estimation Methods for Deep Learning With Safety-Related Metrics. In *SafeAI@ AAAI* (pp. 83-90).
15. Lohdefink, J., Fehrling, J., Klingner, M., Huger, F., Schlicht, P., Schmidt, N. M., & Fingscheidt, T. (2020). Self-supervised domain mismatch estimation for autonomous perception. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops* (pp. 334-335).
16. Rottmann, M., Colling, P., Hack, T. P., Chan, R., Hüger, F., Schlicht, P., & Gottschalk, H. (2020, July). Prediction error meta classification in semantic segmentation: Detection via aggregated dispersion measures of softmax probabilities. In *2020 International Joint Conference on Neural Networks (IJCNN)* (pp. 1-9). IEEE.
17. Klingner, M., Bar, A., Mross, M., & Fingscheidt, T. (2021). Improving online performance prediction for semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 1–11).
18. Klingner, M., & Fingscheidt, T. (2021). Online performance prediction of perception DNNs by multi-task learning with depth estimation. *IEEE Transactions on Intelligent Transportation Systems, 22*(7), 4670–4683.
19. Bär, A., Klingner, M., Löhdefink, J., Hüger, F., Schlicht, P., & Fingscheidt, T. (2022). Performance Prediction for Semantic Segmentation by a Self-Supervised Image Reconstruction Decoder. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 4399-4408).
20. Lee Rodgers, J., & Nicewander, W. A. (1988). Thirteen ways to look at the correlation coefficient. *The American Statistician*, 42(1), 59–66.
21. Zhao, Y., Nasrullah, Z., & Li, Z. (2019). Pyod: A python toolbox for scalable outlier detection. *arXiv preprint* arXiv:1901.01588
22. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770–778).
23. Shyu, M. L., Chen, S. C., Sarinnapakorn, K., & Chang, L. (2003). A novel Anomaly Detection scheme based on principal component classifier. Miami Univ Coral Gables Fl Dept of Electrical and Computer Engineering.
24. Schölkopf, B., Williamson, R., Smola, A., Shawe-Taylor, J., & Platt, J. (2000). Support vector method for novelty detection. *Advances in neural information processing systems, 12*(3), 582–588.
25. Breunig, M. M., Kriegel, H. P., Ng, R. T., & Sander, J. (2000). LOF: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data* (pp. 93–104).
26. He, Z., Xu, X., & Deng, S. (2003). Discovering cluster-based local outliers. *Pattern recognition letters, 24*(9–10), 1641–1650.
27. Tang, J., Chen, Z., Fu, A. W. C., & Cheung, D. W. (2002). Enhancing effectiveness of outlier detections for low density patterns. In *Advances in Knowledge Discovery and Data Mining: 6th Pacific-Asia Conference, PAKDD 2002 Taipei, Taiwan, May 6–8, 2002 Proceedings 6* (pp. 535–548). Springer.

28. Goldstein, M., & Dengel, A. (2012). Histogram-based outlier score (hbos): A fast unsupervised anomaly detection algorithm. *KI-2012: Poster and Demo Track,* 1, 59–63.
29. Ramaswamy, S., Rastogi, R., & Shim, K. (2000). Efficient algorithms for mining outliers from large data sets. In *Proceedings of the 2000 ACM SIGMOD international conference on management of data* (pp. 427–438).
30. Kriegel, H. P., Kröger, P., Schubert, E., & Zimek, A. (2009). Outlier detection in axis-parallel subspaces of high dimensional data. In *Advances in Knowledge Discovery and Data Mining: 13th Pacific-Asia Conference (PAKDD)* (pp. 831–838). Springer.
31. Li, Z., Zhao, Y., Botta, N., Ionescu, C., & Hu, X. (2020). COPOD: copula-based outlier detection. In *2020 IEEE international conference on data mining (ICDM)* (pp. 1118-1123). IEEE.
32. Li, Z., Zhao, Y., Hu, X., Botta, N., Ionescu, C., & Chen, G. (2022). ECOD: Unsupervised outlier detection using empirical cumulative distribution functions. *IEEE Transactions on Knowledge and Data Engineering*.
33. Zong, B., Song, Q., Min, M. R., Cheng, W., Lumezanu, C., Cho, D., & Chen, H. (2018). Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In *International conference on learning representations.*
34. Pevný, T. (2016). Loda: Lightweight on-line detector of anomalies. *Machine Learning, 102*, 275–304.
35. Liu, F. T., Ting, K. M., & Zhou, Z. H. (2008). Isolation forest. In *2008 eighth ieee international conference on data mining* (pp. 413–422). IEEE.
36. Geyer, J., et al. (2020). A2d2: Audi autonomous driving dataset. *arXiv preprint* arXiv:2004.06320
37. Shen, Y., Zheng, L., Shu, M., Li, W., Goldstein, T., & Lin, M. (2021). Gradient-free adversarial training against image corruption for learning-based steering. *Advances in Neural Information Processing Systems, 34*, 26250–26263.
38. Hendrycks, D., & Dietterich, T. (2019). Benchmarking neural network robustness to common corruptions and perturbations, presented at the International Conference on Learning Representations (ICLR).
39. Huang, H., Chaturvedi, V., Quan, G., Fan, J., & Qiu, M. (2014). Throughput maximization for periodic real-time systems under the maximal temperature constraint. *ACM Transactions on Embedded Computing Systems (TECS), 13*(2s), 1–22.