Intelligent Systems with Applications 17 (2023) 200172

Contents lists available at ScienceDirect

Intelligent Systems with Applications

journal homepage: www.journals.elsevier.com/intelligent-systems-with-applications

Assessing operational accuracy of CNN-based image classifiers using an oracle surrogate

Antonio Guerriero [a,*], Michael R. Lyu [b], Roberto Pietrantuono [a], Stefano Russo [a]

[a] Università degli Studi di Napoli Federico II, Italy
[b] The Chinese University of Hong Kong, Hong Kong

ARTICLE INFO

Keywords:
Image classification
Machine Learning
Convolutional Neural Networks
Accuracy assessment
Oracle Problem

ABSTRACT

Context: Assessing the accuracy in operation of a Machine Learning (ML) system for image classification on arbitrary (unlabeled) inputs is hard. This is due to the *oracle problem*, which impacts the ability of automatically judging the output of the classification, thus hindering the accuracy of the assessment when unlabeled previously unseen inputs are submitted to the system.
Objective: We propose the *Image Classification Oracle Surrogate* (ICOS), a technique to automatically evaluate the accuracy in operation of image classifiers based on Convolutional Neural Networks (CNNs).
Method: To establish whether the classification of an arbitrary image is correct or not, ICOS leverages three knowledge sources: operational input data, training data, and the ML algorithm. Knowledge is expressed through *likely invariants* - properties which should not be violated by correct classifications. ICOS infers and filters invariants to improve the correct detection of misclassifications, reducing the number of false positives. We evaluate ICOS experimentally on twelve CNNs – using the popular MNIST, CIFAR10, CIFAR100, and ImageNet datasets. We compare it to two alternative strategies, namely cross-referencing and self-checking.
Results: Experimental results show that ICOS exhibits performance comparable to the other strategies in terms of accuracy, showing higher stability over a variety of CNNs and datasets with different complexity and size.
Conclusions: ICOS likely invariants are shown to be effective in automatically detecting misclassifications by CNNs used in image classification tasks when the expected output is unknown; ICOS ultimately yields faithful assessments of their accuracy in operation. Knowledge about input data can also be manually incorporated into ICOS, to increase robustness against unexpected phenomena in operation, like label shift.

1. Introduction

Machine Learning (ML) systems are today integral part of many applications due to their ability of reaching the same level or of even outperforming human beings (Kühl et al., 2020, He et al., 2015, Silver et al., 2017) for many tasks, like in the image classification (IC) domain. An ML system "is a software system including one or more components that learn how to perform a task from a given data set" (Riccio et al., 2020). The learning components are based on ML models. The main performance indicator of such models is the *accuracy*, namely the number of correctly classified images out of the total. The accuracy of an ML model relies on different factors, like the data chosen for training, the training process itself, and the verification process.

To assess the model accuracy on the field (*operational accuracy*), an arbitrarily large set of operational data could be collected (*operational dataset*) and submitted to the model. However, the correct labels for operational data are generally unknown, and the most reliable approach to define the ground truth is still manual labeling – this is because of the well known *oracle problem* (Murphy et al., 2007).

Evaluating ML-based IC systems with a large number of arbitrary input images without an automatic oracle – thus, by manually checking that each image is correctly classified - is clearly expensive. There are two main strategies in the literature to address this problem: *a)* sampling a conveniently small subset from the operational dataset, according to a certain belief (e.g., selecting those samples more representative of the whole operational dataset (Li et al., 2019) or by selecting the likeliest failing samples (Guerriero et al., 2021)) and then manually la-

* Corresponding author.
E-mail addresses: antonio.guerriero@unina.it (A. Guerriero), lyu@cse.cuhk.edu.hk (M.R. Lyu), roberto.pietrantuono@unina.it (R. Pietrantuono), stefano.russo@unina.it (S. Russo).

https://doi.org/10.1016/j.iswa.2022.200172
Received 28 July 2022; Received in revised form 25 November 2022; Accepted 24 December 2022
Available online 4 January 2023

beling only such inputs to get an estimate of the expected accuracy on the whole operational dataset; *b)* exploiting ML algorithms and statistical techniques to automatically detect failures as the input image is classified.

Our work focuses on the latter strategy, with the aim to avoid the cost of manual labeling the selected samples. A common approach for such a strategy is to build oracles through cross-referencing, like in multiple implementation testing (Srisakaokul et al., 2018), e.g., detecting classification failures by majority voting. The various techniques of this type differ in the way by which multiple models are derived: one can, for instance, train different models on the same set (Srisakaokul et al., 2018, Pei et al., 2017), or consider intermediate models during training (Wang et al., 2020).

A sort of cross-referencing oracle is also used by *SelfChecker*, the self-checking system for Deep Neural Networks (DNNs) proposed by Xiao et al. (2021), which "monitors DNN outputs and triggers an alarm if the internal layer features of the model are inconsistent with the final prediction" The technique is shown to be effective in detecting failures with an acceptable false positives rate - probably the trickiest issue in automatic oracles.

These techniques rely on the knowledge encoded in the training set and/or in the internal structure of the model (single neurons or layers output). However, in operation, well-known phenomena like *concept drift* (Tsymbal, 2004), *distribution shift* and *label shift* (Garg et al., 2020) can strongly impact the accuracy of the model, since the model is called to operate on inputs that deviate from those observed at training time. In these cases, such knowledge becomes less effective as a source to build an automatic oracle, as we show in the experiments.

To target this problem, emerging ML systems lifecycles like MLOps (Alla & Adari, 2021) foresee specialized teams, involving both software and operations engineers. They have to ensure the correct behavior taking into account the characteristics of the actual execution environment and of the operational domain knowledge, collected during active monitoring and exploited to contrast the above-mentioned phenomena.

We propose ICOS (*Image Classification Oracle Surrogate*), a technique to address the *oracle problem* when assessing the operational accuracy provided by ML-based IC systems. It consists of an *oracle surrogate* that judges if the IC program under test correctly classifies an arbitrary input image whose label is unknown. The ICOS automatic oracle aims to robustness to operational changes by: *i)* considering multiple sources of information, including, besides the training set and the ML algorithm, the operational domain knowledge; *ii)* filtering the knowledge in the training set more robust to changes in order to balance the occurrence of false positives and maximize the number of true positives.

ICOS derives a set of *likely invariants* representing properties that all correct outputs should preserve, leveraging the following sources of knowledge:

- *Input data*: the invariants from the operational input (called *input-data-dependent invariants*) encode the operational domain knowledge as rules defined by domain experts on the input and provided to the ML model; the resulting invariants are then automatically checked for violation.
- *Training data*: *training-data-dependent invariants* are automatically extracted from training data to give a characterization of the ML model's expected behavior.
- *ML algorithm*: *algorithm-dependent invariants* capture the information about how the output is computed from the ML algorithm.

When any invariant is violated, ICOS labels the test output as *fail*, otherwise as *pass*. The implementation of ICOS is publicly available on GitHub.[1] A recent work from Google stresses the need and importance of incorporating domain knowledge as a set of rules to improve training (Choudhary, 2022). In line with this work, with input-dependent-invariants, we integrate into ICOS the domain knowledge to assess CNNs operational accuracy. The objective is to create an automatic oracle more effective than the state-of-the-art ones in estimating the accuracy of the CNN during the operation.

We evaluate ICOS on twelve Convolutional Neural Networks (CNNs), the most popular and performing ML-based IC solutions (Sharma et al., 2018). We compare ICOS to the Cross-Referencing Oracle (CRO) implementation provided by Srisakaokul et al. (2018) and to SelfChecker (Xiao et al., 2021). The experimental datasets are MNIST (LeCun & Cortes, 2010), CIFAR10, CIFAR100 (Krizhevsky, 2009), and ImageNet (Deng et al., 2009), widely used in IC. We study the accuracy estimation ability considering the contribution of different types of invariants, the sensitivity to invariant selection criteria, and the robustness of the oracle surrogate in presence of label shift.

Results show that ICOS is able to faithfully estimate the accuracy provided by the CNNs in the operational environment, outperforming CRO and SelfChecker. All the three types of invariants contribute to misclassification detection, but a fine selection of the invariant influences the obtained results. In the results, we see that by selecting more invariants the number of correctly detected failures increases, but paying in terms of false positives. Finally, performance is shown to be more robust than the baselines with respect to unexpected phenomena like label shift, with an error reduction in presence of shift ranging of two orders of magnitude in the best case.

2. Related work

We analyze related research on the operational accuracy assessment of ML systems, with specific reference to CNNs for Image Classification.

A significant research effort has been devoted in recent years to quality evaluation of ML systems (Riccio et al., 2020), yet few works concern the assessment of the accuracy provided in the operational environment. In fact, researchers primarily focused on testing of ML systems, with the main aim of exposing *mispredictions*, namely of spotting as many failing behaviors as possible (Pei et al., 2019, Ma, Juefei-Xu, et al., 2018, Zhang et al., 2018, Ma, Zhang, Xue, et al., 2018, Odena & Goodfellow, 2019).

The output of this type of failure-finding testing (and then debugging) process is an improved model, with higher accuracy. This resembles what is called *debug testing* in the traditional testing literature (Frankl et al., 1998). Clearly, as in the traditional *debug testing*, the so-obtained testing results are not necessarily related to the accuracy actually experienced in operation, and cannot be used for operational accuracy assessment, as testing data may be not representative of the actual operational context. This happens both when test data are generated artificially (like in adversarial examples generation) or when they are natural but differ significantly from input observed in the field. The resulting number of exposed mispredictions and/or the coverage achieved give only an "indirect" indication of the expected accuracy in operation, and ultimately of the confidence that can be placed in the system, but no quantitative estimation is given.

For estimating the accuracy in operation, two main strategies are:

- sampling a subset of the operational input dataset to be manually labeled, and then use it to estimate the accuracy. The idea is to select an as much small as possible subset of inputs, from which an accurate and stable (i.e., small variance) estimate is obtained (Li et al., 2019, Guerriero et al., 2021, Zhao et al., 2022). This mirrors *operational* testing for conventional (not ML-based) systems (Musa, 1996, Pietrantuono & Russo, 2016).
- exploiting ML algorithms and statistical techniques to automatically detect failures in operation. The idea is to evaluate the output automatically, namely to implement an oracle, so as to avoid the need of manually labeling the inputs (Guerriero, 2020).

[1] https://github.com/ICOS-OAA/ICOS.git.

As the cost of manual labeling can be high and is not scalable, this work focuses on the second solution, which also allows an online evaluation of the operational accuracy. The rest of the section focuses on the literature on automated oracles.

Automated oracles The oracle problem in ML testing is one of the main challenges tackled by researchers Zhang et al. (2022). Often the proposed solutions are tailored for, or at least evaluated on, image classification.

A common strategy to build an automatic oracle is to use *cross referencing*, such as multiple-implementation testing (MIT) (Srisakaokul et al., 2018). MIT is proposed by Srisakaokul et al. to test supervised learning software. A test input's proxy oracle is derived from the majority-voted output of multiple implementations of the same algorithm. The cost of multiple implementations is clearly high. On the other hand, the solution is able to obtain a feedback about the output of any arbitrary input submitted to the system under test. The technique does not require any prior knowledge about the images' labels.

Pei et al. adopt multiple Deep Learning (DL) systems in their DeepXplore framework for white-box testing (Pei et al., 2019). They define a neuron coverage metric to measure the parts of the SUT exercised by test inputs. The DL systems are used as cross-referencing oracles to avoid manual checking.

Wang et al. (2020) propose DISSECTOR, a fault tolerance approach to distinguish input potentially causing a failure of the ML system. The input validation is performed by training sub-models on top of the pre-trained model under test, hence using sub-models for cross-referencing.

The common characteristic among the three presented techniques is the source of knowledge used to set up the oracle as cross-referencing. In all the cases, the output of the ML system is evaluated based on the knowledge encoded into the *training set*. The multiple implementations, different from each other (different ML models, or the same ML model but different architecture, or sub-models trained from the same main ML model), aim to extract as much knowledge as possible from the training set to perform a majority voting based on that knowledge. These techniques are strictly affected by biases in the training set. When the training data are not representative of the operational environment, the performance of that oracles degrades significantly.

Corbière et al. (2019) propose a criterion for failure prediction of CNNs based on True Class Probability (TCP). The criterion is learned by a confidence neural network (ConfidNet) built upon a classification model. TCP is shown effective in performing failure prediction on classification and segmentation problems.

Currently, automatic oracles are of great interest also in misbehavior prediction of DNNs in autonomous driving (Jahangirova et al., 2021). Stocco et al. propose SelfOracles to detect unsupported driving scenarios based on DNN run time behavior (Stocco et al., 2020). Based on the images in the training set, autoencoders are used to compute for each operational image a reconstruction error. The higher the error, the higher the probability of failure on the considered sample.

Xiao et al. (2021) recently proposed SelfChecker (SC) for both failure detection of CNNs and autonomous driving systems. SC detects failures in deployment when the output of the internal layers of the model under test is inconsistent with the final prediction. In this case, the internal layers' output is used for cross-referencing. Besides failure detection, SC also suggests an alternative prediction. SC significantly outperforms the state-of-the-art techniques (DISSECTOR (Wang et al., 2020), ConfidNet (Corbière et al., 2019), and SelfOracle (Stocco et al., 2020)).

The difference between the first three (MIT, DeepXplore, and DISSECTOR) and the last three (ConfidNet, SelfOracle, and SC) techniques is how the knowledge is extracted from the training set. In particular, the first three approaches try "different" models learning from the same source, exploiting the ensemble effect. The last three techniques compute metrics to exploit the knowledge encoded in each training image. This strategy is particularly effective for SC, which outperforms state-of-the-art techniques in failure prediction.

The discussed techniques do not account for possible deviations of the operational context from the pre-deployment one. Hence, they are expected to perform insufficiently in presence of phenomena like label shift (Garg et al., 2020).

Two other techniques only partially addressing the oracle problem are mutation testing (Ma, Zhang, Sun, et al., 2018, Li et al., 2022) and metamorphic testing (Xie et al., 2019). The former is proposed by Ma et al. to evaluate test data quality (Ma, Zhang, Sun, et al., 2018). They define a set of source-level operators to inject faults into the sources of a DL model, like training data and programs, and model-level operators to inject faults directly into models. Data-related mutations (Ma, Zhang, Sun, et al., 2018, Li et al., 2022) aim to act directly on input data with a known label, to generate new samples, or alter the correct label. As the mutation approach requires knowledge of the label, it does not allow to submit test cases whose expected output is unknown.

Metamorphic testing leverages relations (*metamorphic relations*) between changes of input and output over various executions; when a relation is violated, a failure is detected. It finds application in many domains, like IC (Xie et al., 2019, Dwarakanath et al., 2018), autonomous driving (Tian et al., 2018) and sentiment analysis (Jiang et al., 2022). In IC, it is used to generate new images preserving semantics (namely, the label) (Xie et al., 2019, Dwarakanath et al., 2018), or images with a label certainly different from the original one (Dwarakanath et al., 2018), to find defects, like implementation bugs. As this way of applying metamorphic testing assumes that tests are generated from images whose labels are already known, it is not applicable to test arbitrary images with unknown labels.

3. ML systems life cycle

The accuracy assessment is essential in the life cycle of ML systems subject in operation to phenomena, which may negatively affect the faithfulness of the predictions. We describe the ML systems life cycle, according to the main proposals in literature (Ashmore et al., 2021, Alla & Adari, 2021), and show how the accuracy assessment task can be integrated into it.

Ashmore et al. (2021) present a detailed ML systems life cycle, represented in Fig. 1. Four main stages can be identified:

- *Data Management*: data collected in operation are processed and selected to generate new training and verification datasets;
- *Model Learning*: an ML model is selected and trained;
- *Model Verification*: the trained model is evaluated on the verification set; if the generalization error exceeds a threshold, the process returns to the Data Management stage;
- *Model Deployment*: the model satisfying the requirements is integrated into the operational environment; its operation is monitored, and the model is updated through offline maintenance or online learning.

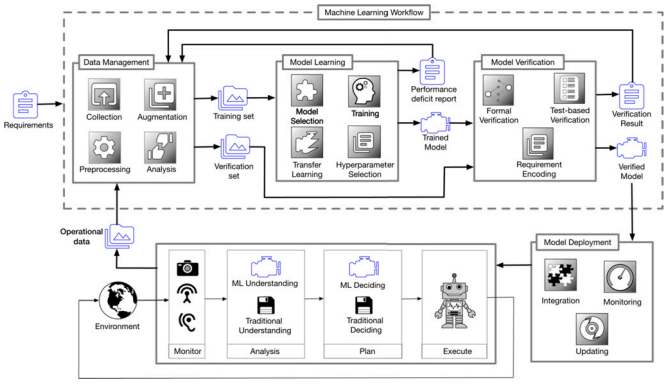


Fig. 1. ML systems life cycle according to ref. (Ashmore et al., 2021).

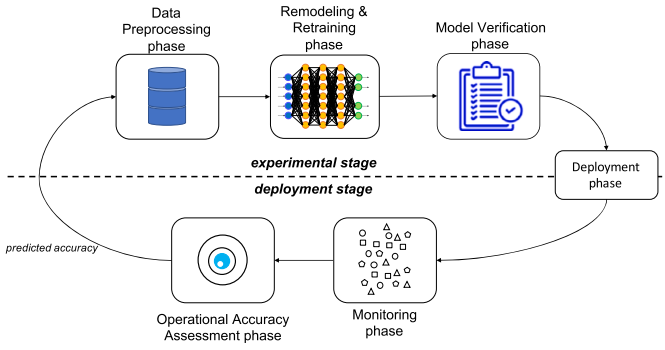


Fig. 2. ML systems life cycle with accuracy assessment.

This spiral life cycle allows the collection of operational data to improve the model over iterations. In the emerging MLOps perspective (Alla & Adari, 2021), the first three stages can be seen as the *experimental stage*; Model Deployment corresponds to the *deployment stage*.

Fig. 2 depicts a generic ML system life cycle, concealing Ashmore and MLOps perspectives. In particular, the *Data Preprocessing*, *Remodeling & Retraining*, and *Model verification* phases correspond to the first three stages proposed by Ashmore, and they are considered in the experimental stage, concerning all the actions performed pre-release, outside the operational environment. The *Deployment* phase includes the activities to deploy the ML system in the operational environment; it represents the transition phase between the experimental and deployment stages. In the deployment stage, the *Monitoring* phase concerns the collection of operational data, peculiar environment characteristics, and the output of the ML system useful to evaluate the ML system's accuracy in operation and to take correcting/improving actions in the next cycle. We refer to this life cycle in the rest of the paper.

MLOps principles (Google, 2022) strongly focus on the concepts of Continuous Integration and Continuous Delivery. They aim to develop a system able to evolve according to the operational environment, stressing the monitoring process, collecting statistics on the model performance (e.g. operational accuracy) based on live data, and envisaging the online auto-improving of model accuracy in operation (e.g. via auto-training). The main threat to achieving these objectives is the *oracle problem*. For ML systems, including CNNs for IC, "there is no reliable test oracle to indicate what the correct output should be for arbitrary input" (Murphy et al., 2008). The automation of an operational accuracy assessment process is limited by the unavailability of the correct label for operational inputs.

The online assessment performed via automatic oracle can partially solve this problem, and it can be integrated into the life cycle of Fig. 2. Specifically, the *Online Assessment* phase can be placed in the *deployment stage*: the oracle computes the predicted accuracy on the data coming from monitoring the CNN in operation. The predicted accuracy is then forwarded into the experimental stage. Based on this estimate, correcting/improving actions can be performed in the Data Preprocessing and Remodeling & Retraining phases.

4. Image Classification Oracle Surrogate

We now present ICOS and describe the three types of likely invariants for evaluating CNN output when the expected output is unknown.

4.1. Overview

According to Murphy et al. (2007), an *oracle* is an entity able to "indicate what the correct output should be for arbitrary input". We define an *oracle surrogate* as an entity able to evaluate as correct or wrong each prediction of the CNN, based on some knowledge. The *Image Classification Oracle Surrogate* implements an oracle surrogate in the IC domain inferring a set of invariants from input data, training data,

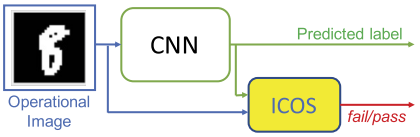


Fig. 3. ICOS workflow.

and the adopted ML algorithm, representing assertions that a correct CNN response should never violate. As Fig. 3 shows, both the image and the CNN output are submitted to ICOS, which checks against the invariants: if at least one invariant is violated, the ICOS output is *fail*, otherwise *pass*. When a failure is detected by one type of invariant, the remaining ones are neglected.

Invariants are expressed, as in constraint logic programming, as *clauses* in the form:

$$H \text{ :- } C_1 \wedge \cdots \wedge C_n, B_1 \wedge \cdots \wedge B_m \tag{1}$$

where H and B_i are atomic formulas and C_i are constraints, and it is read as a rule: H is true if C_1 and C_2 and C_n are satisfied, and B_1 and B_2 and B_m are true. This representation suits the three types of invariants considered. The following sections detail the invariants.

As an example, consider the following clause:

$$\begin{aligned}(outcome = Fail) \text{ :- } &(pixel_1 > 25) \wedge \cdots \wedge (pixel_n < 250),\\ &(predicted\_label \neq 3) \wedge (predicted\_label \neq 6)\end{aligned} \tag{2}$$

stating that the outcome of ICOS is *Fail* if the values of the specified pixels satisfy the respective constraints, and the predicted label is different from 3 and 6. In this example, $outcome = Fail$ is the atomic formula H, $predicted\_label! = 3$, and $predicted\_label! = 6$ are the atomic formulas B, and $pixel_1 > 25$ and $pixel_n < 250$ are the constraints C.

4.2. Input-data-dependent invariants

The invariants considered in the first stage of ICOS aim to partition the *operational input*, based on specific features of the operational environment (e.g. the way images are generated) that are unavailable for training images, e.g., deriving from the operational domain knowledge.

Knowledge about the context in which the classifier operates can help reduce the error, as it can, for instance, exclude labels that could never occur in that context. Reasonably, this information can be used to define an invariant such as "if the operational input belongs to the partition X, then the label predicted by the CNN cannot be y". This invariant can be considered deterministic for a system, where X is the partition of all images generated from the camera pointed in the garden, and y is the airplane label.

Another example is the classification of handwritten digits, a very common task in image classification research (Lecun et al., 1998). Consider a system with two input forms: in the first form the user must enter only digits without straight lines; in the second form, the user has to enter only digits with straight lines. The operational input may be clearly divided into two partitions: digits without straight lines ($P1$) and digits with straight lines ($P2$). The corresponding invariants are:

$$fail \text{ :- } input\_image \in P_1,\ output \notin \{0,3,6,8,9\}$$

$$fail \text{ :- } input\_image \in P_2,\ output \notin \{1,2,4,5,7\}$$

ICOS aims to incorporate such additional invariants when available. For instance, in a successive release of the system, the second form is replaced by two forms that require the user to insert respectively: input images without curves ($P_{2,1}$), and digits with both curves and straight lines ($P_{2,2}$). Accordingly, the second invariant is updated as follows:

$$fail \text{ :- } input\_image \in P_{2,1},\ output \notin \{1,4,7\}$$

$$fail \text{ :- } input\_image \in P_{2,2}, output \notin \{2,5\}.$$

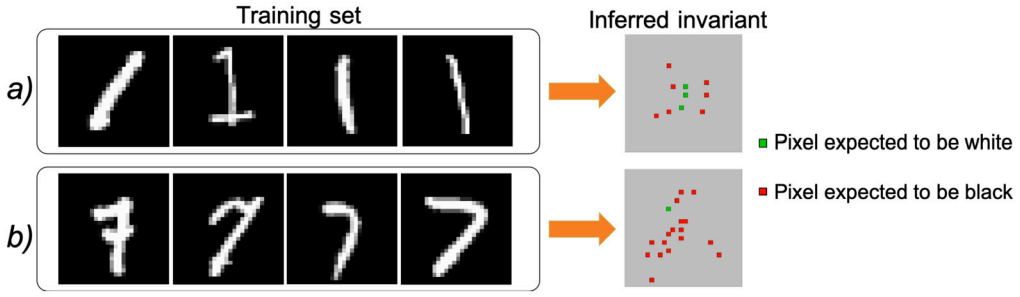


Fig. 4. Two training-data-dependent invariants for the MNIST dataset.

Table 1
Two misclassifications detected by the invariants in Fig. 4.

Input image	EO	PO	First matching invariant	Image vs invariant	Class of matching invariant	IO	CME
	1	7			1	*fail*	TP
	7	1			7	*fail*	TP

EO: Expected output; PO: output predicted by the CNN; IO: ICOS outcome; CME: Confusion Matrix Element.

This information represents upfront knowledge related to the system and its domain and does not depend on the training data and/or the CNN architecture. Google researchers confirm the advantages of user-defined rules in Deep Neural Networks (Seo et al., 2021), incorporating a rule encoder directly into the models. They also envisage an improvement in terms of domain adaptation using the rule strength, becoming robust to distribution shift.

These invariants are called *input-data-dependent invariants* (IDIs) since they depend on features specific of the *operational input*. Let F_{IDI} denote the set of mispredictions detectable through IDIs, called *input-related failures*. When the CNN output violates some IDIs, an input-related failure has occurred (i.e., the accuracy as for the detection of *input-related failures* is 100%, with no *false positives*); on the other hand, there may be many failures not detected through IDIs (i.e., a high number of false negatives). IDIs are defined manually.

4.3. Training-data-dependent invariants

The second source of knowledge considered is the training data. A CNN is trained on a *training set* reflecting the expected behavior in operation. ICOS automatically infers knowledge from the training set through *explainable* ML algorithms, such as *decision rules* or *decision trees*. It relates incorrect outputs to the observed inputs, encoding the inferred relation as a list of invariants.

In image classification, an invariant inferable from training data may consist of sets of pixels which, every time they have values beyond specific thresholds, make the CNN assign a certain class to the input image. These are *training-data-dependent invariants* (TDIs). A TDI violation occurs when the class assigned by the CNN is different from the class corresponding to the first invariant in the inferred ordered list, which matches the image in input.

As examples, Fig. 4 shows two invariants extracted from the training set of the MNIST dataset. Training data in Fig. 4 *a)*, labeled as digit 1, have an invariant represented here with green and red pixels: green ones are pixels expected to be close to white (pixel value close to 0) for an input image to be classified as 1; red pixels are pixels expected to be close to black (value close to 255) in a 1. Similarly for a digit 7 in Fig. 4 *b)*. Table 1 reports two misclassifications detected based on these invariants.

Let F_{TDI} denote the set of failures detectable by *training-data-dependent invariants*, called *training-related failures*. F_{TDI} may contain failures different from F_{IDI}, including false positives. Indeed, unlike the previous case, these are *likely* invariants, since the consequent of the rule is only probabilistically true given the antecedents. But they are expected to significantly improve the oracle in terms of low number of false negatives, at the price of more false positives. The idea is similar to "mirror programs" proposed by Qin et al. (2018), generated from training data, and used as pseudo oracles for ML programs testing.

The effectiveness of TDIs depends mostly on how well the training set represents operational inputs. A training set T is *representative* of operational inputs if the accuracy $Acc(T)$ achieved using it to train a classifier is such that $Acc(T) \approx Acc(T^*)$, where T^* is an ideal perfect training set (Borovicka et al., 2012). The failure detection ability of TDIs depends on how close $Acc(T)$ is to $Acc(T^*)$: TDIs can work well if T is sufficiently representative of actual inputs; otherwise, they are likely to lead to many *false positives*.

4.4. Algorithm-dependent invariants

A CNN can fail in ways that may not be detected by IDIs and TDIs. Specific characteristics of the CNN can be observed to look for possible patterns occurring when there is a failure, e.g., in the output of a certain layer of the neural network. For instance, Ma et al. (2019) show that for an input belonging to a specific class, a specific set of neurons is activated. They exploit these conditions to define *invariants* to detect adversarial samples. Observing how the algorithm behaves in the nominal case, one should be able to collect the *patterns* describing it. Based on the idea by Ma et al., ICOS is able to detect failures when such patterns are violated.

The output of the last layer of the CNN, namely a probability vector, is considered a relevant feature, as the definition of invariants based on the closeness of the output softmax values is a well-known way to define the uncertainty of a model (Feng et al., 2020, Weiss & Tonella, 2021a, 2021b). ICOS extracts the patterns observed in the nominal case from the probability values, notifying a failure when a pattern is violated. Nominal patterns are extracted by a *random forest* algorithm. These are *algorithm-dependent invariants* (ADIs). Let F_{ADI} be the set of failures detectable by ADIs, called *algorithm-related* failures. F_{ADI} may

Table 2
Examples of ICOS output for MNIST.

Test case	Label	CNN	ICOS	CME	Type of invariant violated
	9	0	*fail*	TP	IDI
	5	8	*pass*	FN	*none*
	8	7	*fail*	TP	TDI
	4	4	*fail*	FP	TDI
	6	8	*fail*	TP	ADI
	6	6	*fail*	FP	ADI

CME: Confusion Matrix Element.

contain failures different from those in the set $F_{IDI} \cup F_{TDI}$, including false positives. Therefore, as for IDIs, these are likely invariants.

Table 2 shows six examples from the MNIST dataset, along with the CNN predictions, ICOS response, and the type of invariant violated. For instance, the third row is a handwritten digit, whose label is 8; the digit is wrongly classified by the CNN as a 7, and the misclassification is correctly detected by ICOS (a true positive), through the violation of some TDIs. The last row represents a 6, it is correctly classified by the CNN, yet ICOS outcome is *fail* (a false positive).

A final remark is about the relation between the invariants: since the invariants are evaluated in sequence, the TDIs and ADIs aim to detect failures that "escape" the previous IDIs. It may well be that two inputs share some invariants. Moreover, it may well be that a CNN failure on a given input is detectable by violations of more than one type of invariants – ICOS notifies a detection at the first violation of any invariant (the possibility to adopt more elaborated strategies, e.g., a majority voting on violations of different invariants to notify a detection or associating confidence depending on how many violations are raised, is left to future investigation).

5. Evaluation

We formulate the Research Questions and outline the experimental settings. We then provide a detailed description of how ICOS is implemented, and we report a description of the baselines for comparisons.

5.1. Research questions

The evaluation targets three research questions:

RQ1 (Effectiveness): *How effective is ICOS at evaluating the operational accuracy, compared to a cross referencing oracle and to SelfChecker?*
RQ1 aims at assessing the extent to which ICOS faithfully evaluates the accuracy of the CNN in operation, compared to the baselines. As we expect that partitioning plays a role, we consider two partitioning criteria, detailed in Section 5.4. The experimental results are scrutinized to infer the contribution of each type of invariant to performance.

RQ2 (Sensitivity to invariants selection): *How does invariants selection influence the ICOS performance?*
RQ2 aims at assessing the extent to which criteria used to extract invariants influence the results. The focus is on TDI, which are controlled by tuning ICOS hyper-parameters (IDIs are covered by RQ1, while ADIs are built by a random forest algorithm, whose hyper-parameters tuning is outside our scope). ICOS hyper-parameters are *support* and *confidence* of rules to extract. Support is the number of samples covered by the rule: it refers to how often the rule appears in the dataset. Confidence is the ratio of the times the rule gives a correct prediction to the times it appears (namely, divided by its support). We consider three combinations of support and confidence, corresponding to three invariants' selection criteria.

RQ3 (Robustness): *How does ICOS perform in presence of label shift?*
RQ3 aims to evaluate ICOS when operational inputs strongly diverge from those used in training, namely in presence of *label shift*. The shift is emulated by mutating the training dataset, as done by Li et al. (2019).

5.2. Experimental subjects

The subjects are twelve Convolutional Neural Networks, three for each of the following four datasets: MNIST (LeCun & Cortes, 2010), CIFAR10 (Krizhevsky, 2009), ten synsets of ImageNet (Deng et al., 2009), and CIFAR100. Table 3 reports the number of layers and of parameters for each CNN. Further details are made publicly available on Github.[1] We experiment with datasets and CNNs of various complexities to guarantee diversity and improve the generalizability of results. The datasets are diverse in the number of pixels and classes, and the networks are diverse in the number of layers and parameters.

We split the datasets into training, validation and operational sets. For *validation set*, we mean the set of images used to evaluate the CNN and to compute the generalization error. For *operational set*, we mean the unlabeled images used to assess the operational accuracy. This splitting is done to avoid biases: Recht et al. (2019) showed in fact that using previously unseen test sets in the assessment causes the real accuracy to drop by 3% to 15% on CIFAR10 and by 11% to 14% on ImageNet, compared to the claimed accuracy.

The training set size is set as follows:

- For MNIST and CIFAR10: 40% of the available dataset size (28,000 and 24,000, respectively);
  The ImageNet and CIFAR100 datasets need bigger training sets to have an acceptable accuracy (we assume at least 0.5). Therefore, the training set sizes are set as follows.
- For ImageNet: we consider 10 synsets, resulting in a training set of about 3,000 samples, which is 60% of the dataset;
- For CIFAR100: 40,000 samples, yielding at least 400 examples per class, which is 66% of the dataset.

The validation set size is:

- For MNIST and CIFAR10: 2,500 samples (250 examples for each class);
- For ImageNet: 1,200 samples (120 examples for each class);
- For CIFAR100: 5,000 samples (50 examples for each class).

Table 3
Characteristics of the twelve experimental subjects.

CNN	Dataset	# of Layers	# of Parameters	Accuracy on the validation set
A		7	6,237	0.965
B	MNIST	6	97,114	0.968
C		8	545,546	0.964
D		13	1,084,234	0.697
E	CIFAR10	10	258,762	0.657
F		12	550,570	0.625
G		13	476,874	0.664
H	ImageNet	9	1,307,338	0.578
I		13	4,247,985	0.602
L		16	15,047,588	0.552
M	CIFAR100	9	564,484	0.522
N		13	1,465,220	0.579

Operational samples are selected randomly (without replacement) from those remaining after constructing the training and validation sets (in the number of 39,500 for MNIST, 33,500 for CIFAR10, 3,072 for ImageNet, and 15,000 for CIFAR100). The operational set size is so set:

- For MNIST and for CIFAR10: 20,000 samples;
- For ImageNet: 2,000 samples;
- For CIFAR100: 3,000 samples.

Each experiment is repeated 30 times for statistical significance. The random selection procedure is performed before each repetition. The labels of the selected samples are removed to emulate unknown expected output. Table 3 also provides a quantitative evaluation of each CNN's accuracy based on the scores achieved on the corresponding validation set. Since the operational dataset changes with each repetition, we do not report the accuracy there.

5.3. Evaluation metrics

The metric to evaluate ICOS performance in predicting the operational accuracy is the *Mean Absolute Error* (MAE), computed as:

$$MAE(\hat{\phi}) = \frac{1}{N}\sum_{i=1}^{N} |\hat{\phi}_i - \phi_i| \tag{3}$$

where N is the number of repetitions ($N = 30$ in our experiments), ϕ_i is the actual accuracy computed on the operational dataset sampled during the i^{th} repetition, and $\hat{\phi}_i$ is the predicted accuracy computed through oracle predictions on the same operational dataset.

We consider also the metrics used by Xiao et al. (2021) to evaluate SelfChecker:

- **True Positive Rate (TPR)**: the ratio of failures correctly detected out of all failures ($TPR = TP/(TP + FN)$);
- **False Positive Rate (FPR)**: $FPR = FP/(TN + FP)$
- **F1-score (F1)**: $F1 = (2 \times TP)/(2 \times TP + FN + FP)$

The goal is to achieve high TPR, low FPR, and high F1.

5.4. ICOS implementation

The invariants considered by ICOS are defined/extracted as follows.

Input-data-dependent invariants IDIs are defined assuming an operational domain with various input sources (like the examples in subsection 4.2), where each source produces output belonging to a certain subset of labels. This way, we can consider disjoint partitions.

For the evaluation, we consider three partitions for MNIST:

- set of all digits without straight lines (0, 3, 6, 8, 9);
- set of all digits with straight lines (1, 4, 7) only;
- set of remaining digits (2, 5);

the following two partitions for CIFAR-10 and ImageNet:

- set of all animals (CIFAR10: *dog, frog, horse, bird, cat, deer*; ImageNet: *armadillo, cat, gorilla, hyena, zebra*);
- set of all non-animals (CIFAR10: *airplane, car, ship, truck*; ImageNet: *bikes, missile, revolver, ships, taxi*);

and the following seven partitions for CIFAR100 (defined for brevity on the coarse-grain classes):

- P_1: aquatic mammals, fish;
- P_2: flowers, fruit, vegetables, food containers;
- P_3: large carnivores, large omnivores, herbivores, medium-sized mammals, small mammals;
- P_4: insects, non-insect invertebrates, reptiles;
- P_5: vehicles 1, vehicles 2;
- P_6: large outdoor objects, large natural outdoor scenes, trees;
- P_7: household electrical devices, furniture, people.

We call this partitioning criterion *fine partitioning* (*fp*).

To show the impact of partitioning (different partitionings can yield different results), we also consider a second criterion applicable to all datasets, which just splits the operational set into two partitions:

- Partition P_1 is made by the following sets:
  - $\{0, 1, 2, 3, 4\}$ for MNIST;
  - $\{airplane, automobile, bird, cat, deer\}$ for CIFAR10;
  - $\{armadillo, bikes, cat, gorilla, hyena\}$ for ImageNet;
  - the first 50 classes $\{apple, ..., mountain\}$ for CIFAR100.
- Partition P_2 is made by:
  - $\{5, 6, 7, 8, 9\}$ for MNIST;
  - $\{dog, frog, horse, ship, truck\}$ for CIFAR10;
  - $\{missile, revolver, ships, taxi, zebra\}$ for ImageNet;
  - the last 50 classes $\{mouse, ..., worm\}$ for CIFAR100.

We call this partitioning criterion *equal partitioning* (*ep*).

Training-data-dependent invariants Training-data-dependent rules are derived by applying the **C4.5** algorithm (Quinlan, 1993), an implementation of decision trees known for its ability to generate classifiers interpretable by humans (Alonso et al., 2018); the setting is in Table 4.

Table 4
C4.5 parameter configuration.

Parameter	Value
Quality measure	*Gini index*
Pruning method	*No pruning*
Reduced error pruning	*false*
Min number records per node	*5*

The list of rules we obtain represents conditions satisfied by samples in the training set. Every time a condition is violated, a failure is detected. As an instance, consider again Fig. 4.a; the following invariant states that when the values of pixels (shown between "$" symbols) are beyond given thresholds, a CNN output different from '1' means a failure occurred (image pixels are ordered left to right and top to bottom):

$$fail \text{ :- } \$514\$ <= 253 \wedge \$406\$ > 157 \wedge \$539\$ <= 111 \wedge$$

$$\$411\$ <= 61 \wedge \$347\$ <= 2 \wedge \$206\$ <= 0 \wedge \$327\$ <= 7$$

$$\wedge \$522\$ <= 0 \wedge \$489\$ > 56 \wedge \$350\$ > 165, \ output \neq 1$$

The set of selected invariants depends on the values of confidence and support. For RQ1, we set the minimum confidence value at $C = 0.99$ for all models and datasets, and, since the discriminating power of the rules depends on the support on which such a confidence is obtained, we filtered out low-support rules with the following criteria:

- For MNIST, we consider the minimum support, under $C \geq 0.99$, for which at least the 50% of operational samples are evaluated;
- For CIFAR10, ImageNet and CIFAR100, whose rules have a low average support, we consider the median support over the rules with $C \geq 0.99$.

The resulting values are: $S = 140$ for MNIST, $S = 14$ for CIFAR10, $S = 11$ for ImageNet, and $S = 8$ for CIFAR100.[2]

[2] These parameters can be fine-tuned for improving performance on a case-by-case basis. In RQ1 experiments, we kept this configuration. For RQ2, further configurations are considered.

Table 5
Random forest parameter configuration for ADIs extraction.

Parameter	Value
Split criterion	*Information Gain Ratio*
Number of models	*100*

For RQ2, confidence and support are set in three different ways to study the sensitivity to invariants selection criteria:

- *Criterion 1*: Select invariants with high confidence ($C = 0.99$) and high support, greater than 140 for MNIST, $S = 14$ for CIFAR10, $S = 11$ for ImageNet, and $S = 8$ for CIFAR100;
- *Criterion 2*: Select invariants with confidence greater than 0.99 ($C = 0.99$), regardless of support ($S = 0$); this is expected to give a wider set of rules (which can allow detecting more failures), but those with low support are expected to provide more false positives and to underestimate the CNN accuracy.
- *Criterion 3*: Select all the invariants, regardless their confidence ($C = 0$) and support ($S = 0$).

Algorithm-dependent invariants ADIs extraction uses the KNIME implementation of Random Forest (Berthold et al., 2007), with default parameter values (reported in Table 5) and considers the validation set as the training set. We collect the output of the last layer of each CNN corresponding to each element of the validation set. We use this dataset as the training set for Random Forest. The aim is to train the Random Forest algorithm to detect patterns corresponding to correct outcomes, so as to find misclassifications each time they are not satisfied.

5.5. Cross-referencing oracle

As baseline for comparison, like Srisakaokul et al. (2018) we use a cross-referencing oracle (CRO) for *multiple-implementation testing*. For each dataset, we implemented a *majority oracle* with the three corresponding CNNs of Table 3, adjudging as correct output the most voted value. When the three models fully disagree, CRO takes no decision.

5.6. SelfChecker

SelfChecker (SC) is an automatic oracle for *in deployment* evaluation of CNNs (Xiao et al., 2021). SC evaluates the final output provided by the monitored CNN considering features extracted from the internal layers. It also suggests alternative predictions in case of misclassifications detected. The workflow of SC is shown in Fig. 5. First, the training set is used to compute layer-wise density distributions for each layer using *kernel density estimation* (KDE). Then, an optimal set of layers is selected exploiting the predictions on the validation set. Finally, the density values of the selected layers are used to decide whether to provide an alarm and an alternative prediction (advice) in case of misclassification.

6. Results

We present the results of the experimentation, describe the findings, and discuss the advantages and limitations compared to baselines.

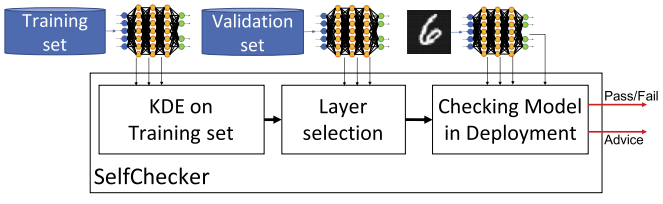


Fig. 5. Workflow of SelfChecker.

6.1. RQ1: effectiveness

The results for the MAE metric for each dataset are shown in Fig. 6:

- On MNIST (Fig. 6a), ICOS_{fp} shows the best values of MAE. ICOS_{ep} shows values lower than the baselines for CNN A and B;
- On CIFAR10 (Fig. 6b), ICOS_{fp} outperforms the other techniques, while ICOS_{ep} shows values worse than SC, except for CNN D;
- On ImageNet (Fig. 6c), SC performs significantly better for CNN G and slightly better for CNN H compared to ICOS. ICOS_{fp} performs slightly better on CNN I;
- On CIFAR100 (Fig. 6d), ICOS_{fp} performs better than the other oracles for models L and M; SC shows the best value for model N.

Overall, while SelfChecker performs particularly well on complex datasets (for dimension of the images and number of classes) and networks (for layers and parameters, like CNN C for MNIST), ICOS_{fp} is more stable in terms of mean absolute error, for which it shows always low values.

To assess the statistical significance, we run the Friedman test (Iman & Davenport, 1980) on the absolute error (i.e., the *offset*) obtained in each repetition, each automatic oracle, and each subject, considering a significance level equals to 0.05. The test assesses if there is a significant difference among any of the five automatic oracles. The *p-value* = 2.2E-16 allows rejecting the null hypothesis that "there is no difference among the offsets".

Fig. 7 shows the critical differences resulting from the *post hoc* Nemenyi's test (Calvo & Santafe, 2015) on the offset values. Automatic oracles with no significant difference are grouped using a bold horizontal line – the higher the distance between two oracles (distance being the average ranking), the smaller the $p - value$ for the null hypothesis of equal performance. In the figure, oracles are ordered from the worst to the best (higher values of offset mean lower values of predicted accuracy). The test shows there is no significant difference only between ICOS_{ep} and SC, which are confirmed to perform similarly in evaluating the accuracy of the CNN, while ICOS_{fp} is the best one due to the fine partitioning.

For a detailed view, Tables 6, 7, and 8 report the average TPR, FPR, and F1 over 30 repetitions. The best values per metric and per CNN are in bold.

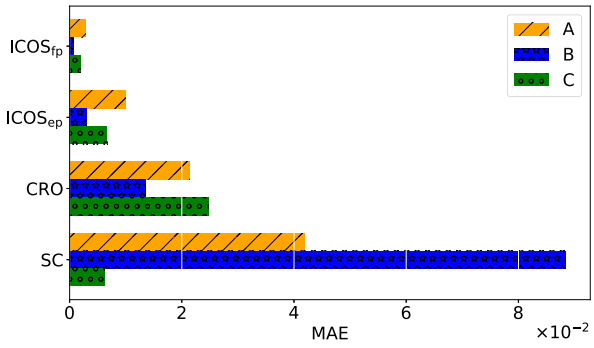
As for TPR (Table 6), ICOS_{ep} shows the best values 5 times, SC 6 times, and ICOS_{fp} just 1 time. CRO shows the best values of FPR, but with a TPR lower than 0.5 except for CNN A.

As for FPR (Table 7), excluding CRO, ICOS shows the best values 5 times (3 times ICOS_{fp}, 2 times ICOS_{ep}), while SC shows the best values 7 times. On the opposite, the higher number of false positives for simpler models causes a strong divergence of the predicted accuracy (high MAE values). ICOS_{fp} shows better values in terms of MAE due to the better balancing of TPR and FPR.

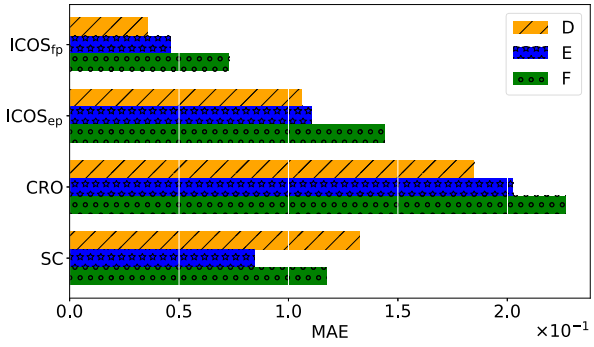
As for the F1 metric (Table 8), ICOS_{fp} has the best values for MNIST CNNs, ICOS_{ep} has the best value for CNN H and L (CIFAR10). SC has the best values for CNN G, I, M, and N.

ICOS' results in terms of F1-score highlight the importance of choosing IDIs effectively. For networks classifying MNIST images, fine partitioning is more effective, while in the other cases (CNN D, E, F, H, and I) the equal partitioning works better. Specifically, since F1-score represents the balance between Recall and Precision, ICOS is not always able to keep this ratio high, but it can still balance TP and FP to have faithful accuracy estimates (low MAE). About the baselines, SC achieves the best performance in terms of TPR. These results depend on two main factors:

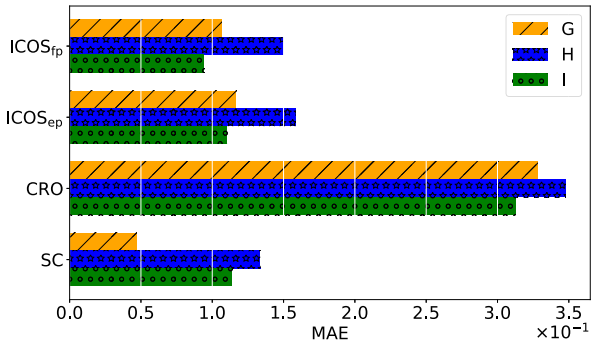
- For CNN A and B, SC is more prone to classify the operational examples as failures. This gives a high TPR but also the worst False Positive Rate (FPR) for these networks, and consequently the worst Mean Absolute Error (MAE). The reason behind these results can be



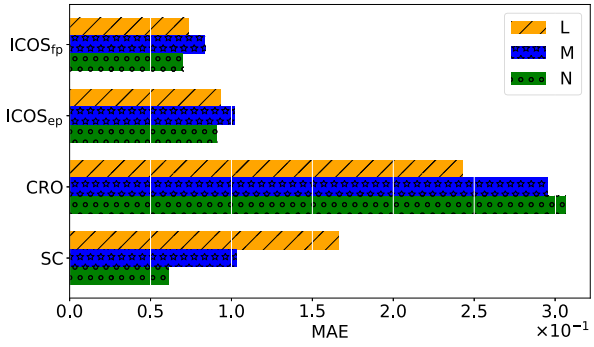
(a) MNIST



(b) CIFAR10



(c) ImageNet



(d) CIFAR100

Fig. 6. RQ1: ICOS *vs* CRO and SelfChecker - MAE.

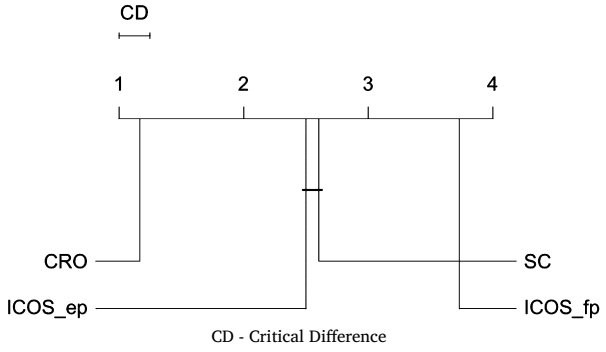


Fig. 7. RQ1: Plot of *post hoc* Nemenyi's test on offset values.

related to the low complexity of the MNIST dataset and the CNN and consequently a low number of features for the SC classification task.

- For CNN I-N, SC shows good values of TPR due to the higher complexity of CNNs and datasets, and consequently to a high number of features about the classification accuracy of these networks.

The low values of FPR shown by CRO with respect to the other techniques depend intuitively on the networks considered for the majority voting. This means that the CNNs will be more prone to agree than to disagree since they share the same knowledge (they are trained on the same training dataset). For this reason, they lead to miss more failures than the other techniques. On the contrary, a failure detected by CRO is more robustly related to an actual misclassification.

Contributions of the invariants to misclassifications detection Fig. 8 shows the contribution to the TPs of the three types of invariants considered for ICOS, averaged over the three CNNs for each dataset. IDIs contribute

Table 6
RQ1: True Positive Rate.

	TPR ↑			
CNN	ICOS_fp	ICOS_ep	CRO	SC
A	0.7840	0.6252	0.5042	**0.7997**
B	0.7641	0.6609	0.4078	**0.8364**
C	**0.7704**	0.6495	0.3570	0.7237
D	0.6143	**0.8487**	0.3621	0.8318
E	0.6623	**0.8498**	0.4282	0.7927
F	0.6921	**0.8913**	0.4142	0.8138
G	0.8075	**0.8243**	0.3481	0.8123
H	0.8439	**0.8582**	0.3849	0.8245
I	0.7779	0.8151	0.3455	**0.8405**
L	0.7843	0.8474	0.2945	**0.8903**
M	0.8168	0.8622	0.4166	**0.8893**
N	0.7990	0.8476	0.3505	**0.8700**

Table 7
RQ1: False Positive Rate.

	FPR ↓			
CNN	ICOS_fp	ICOS_ep	CRO	SC
A	*0.0075*	0.0077	**0.0040**	0.0532
B	0.0085	*0.0083*	**0.0071**	0.0958
C	0.0076	0.0077	**0.0023**	*0.0052*
D	*0.2166*	0.2168	**0.0440**	0.3460
E	*0.2441*	0.2448	**0.0379**	0.2451
F	0.2854	0.2851	**0.0294**	*0.2709*
G	0.3766	0.3803	**0.0483**	*0.2584*
H	0.4638	0.4654	**0.0415**	*0.4543*
I	0.3603	0.3572	**0.0490**	*0.3420*
L	0.2209	*0.2190*	**0.0391**	0.3073
M	0.2810	0.2817	**0.0230**	*0.2607*
N	0.2619	0.2615	**0.0195**	*0.1946*

Table 8
RQ1: F1-score.

	F1 ↑			
CNN	ICOS_fp	ICOS_ep	CRO	SC
A	**0.8086**	0.7007	0.6318	0.5455
B	**0.7581**	0.6928	0.4956	0.3276
C	**0.7895**	0.7082	0.5024	0.7841
D	0.5801	**0.7212**	0.4777	0.7176
E	0.6200	**0.7312**	0.5560	0.7083
F	0.6282	**0.7423**	0.5538	0.6927
G	0.7260	0.7339	0.4807	**0.7739**
H	0.7351	**0.7420**	0.5250	0.7288
I	0.7049	0.7268	0.4756	**0.7470**
L	0.7069	**0.7442**	0.4095	0.7143
M	0.7435	0.7705	0.5637	**0.7931**
N	0.7365	0.7631	0.4985	**0.8100**

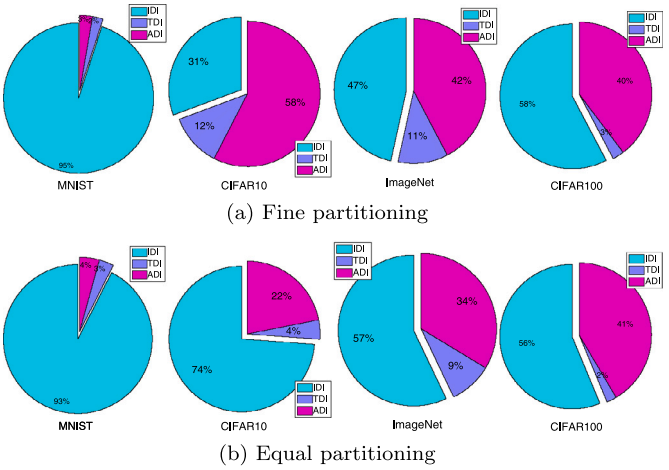


Fig. 8. RQ1: Contribution of the three types of invariants to true positives.

more, except for CIFAR 10 with fine partitioning. Although in the case of MNIST the contribution of the automatic invariants (TDI and ADI) is small, it should be considered that the CNNs are very accurate (more than 0.96 on the validation data, with 790 actual misclassification on average for each testing session), and detecting failures is hard.

In the other cases, where the accuracy is lower (on average: 0.65 for CIFAR10 CNNs, with 6,650 actual misclassification for each repetition; 0.55 for ImageNet CNNs, with 960 actual failures for each repetition; and 0.61 for CIFAR100 CNNs, with 9,125 actual failure for each repetition) the contribution of the automatic invariants is more evident.

Fig. 9 reports the sole contribution of the automatic invariants. Altogether, TDIs and ADIs detect 13% of the total misclassifications for MNIST, 58% for CIFAR10, 71% for ImageNet, and 72% for CIFAR100. Thus, the automatic invariants of ICOS can significantly contribute: although the tuning is coarse-grained for CIFAR10, ImageNet and CIFAR100 (compared to the one used for MNIST), TDIs and ADIs detect more than 50% of misclassifications.

RQ1 answer

ICOS provides a better prediction accuracy compared to CRO. The $ICOS_{fp}$ variant also provides better accuracy than SelfChecker, while the $ICOS_{ep}$ variant is statistically equivalent to SelfChecker.
Overall, ICOS performs comparably to SelfChecker (and better than CRO), exhibiting higher stability over a variety of datasets and CNNs.

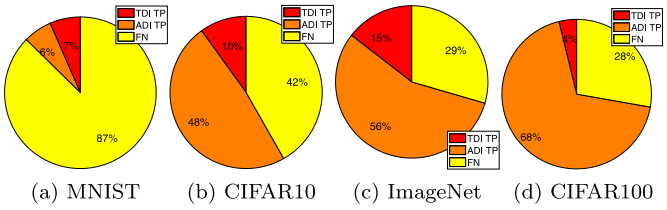


Fig. 9. RQ1: Contribution of automatic invariants to true positives (total failures = TP + FN).

6.2. RQ2: influence of invariants selection criteria

The results can be influenced by the selection of invariants. We filter TDIs by tuning both minimum confidence and support by the three criteria described in Section 5, considering *fine partitioning*. Fig. 10 reports TPR and FPR for each criterion, and the results obtained exploiting CRO and SC.

The three criteria are characterized by intuitive trends: adding more invariants from criterion 1 to 3, the TPR increases considerably; on the other hand, FPR increases too. In particular, relaxing the constraint about TDI support (ICOS, criterion 2), TPR increases by about 6% on average (9% for MNIST, 11% for CIFAR10, 3% for ImageNet, 2% for CIFAR100). Considering all TDIs (ICOS, criterion 3), TPR increases by about 26% on average (20% for MNIST, 42% for CIFAR10, 19% for ImageNet, 23% for CIFAR100). Contextually, with ICOS criterion 2 the FPR increases by five times for MNIST, 53% for CIFAR10, 30% for ImageNet, and 92% for CIFAR100. Considering all TDIs (ICOS, criterion 3) it increases by almost fifteen times for MNIST, 213% for CIFAR10, 120% for ImageNet, and 220% for CIFAR100.

The comparison with ICOS and SC highlights that if we increase the number of TDIs considered, ICOS achieves a better TPR paying in terms of False Positives.

RQ2 answer

By selecting more invariants (ICOS, criterion 2), the TPR increases by 6% on average, but we pay in terms of False Positives, which can increase up to 5 times. By taking all the invariants (ICOS, criterion 3, actually corresponding to no selection), TPR increases by 26% on average, but FPR can increase up to 15 times. Invariant selection is therefore needed to avoid excessively high FPR.

6.3. RQ3: robustness

An analysis of robustness is performed on ICOS and SC to evaluate the performance of these approaches in case of an operational dataset very different from the training dataset. To this purpose, we build a mutated training dataset, as by Li et al. (2019) to emulate a divergence between training and operational data. In detail, we switched the label of samples in the training dataset, while the operational dataset is left unchanged. We considered three different label switches (2,7), (5,6), and (6,8) for the MNIST dataset. The model considered for this analysis is CNN C, due to the good results shown by SC in the previous subsection. The three versions of CNN C trained on the mutated datasets are called respectively C_1, C_2, and C_3. As shown in Fig. 11, the performance of the three techniques depends on the specific mutation adopted:

- in the C_1 case, ICOS outperforms SC both in terms of MAE, thanks to the IDI, which is able to detect failures due to the label switching;
- in the C_2 case, $ICOS_{ep}$ degrades its performance due to the ineffectiveness of its IDI to the label shift;
- in the C_3 case, the three techniques perform badly in the same way.

To assess the statistical significance, we run the Friedman test on the absolute error obtained in each repetition for each automatic oracle considering a significance level equal to 0.05. The *p-value* = 9.3E-15

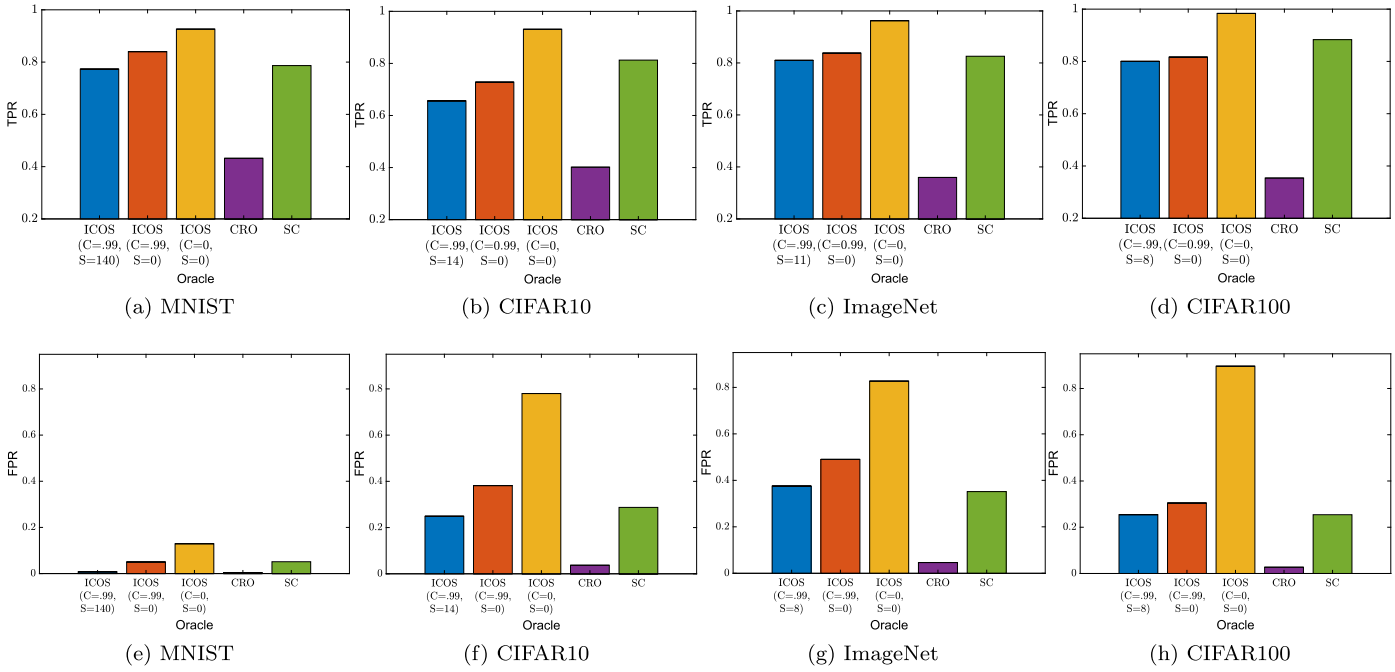


Fig. 10. RQ2: ICOS True Positive Rate and False Positive Rate for various invariant selection criteria.

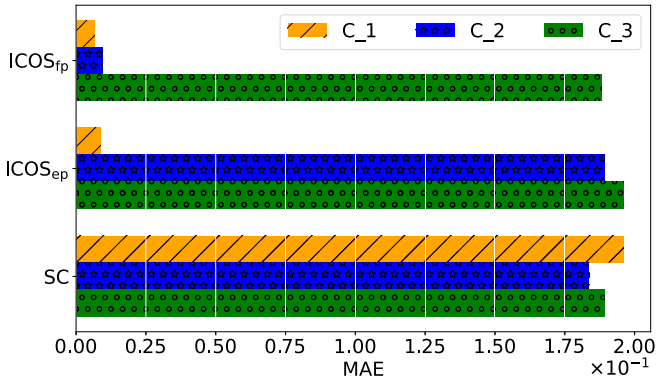


Fig. 11. RQ3: ICOS *vs* SelfChecker.

allows rejecting the null hypothesis that "there is no difference among the offsets". Fig. 12 shows the critical differences resulting from the Nemenyi test. The oracles are ordered from the worst to the best. The test shows that all the differences are significant.

The manual definition of IDIs can be decisive in case of unexpected inputs in operation. In this experimentation, the defined IDIs are very simple. Despite their simplicity, they can actually save CNN's performance in operation when some unexpected happens. In case of a strong distribution shift, with consequent label shift, IDIs can help understand misclassifications depending on label shifts. SC, similarly to TDIs and ADIs, are not able to be robust against this kind of issue, due to their dependence on the training set, and on the model under test. Although the definition of IDIs can be costly, they can be very useful when unexpected phenomena like label shift affect data in operation.

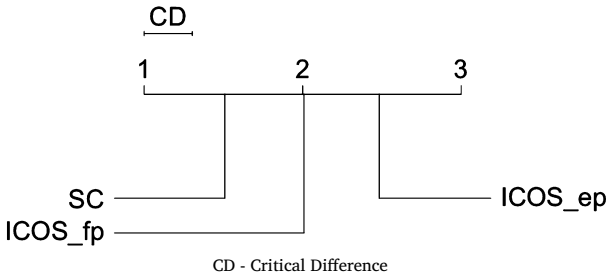


Fig. 12. RQ3: Plot of *post hoc* Nemenyi's test on offset values.

RQ3 answer

Knowledge about input data, expressed by user-defined invariants, make ICOS accuracy assessment more stable than SelfChecker, increasing robustness against unexpected phenomena in operation, like label shift.

6.4. Discussion

Compared to the baselines, ICOS balances TP and FP better, providing more accurate estimates and bringing to low MAE (RQ1). As for *training-data-dependent invariants*, results show that both confidence and support have a relevant impact on detections. TDIs with low confidence and low support are representative of rare samples in the training set, typically the ones more likely misclassified. On the other hand, these invariants are more prone to generate FPs. A coarse-grain selection of TDIs leads to a high ability in finding failures, while a fine-grained selection leads to a low number of False Positives. Depending on the testing objective, users can adjust configuration parameters to have more conservative or more alerting operational accuracy estimates (RQ2).

As for *algorithm-dependent invariants*, starting from only 2,500 samples in the validation set, the automatically inferred ADIs spot several failures unrevealed by the two previous stages. The number of detections is clearly higher when the SUT has a higher number of failures (Fig. 9*b* and 9*c*), yet they detect some failures also for very accurate systems. (Fig. 9*a*).

About baselines, SC achieves the best performance in terms of TPR, but it often shows the worst values of FPR resulting in worse values of MAE. Results from ICOS are more robust than SC in case of unexpected phenomena in the operational environment because of IDI (RQ3). For this reason, ICOS is preferred when the user can extract or define IDI from the available knowledge of the operational environment. Due to the majority voting, CRO's networks are more prone to agree than to disagree since they share the same knowledge. This results in missing

more failures than the other techniques. While its accuracy estimates are worse than the others, a failure detected by CRO is more robustly related to an actual misclassification.

A final remark is about *interpretability* of the predictions of the oracle surrogate. The output of ICOS is, to some extent, interpretable. For instance, considering the results of CNN C, we saw that the more frequent misclassification detected by IDIs is related to samples representing a 9 classified as 4. Reasonably, this condition depends on a certain similarity between the two samples and a low representativeness of the correspondent values in the training set. When the failure is caused by a *training-data-dependent* invariant violation, we could spot which parts of the input are particularly prone to cause the failure. For instance, if the greatest part of the white pixels is very close to each other, and they are clustered in the center of the image, reasonably the image is a 1. This feedback is not possible with a Cross-Referencing Oracle, in which case a detection means that a subset of the versions disagrees with the SUT. One could just derive something like a *training-data-dependent* invariant, considering that the alternatives are trained on the same training set: "each time the SUT disagrees with the majority the outcome could be considered as *fail*". As future development, it makes sense to consider CRO as part of ICOS to complement the ability to extract training-data-dependent invariants.

The automatic oracles have a cost for knowledge extraction from the training dataset:

- ICOS requires training for extracting TDIs and ADIs;
- SelfChecker requires to compute layer-wise density distributions for each layer and each image in the training dataset;
- Cross-Referencing Oracles require training for the diverse implementations.

CRO can be considered more costly in terms of training compared to ICOS and SC. However, they are characterized by additional cost, concerning the manual invariants definition (IDIs for ICOS) and parameters tuning (confidence and support of TDIs for ICOS, and layer selection for SC). The latter cost occurs *una tantum*, while the training cost is required whenever the training and verification sets are altered. Since they can be carried out mainly simultaneously, the training cost can be viewed as reasonable when compared to the expense of training the CNN. For these reasons, one possibility is to use SC, and in general fully automated approaches, when the knowledge about the operational environment is scarce, or during the first deployments of the system to allow the operational team to learn about the execution context. ICOS may be preferable (or used in conjunction) for a more accurate output evaluation under varying operating conditions and/or in a continuous refinement perspective.

7. Threats to validity

Internal validity The presented results assume the correctness of the implementation of ICOS, as well as of the experimental code. We made the code available so as to favor verifiability and reproducibility.[1] The correctness of the CNNs can be a further internal threat; the models used in experiments are derived from public repositories, and training was performed from scratch in order to avoid biases. About IDIs, different partitioning can provide different results; a detailed investigation of partitioning is a matter of future work.

Construct validity The training-data-dependent and algorithm-dependent invariants inferred by ICOS rely on well-established algorithms, namely *C4.5* and *Random Forest*. These have been used with the default setting as provided by KNIME, since we focused primarily on investigating the ICOS's hyperparameters (i.e., support and confidence) rather than hyperparameters of such external algorithms which ICOS relies on. Consequently, results would be different if other algorithms and/or other hyperparameters settings are adopted.

External validity We considered nine CNNs and three well-known datasets; however, results need to be confirmed across different CNN architectures and, more interestingly, across potential application domains (e.g., autonomous driving) for improving generalizability.

8. Conclusions

Automatic assessment of the accuracy of Machine Learning image classification systems with arbitrary inputs is a challenging task, due to the oracle problem. We presented ICOS, an oracle surrogate for image classification systems, showing that its mechanism to infer likely invariants helps assessing the actual accuracy provided by CNNs in operation, by detecting misclassifications when the expected output is unknown.

We have found that the invariant selection impacts ICOS performance: by selecting more invariants, the ability to detect failures of the test oracle improves, although the number of false positives increases (causing the divergence of the estimated operational accuracy). The impact varies depending on the accuracy of the model under assessment, but mostly on the representativeness of the training set with respect to the operational inputs. We have also seen that invariants with low support can detect more complex misclassifications in operation, but they cause a large number of false positives.

ICOS can be tuned to provide a trade-off between true and false positives to make the estimates of the operational accuracy converge to the true value. ICOS can be particularly effective for assessing the operational accuracy of an ML system since it can be robust to unexpected phenomena that are typical of the operational environment (like distribution and label shifts). This is possible by incorporating operational features in the considered invariants.

The assessment of the accuracy of CNNs, and of DNNs in general, can be of interest beyond the IC domain. For instance, the steering angle prediction in the Autonomous Driving domain - a regression problem - is a potential application. To this purpose, ICOS requires specific customization: IDIs must be defined in the Autonomous Driving context (e.g. based on traffic rules); TDI and ADI must be re-conceived to work on continuous values. The integration of ICOS in the life cycle of the DNNs is an opportunity to deal with the collection and evaluation of operational features, which can be useful but needs specific reasoning to infer constraints. An interesting research direction is to apply inferential engines on operational features to automatically extract IDIs which can help detect failures, making the accuracy estimates more faithful.

CRediT authorship contribution statement

Antonio Guerriero: Conceptualization, Methodology, Investigation, Software, Formal analysis, Validation, Writing, Visualization. **Michael R. Lyu:** Writing, Visualization, Supervision. **Roberto Pietrantuono:** Methodology, Supervision, Funding acquisition, Project administration, Writing, Visualization. **Stefano Russo:** Methodology, Supervision, Funding acquisition, Resources, Writing, Visualization.

Declaration of competing interest

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.

Data availability

Data has been shared on GitHub (link in the paper).

Acknowledgements

This work has been co-funded by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 871342 "uDEVOPS". Work by A. Guerriero has also been funded by the COSMIC project of the DIETI department.

References

Alla, S., & Adari, S. K. (2021). *What is MLOps?* Berkeley, CA: Apress (pp. 79–124).

Alonso, J. M., Ramos-Soto, A., Castiello, C., & Mencar, C. (2018). Explainable AI beer style classifier. In *SICSA reasoning, learning and explainability workshop*. CEUR.

Ashmore, R., Calinescu, R., & Paterson, C. (2021). Assuring the machine learning lifecycle: Desiderata, methods, and challenges. *ACM Computing Surveys*, *54*(5).

Berthold, M. R., Cebron, N., Dill, F., Gabriel, T. R., Kötter, T., Meinl, T., Ohl, P., Sieb, C., Thiel, K., & Wiswedel, B. (2007). KNIME: The Konstanz Information Miner. In *Studies in classification, data analysis, and knowledge organization*. Springer.

Borovicka, T., Jirina, M., & Kordík, P. (2012). Selecting representative data sets. In *Advances in data mining knowledge discovery and applications* (pp. 43–70). IntechOpen.

Calvo, B., & Santafe, G. (2015). scmamp: Statistical comparison of multiple algorithms in multiple problems. *The R Journal*, *8*(1), 248–256.

Choudhary, A. (2022). Google AI researchers present a new method to train models, 'DeepCTRL'. *Analytics India Magazine (online)*, *13*. https://analyticsindiamag.com/google-ai-researchers-present-a-new-method-to-train-models-deepctrl/. (Accessed 3 October 2022).

Corbière, C., Thome, N., Bar-Hen, A., Cord, M., & Pérez, P. (2019). Addressing failure prediction by learning model confidence. In *Proceedings of the 33rd international conference on neural information processing systems*. Curran Associates Inc.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition* (pp. 248–255).

Dwarakanath, A., Ahuja, M., Sikand, S., Rao, R. M., Bose, R. P. J. C., Dubash, N., & Podder, S. (2018). Identifying implementation bugs in machine learning based image classifiers using metamorphic testing. In *Proc. 27th ACM SIGSOFT international symposium on software testing and analysis (ISSTA)* (pp. 118–128). ACM.

Feng, Y., Shi, Q., Gao, X., Wan, J., Fang, C., & Chen, Z. (2020). DeepGini: Prioritizing massive tests to enhance the robustness of deep neural networks. In *Proc. 29th ACM SIGSOFT international symposium on software testing and analysis (ISSTA)* (pp. 177–188). ACM.

Frankl, P. G., Hamlet, R. G., Littlewood, B., & Strigini, L. (1998). Evaluating testing methods by delivered reliability. *IEEE Transactions on Software Engineering*, *24*(8), 586–601.

Garg, S., Wu, Y., Balakrishnan, S., & Lipton, Z. C. (2020). A unified view of label shift estimation. In *34th conference on neural information processing system (NeurIPS)* (pp. 3290–3300).

Google (2022). MLOps: Continuous delivery and automation pipelines in machine learning. https://cloud.google.com/solutions/machine-learning/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning. (Accessed 5 July 2022).

Guerriero, A. (2020). Reliability evaluation of ML systems, the oracle problem. In *IEEE international symposium on software reliability engineering workshops (ISSREW)* (pp. 127–130). IEEE.

Guerriero, A., Pietrantuono, R., & Russo, S. (2021). Operation is the hardest teacher: Estimating DNN accuracy looking for mispredictions. In *IEEE/ACM 43rd international conference on software engineering (ICSE)* (pp. 348–358). IEEE.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE international conference on computer vision (ICCV)* (pp. 1026–1034). IEEE.

Iman, R. L., & Davenport, J. M. (1980). Approximations of the critical region of the fbietkan statistic. *Communications in Statistics. Theory and Methods*, *9*(6), 571–595.

Jahangirova, G., Stocco, A., & Tonella, P. (2021). Quality metrics and oracles for autonomous vehicles testing. In *2021 14th IEEE conference on software testing, verification and validation (ICST)* (pp. 194–204).

Jiang, M., Chen, T. Y., & Wang, S. (2022). On the effectiveness of testing sentiment analysis systems with metamorphic testing. *Information and Software Technology*, *150*, Article 106966.

Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. Technical Report TR-2009, University of Toronto. https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf. (Accessed 5 July 2022).

Kühl, N., Goutier, M., Baier, L., Wolff, C., & Martin, D. (2020). Human vs. supervised machine learning: Who learns patterns faster? CoRR, arXiv:2012.03661 [abs].

Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

LeCun, Y., & Cortes, C. (2010). MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/.

Li, Y., Shen, W., Wu, T., Chen, L., Wu, D., Zhou, Y., & Xu, B. (2022). How higher order mutant testing performs for deep learning models: A fine-grained evaluation of test effectiveness and efficiency improved from second-order mutant-classification tuples. *Information and Software Technology*, *150*, Article 106954.

Li, Z., Ma, X., Xu, C., Cao, C., Xu, J., & Lü, J. (2019). Boosting operational DNN testing efficiency through conditioning. In *Proc. 27th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering (ESEC/FSE)* (pp. 499–509). ACM.

Ma, L., Juefei-Xu, F., Zhang, F., Sun, J., Xue, M., Li, B., Chen, C., Su, T., Li, L., Liu, Y., Zhao, J., & Deepgauge, Y. W. (2018). Multi-granularity testing criteria for deep learning systems. In *Proceedings of the 33rd ACM/IEEE international conference on automated software engineering* (pp. 120–131). ACM.

Ma, L., Zhang, F., Sun, J., Xue, M., Li, B., Juefei-Xu, F., Xie, C., Li, L., Liu, Y., Zhao, J., & Wang, Y. (2018). DeepMutation: Mutation testing of deep learning systems. In *29th international symposium on software reliability engineering (ISSRE)* (pp. 100–111). IEEE.

Ma, L., Zhang, F., Xue, M., Li, B., Liu, Y., Zhao, J., & Wang, Y. (2018). *Combinatorial testing for deep learning systems*. arXiv.

Ma, S., Liu, Y., Tao, G., Lee, W., & Zhang, X. (2019). NIC: Detecting adversarial samples with neural network invariant checking. In *26th network and distributed system security symposium (NDSS)*.

Murphy, C., Kaiser, G. E., & Arias, M. (2007). An approach to software testing of machine learning applications. In *19th int. conference on software engineering and knowledge engineering (SEKE)* (pp. 167–172).

Murphy, C., Kaiser, G., Hu, L., & Wu, L. (2008). Properties of machine learning applications for use in metamorphic testing. In *20th international conference on software engineering and knowledge engineering (SEKE)* (pp. 867–872).

Musa, J. D. (1996). Software reliability-engineered testing. *Computer*, *29*(11), 61–68.

Odena, A., & Goodfellow, I. (2019). TensorFuzz: Debugging neural networks with coverage-guided fuzzing. In *Proceedings of machine learning research: Vol. 97. Proc. 36th int. conference on machine learning* (pp. 4901–4911).

Pei, K., Cao, Y., Yang, J., & Jana, S. (2017). DeepXplore: Automated whitebox testing of deep learning systems. In *Proc. 26th symposium on operating systems principles (SOSP)* (pp. 1–18). ACM.

Pei, K., Cao, Y., Yang, J., & Jana, S. (2019). DeepXplore: Automated whitebox testing of deep learning systems. *Communications of the ACM*, *62*(11), 137–145.

Pietrantuono, R., & Russo, S. (2016). On adaptive sampling-based testing for software reliability assessment. In *27th international symposium on software reliability engineering* (pp. 1–11). IEEE.

Qin, Y., Wang, H., Xu, C., Ma, X., & Lu, J. (2018). SynEva: Evaluating ML programs by mirror program synthesis. In *International conference on software quality, reliability and security* (pp. 171–182). IEEE.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Recht, B., Roelofs, R., Schmidt, L., & Shankar, V. (2019). Do ImageNet classifiers generalize to ImageNet? In K. Chaudhuri, & R. Salakhutdinov (Eds.), *Proc. of machine learning research (PMLR), Vol. 97* (pp. 5389–5400).

Riccio, V., Jahangirova, G., Stocco, A., Humbatova, N., Weiss, M., & Tonella, P. (2020). Testing machine learning based systems: A systematic mapping. *Empirical Software Engineering*, *25*(6), 5193–5254.

Seo, S., Arik, S. Ö., Yoon, J., Zhang, X., Sohn, K., & Pfister, T. (2021). Controlling neural networks with rule representations. In *Advances in neural information processing systems: Vol. 34*.

Sharma, N., Jain, V., & Mishra, A. (2018). An analysis of convolutional neural networks for image classification. *Procedia Computer Science*, *132*, 377–384.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., Driessche, G., Graepel, T., & Hassabis, D. (2017). Mastering the game of go without human knowledge. *Nature*, *550*, 354.

Srisakaokul, S., Wu, Z., Astorga, A., Alebiosu, O., & Xie, T. (2018). Multiple-implementation testing of supervised learning software. In *AAAI workshops. association for the advancement of artificial intelligence*.

Stocco, A., Weiss, M., Calzana, M., & Tonella, P. (2020). Misbehaviour prediction for autonomous driving systems. In *Proc. of the 42nd int. conference on software engineering* (pp. 359–371). ACM.

Tian, Y., Pei, K., Jana, S., & Ray, B. (2018). DeepTest: Automated testing of deep-neural-network-driven autonomous cars. In *40th international conference on software engineering (ICSE)* (pp. 303–314). ACM.

Tsymbal, A. (2004). The problem of concept drift: Definitions and related work. *Computer Science Department, Trinity College Dublin*, *106*(2), 58.

Wang, H., Xu, J., Xu, C., Ma, X., & Lu, J. (2020). Dissector: Input validation for deep learning applications by crossing-layer dissection. In *IEEE/ACM 42nd international conference on software engineering (ICSE)* (pp. 727–738). ACM.

Weiss, M., & Tonella, P. (2021a). Fail-safe execution of deep learning based systems through uncertainty monitoring. In *2021 14th IEEE conference on software testing, verification and validation (ICST)* (pp. 24–35).

Weiss, M., & Tonella, P. (2021b). Uncertainty-wizard: Fast and user-friendly neural network uncertainty quantification. In *2021 14th IEEE conference on software testing, verification and validation (ICST)* (pp. 436–441).

Xiao, Y., Beschastnikh, I., Rosenblum, D. S., Sun, C., Elbaum, S. G., Lin, Y., & Song Dong, J. (2021). Self-checking deep neural networks in deployment. In *IEEE/ACM 43rd international conference on software engineering (ICSE)* (pp. 372–384). IEEE.

Xie, X., Ma, L., Juefei-Xu, F., Xue, M., Chen, H., Liu, Y., Zhao, J., Li, B., Yin, J., & See, S. (2019). DeepHunter: A coverage-guided fuzz testing framework for deep neural networks. In *Proc. 28th ACM SIGSOFT international symposium on software testing and analysis (ISSTA)* (pp. 146–157). ACM.

Zhang, J., Harman, M., Ma, L., & Liu, Y. (2022). Machine learning testing: Survey, landscapes and horizons. *IEEE Transactions on Software Engineering*, *48*, 1–36.

Zhang, M., Zhang, Y., Zhang, L., Liu, C., & Khurshid, S. z. (2018). DeepRoad: GAN-based metamorphic testing and input validation framework for autonomous driving systems. In *33rd ACM/IEEE int. conference on automated software engineering (ASE)* (pp. 132–142). ACM.

Zhao, C., Mu, Y., Chen, X., Zhao, J., Ju, X., & Wang, G. (2022). Can test input selection methods for deep neural network guarantee test diversity? A large-scale empirical study. *Information and Software Technology*, *150*, Article 106982.