UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II

UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

PH.D. THESIS IN

INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

OPERATIONAL ACCURACY ASSESSMENT OF CNN-BASED IMAGE CLASSIFIERS

ANTONIO GUERRIERO

TUTOR: PROF. STEFANO RUSSO

CO-TUTOR: PROF. ROBERTO PIETRANTUONO

XXXIV CICLO

SCUOLA POLITECNICA E DELLE SCIENZE DI BASE
DIPARTIMENTO DI INGEGNERIA ELETTRICA E DELLE TECNOLOGIE DELL'INFORMAZIONE

Candidate's declaration

I hereby declare that this thesis submitted to obtain the academic degree of Philosophiæ Doctor (Ph.D.) in Information Technology and Electrical Engineering is my own unaided work, that I have not used other than the sources indicated, and that all direct and indirect sources are acknowledged as references.

Parts of this dissertation have been published in international conference articles. In particular, the oracle problem in Machine Learning systems has been discussed in reference C4 of the author's publication list at the end of this thesis, and the DeepEST technique proposed in Chapter 4 has been published in reference C2.

Napoli, March 14, 2022

_____

Antonio Guerriero

This page intentionally left blank.

Abstract

Machine Learning (ML) systems are nowadays largely adopted in many application domains. In the field of Image Classification (IC), where Convolutional Neural Networks (CNN) represent the state of the art ML models, they can even outperform human beings. The performance of a CNN once deployed in the operational environment can be very different from the one estimated before release, due to unpredictable/unconsidered operating conditions.

The life cycle of ML systems in use in big companies like Google envisages a loop where the system is continuously monitored in operation, and gathered data are used to decide corrections/improvements to be applied in the next cycle. Since the amount of monitoring data in a cycle can be huge and the correct output for each operational input is unknown (manual labeling is required), the evaluation of the accuracy of the system in operation (*operational accuracy*) is costly and time consuming.

This dissertation deals with the problem of assessing the operational accuracy of CNN-based image classifiers. In line with the emerging life cycle for ML systems, the thesis targets the assessment problem from two perspectives: *online* assessment, directly in the operational environment, and *offline* assessment, in the development environment.

Online assessment is based on automated oracles, typically used for failure detection. Its advantage is to provide a continuous evaluation of the accuracy, as close as possible to the *actual* one, without requiring human intervention. The drawback is that online assessment is typically based on a probabilistic (not deterministic) knowledge of the correct label of an operational input.

Offline assessment typically involves human beings, who have to provide the correct label of the operational images, ultimately yielding more faithful accuracy estimates.

This thesis investigates experimentally the complementary characteristics of online and offline assessment techniques, and then proposes to combine them for providing continuous yet faithful estimates of the operational accuracy of CNN-based image classifiers, limiting the involvement of human beings to a level that may be considered affordable in many applications.

The thesis proposes and evaluates experimentally two online assessment techniques and one offline technique to evaluate the accuracy of CNN. The results of experiments show their respective strengths and limitations. In particular, the higher cost to perform the offline assessment compared to the online one is balanced by estimates closer to the actual accuracy.

Building on these experimental results, the thesis proposes a hybrid CNN Accuracy Assessment Cycle (CNN-AAC) – combining online and offline techniques - which can be integrated into iterative industrial-strength life cycle models for CNN-based systems.

Keywords: Machine Learning, Image Classification, Automatic Oracles, Sampling, Assessment.

Acknowledgements

I would like to express my sincere gratitude to my advisors Prof. Stefano Russo and Prof. Roberto Pietrantuono for the continuous support of my Ph.D. study and related research, for their patience, motivation, and immense knowledge. Their guidance helped me throughout the research and writing of this thesis. I could not have imagined having a better advisor and mentor for my Ph.D. study.

I would like to extend my sincere thanks to Prof. Michael R. Lyu for his invaluable supervision and support in particular during my period abroad at the Chinese University of Hong Kong.

I am deeply grateful to the thesis evaluators for the valuable feedback and the precise and detailed comments.

My sincere thanks also go to the whole Dependable and Secure Software Engineering and Real-Time systems (DESSERT) research group, for making the research laboratory a pleasant place to conduct my activities and share knowledge.

This page intentionally left blank.

Contents

This page intentionally left blank.

List of Acronyms

The following acronyms are used throughout this text.

AAC	Accuracy Assessment Cycle
ADI	Algorithm-dependent invariants
C	Combo
CES	Cross-Entropy Sampling
CNN	Convolutional Neural Network
CRO	Cross Referencing Oracle
CS	Confidence
DeepEST	Deep neural networks Enhanced Sampling Technique
DNN	Deep Neural Networks
DSA	Distance-based Surprise Adequacy
FN	False Negative
FP	False Positive

FPR	False Positive Rate
IC	Image Classification
ICOS	Image Classification Oracle Surrogate
IDI	Input-data-dependent invariants
LSA	Likelihood-based Surprise Adequacy
ML	Machine Learning
MSE	Mean Squared Error
NFP	Number of Failing Points
PAOC	Partitioning-based Automatic Oracle for CNN
RQ	Research Question
SC	SelfChecker
SRS	Simple Random Sampling
TDI	Training-data-dependent invariants
TN	True Negative
TP	True Positive
TPR	True Positive Rate
VFP	Variance of Failing Points
WBS	Weight-based Sampling

List of Figures

List of Tables

This page intentionally left blank.

Chapter 1

Introduction

Motivation Machine Learning (ML) systems are today integral part of many applications due to their ability reaching the same level or may even outperform human beings [30, 22, 62] 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" [56].

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 the training, the training process itself, and the verification process.

An ML model is trained with a set of data (*training dataset*) and is meant to operate in a given context (*operational context*). An arbitrarily large set of operational data can be collected (*operational dataset*), containing examples whose correct label is unknown. This data can be used to evaluate the accuracy provided by the model during the operation, but manual labeling is required. Due to unexpected phenomena occurring in the operational environment, such as *distribution shift* or *label shift* [16],

the accuracy estimated before the release of the ML systems can be very different compared to the one provided in operation.

Recht *et al.* showed how a broad range of ML models have important drops in the accuracy (up to 15%) when completely new data are submitted [55]. This issue depends on many factors, like incorrect training (overfitting), biases in the training process (such as the utilization of the test set both in the training phase, and to estimate the generalization error), utilization of unrepresentative datasets.

Life cycles (e.g. MLOps [1, 17]) specific for ML systems are envisaged by companies like Google, where development and operational stages are linked in a loop [3] aiming to assess and improve the accuracy of the ML system according to the operational environment. In particular, they aim to exploit operational data for remodeling and/or retraining of the ML system before the new deployment (*experimental stage*), and for both automatic evaluation of the accuracy and automatic retraining of the models on the field (*deployment stage*).

Problem Statement The operational data collected by monitoring the ML system are characterized by the absence of the true label. This represents an issue for the evaluation of the provided accuracy, and it is commonly known as *oracle problem* [44, 45, 18]. According to Murphy *et al.*, for ML models "there is no reliable test oracle to indicate what the correct output should be for arbitrary input" [44, 45]. Moreover, in continuous monitoring (such as MLOps), the size of the collected operational data is usually huge, making the manual labeling costly and time consuming.

In this dissertation, the problem of evaluating accuracy using operational data is addressed in the IC domain. In the rest of the thesis, operational data are considered to consist of images only, and Convolutional Neural Networks (CNN) are considered as ML systems, as they are the most popular and performing solution in this field [60].

In this domain, the oracle problem is usually addressed via automatic pseudo-oracles (for instance, based on ML models), which perform failure detection exploiting various sources of knowledge (e.g., training set, ML model, operational features). However, these classifications are only probabilistic. Automatic oracles are more sensitive to unexpected biases which can affect the classification causing the assessed operational accuracy to divergence.

An alternative approach consists of reducing the size of the operational dataset, sampling a few representative samples to compute the estimate. The evaluation of these samples can be performed by a human being, who can be considered deterministic but still characterized by a high cost.

Contribution The operational accuracy assessment is explored with reference to two different strategies: *online*, during the deployment stage, and *offline*, during the experimental stage.

The online assessment aims to continuously evaluate the accuracy provided by the CNN under assessment directly in the operational environment. It performs an on-field evaluation of the predictions for each new image automatically, namely via automatic oracles, avoiding human intervention. Each time the predicted accuracy does not satisfy the requirements, a remodeling & retraining step is triggered.

The offline assessment estimates the accuracy provided by the CNN in operation exploiting the human intervention. The most representative operational input is sampled, and human beings define the correct labels of the selected images. The operational accuracy is estimated by evaluating CNN predictions by comparison with the new correct labels.

The cost of the offline assessment is higher than the online one due to the involvement of human beings. On the other side, it opens to a new perspective for CNN improvement, building a training dataset more representative of the input submitted in operation by adding the new labeled

images. Indeed, the higher cost of the offline assessment with respect to the automatic one pays in terms of greater accuracy. While automatic oracles are less accurate, they can be very informative about the performance of the ML system. In particular, they give information about accuracy drops, without human intervention.

For online operational accuracy assessment, the thesis proposes two techniques – ICOS and PAOC - to implement an automatic oracle. ICOS aims to exploit different sources of knowledge to extract (manually and automatically) *invariants* to evaluate if the output of a CNN is failing or not. PAOC is a fully automated oracle exploiting clusterization and hierarchical classification to detect misclassifications of the CNN monitored.

For the offline operational accuracy assessment, the thesis proposes the DeepEST technique to spot as many misclassifications as possible, while providing an unbiased estimate of the operational accuracy. This is achieved leveraging an advanced rare population statistical sampling technique.

As a final contribution, the thesis proposes a hybrid CNN *Accuracy Assessment Cycle*, integrated into the life cycle of the CNN, combining online and offline assessment to exploit the advantages of both approaches while minimizing the assessment cost.

The rest of the thesis is structured as follows.

Chapter 2 sets background notions about ML fundamentals, ML systems life cycle, CNN for image classification, and introduces related work on ML systems operational accuracy assessment.

Chapter 3 presents solutions for the online assessment of CNN.

Chapter 4 presents solutions for the offline assessment.

Chapter 5 presents the Accuracy Assessment Cycle resulting by the combination of the two discussed approaches.

Chapter 6 reports the conclusions.

Chapter 2

Background and Related Work

This chapter presents the background and the related work. The background focuses on ML fundamentals, ML systems life cycle, and the state of the art model in the IC domain, namely the Convolutional Neural Networks. The related work targets the operational accuracy evaluation of ML systems, highlighting the differences among the state of the art approaches.

2.1 Machine Learning fundamentals

Machine learning solutions aim to build systems able to learn and improve themselves via examples, without human programming. Zhang *et al.* [75] list the following tasks ML models are usually applied:

- Classification: for each input, the model predicts a discrete value, called label (the class). An example is image classification, where the ML model predicts the label of the main subject of each image.
- Regression: the model predicts a continuous value for each input. An example is the steering angle prediction for autonomous vehicles.

- Clustering: the examples are partitioned into homogeneous regions.
- Dimension reduction: the complexity of examples used for ML model training is reduced; for instance, by filtering of non-informative features, or data representation.
- Control: depending on the input, the model performs actions aiming to maximize a reward.

Based on the selected task, different types of ML can be chosen. Based on the characteristics of the training data, they can be distinguished in:

- Supervised learning: the examples used for training contain the ground truth.
- Unsupervised learning: the ground truth is unavailable and the ML model aims to understand the data itself.
- Reinforcement learning: the data are represented as sequences of actions coupled with a reward.

This thesis focuses on Image Classification (IC) performed via supervised learning, which is the most widely used type of machine learning [27].

Independently of the specific task, six steps are needed to build an ML model: data gathering, data preparation, model choice, model training and parameter tuning, model evaluation, prediction.

The first two steps concern the collection and preparation of data (images), on which the ML model is trained. For instance, images are structured as a set of features, each one consisting of a matrix of pixels and a label. The objective is to build three datasets: a training set, a validation set, a verification (or test) set. The training set is a dataset of examples "representative" of the images expected in the execution environment. The

validation set is used to perform the parameter tuning of the ML model to improve the performance. The verification (or test) set is made by data previously unseen by the ML model to foresee the performance provided after the deployment in the execution environment.

The choice of the ML model depends on various factors: the specific task (image classification, natural language processing, and so on), the amount of data available, and the number of features. In the IC domain, Convolutional Neural Networks (CNN) represent the state-of-the-art model [66]. Nowadays, a plethora of very complex models, based on CNN, have been defined and used [63, 67, 23].

The training and parameter tuning step aim to teach and improve the performance of an ML model. A metric of the goodness of an ML model is the *accuracy*, namely *the ratio of the number of correctly classified examples to the total number of submitted ones. Correctly classified examples* are images with a prediction of the model that matches the correct label. On the opposite, a wrong classification is a prediction that does not match the correct label, and it is called *misprediction* or *misclassification* (in the classification domain).

The purpose of the model evaluation step is to compute the accuracy obtained with the images in the verification set, whose correct predictions are known. Since the data in the verification set are previously unseen by the ML model, this step provides a measure of the *generalization error*, indicating the error of an ML model predicting labels for previously unseen data. This step corresponds to the acceptance testing step of classical software systems, where the ML model is released only if the accuracy reached with the verification dataset meets the requirements (namely, it is beyond a certain threshold).

Concerning the prediction step, the ML model is assumed to run in its operational environment. In this environment, the ML model predicts labels for operational examples, namely arbitrary images whose labels are

unknown (*operational dataset*). For this reason, the evaluation of the accuracy of those data is tricky and very costly due to the absence of an *oracle*. In traditional software testing, an oracle is able to verify the output of the system under test against the values expected by the developer. An oracle in the ML domain is difficult to be defined since such systems "are designed to provide an answer to a question for which no previous answer exists" [75]. In current practices, a reliable oracle for arbitrary input in the ML domain corresponds to a human being, which manually labels a very large set of images to find the misclassifications.

In this thesis, the accuracy computed on the operational data is defined as *operational accuracy*, namely the accuracy computed considering the examples into the operational dataset.

2.2 ML systems life cycle

A detailed description of ML systems life cycle has been described by Ashmore *et al.* [3], which distribute the six steps described in the previous section in a spiral. A representation of this cycle is reported in Figure 2.1 where 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 according to the data contained in the training set;
- *Model Verification*: the trained model is evaluated on the verification set, and if the generalization error violates the requirements, the process returns to the Data Management stage;
- *Model Deployment*: models satisfying the requirements are integrated into the operational environment, with the monitoring of its

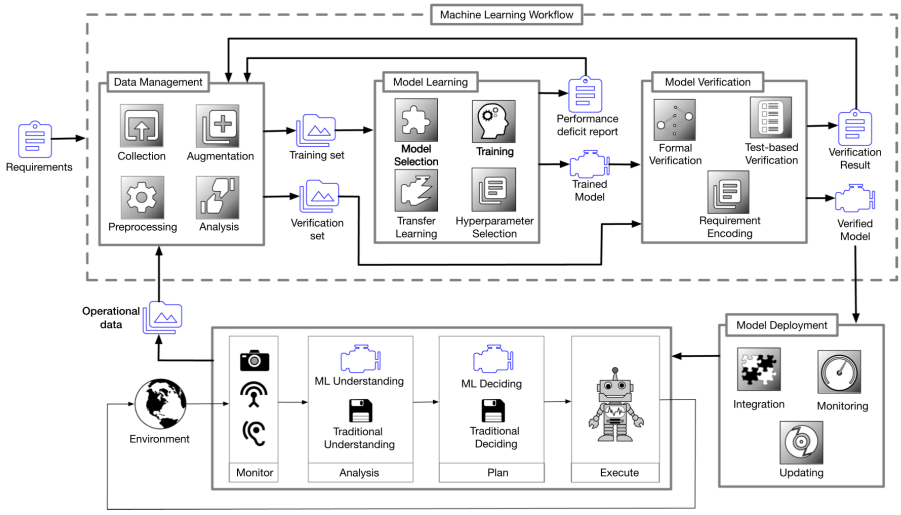


Figure 2.1. ML systems life cycle according to ref. [3]

operation, and with its updating thorough offline maintenance or online learning.

This spiral life cycle allows the collection of operational data to improve the model in successive steps. According to MLOps perspective [1], the first three stages can be seen as the *experimental stage*; the Model Deployment stage corresponds to the *deployment stage*.

A view representing both Ashmore and MLOps perspectives about the ML system life cycle is reported in Figure 2.2. 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 concerns all the actions needed 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

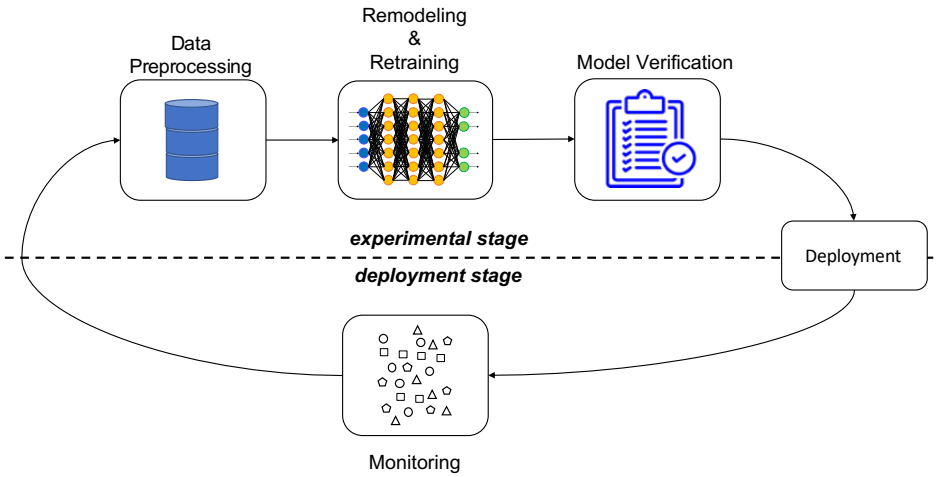


Figure 2.2. Generic ML systems life cycle

concerns the collection of operational data, peculiar environment characteristics, and the output of the ML system useful to evaluate the ML system accuracy in operation and to take correcting/improving actions in the next cycle. This life cycle is the one considered in the rest of the thesis.

MLOps principles [17] 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 called *oracle problem*. For ML systems, included CNN for IC, "there is no reliable test oracle to indicate what the correct output should be for arbitrary input" [44]. The automation of the operational accuracy assessment processes is limited by the absence of the correct label for each operational input.

2.3 Convolutional Neural Networks for image classification

The first practical model of a CNN has been introduced by LeCun, which developed the LeNet-5 [31].

As shown in Figure 2.3, a CNN is typically composed of: one or more convolutional and pooling blocks, one or more fully-connected layers, and an output layer.

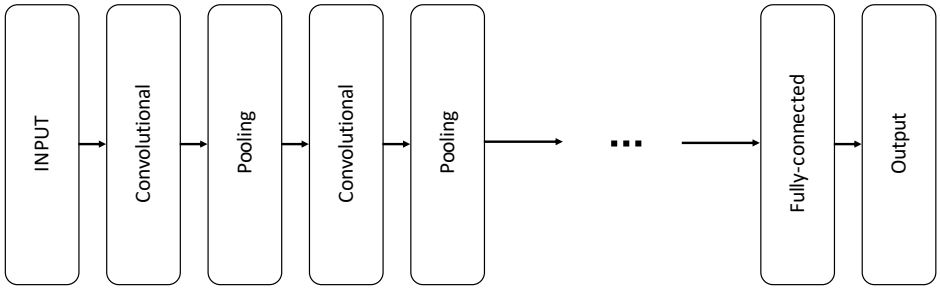


Figure 2.3. Generic CNN model

Convolutional Layers

The convolutional layer is the core of a CNN. It represents the layer aiming to extract features from the image in input via *convolution*, a mathematical operation that applies a filter (Kernel) to the input image to extract features. Features extraction is performed by applying the filter through all the points of the image and convolving them into a single output. Different filters can perform different operations, such as edge detection, blur, and sharpening.

Pooling

Pooling layers aim to reduce the number of parameters for large images (sub-sampling). They take as input the output of the convolutional layer producing a single value. Many different pooling techniques can be used: max pooling (taking the maximum value), average pooling (taking the average value), and so on. The purpose of these layers is to reduce the number of parameters to make the network invariant to translations in shape, size, and scale of the images.

Fully-connected layers

Fully-connected layers are composed of a set of interconnected neurons. Each neuron applies a linear transformation to the input vector through a weights matrix producing the output via an activation function. Fully-connected layers are those layers where all the inputs from one layer are connected to every activation unit of the successive layer. These layers take as input all the features values extracted by Convolutional and Pooling layers to compute the final output.

2.4 Operational accuracy assessment of CNN-based image classifiers

A significant research effort has been put on quality evaluation of ML systems [56], but just few of them concern the assessment of the accuracy provided in the operational environment. In fact, a primary goal is to find adversarial examples causing *mispredictions*, namely to expose as many failing behaviours as possible [48, 39, 76, 41, 47]. Specificly for Deep Neural Networks, included Convolutional Neural Networks for Image Classification, several structural coverage criteria have been proposed to drive the automated generation of test inputs and assess the test adequacy

– neuron coverage [48], k-multisection neuron coverage, neuron boundary coverage [39], combinatorial coverage [41]. It has been argued that these criteria may be misleading, because of the low correlation between the number of misclassified inputs in a test set and its coverage [33]. Wu *et al.* [72] and Kim *et al.* [28] recently considered discrepancy measures between the training/validation data and test data, to improve fault detection and to have coverage criteria better correlated to failure-inducing inputs.

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 [15]. Beside differences in testing ML models and conventional software (e.g., the oracle definition), which make ML testing problematic, a further issue is that the so-obtained testing results are not necessarily related to the accuracy actually experienced in operation. In fact, testing data may be not representative of the actual operational context. This may happen when test data are generated artificially (like in adversarial examples generation) or they differ significantly from input observed in the field. The number of mispredictions and/or the coverage achieved give only an "indirect" (and, for what said, inaccurate) measure of the expected accuracy in operation, and ultimately of the confidence that can be placed in these kind of systems.

To compute a "direct" estimate of the accuracy provided in operation by the ML model under test, two main strategies can be adopted:

- exploiting ML algorithms and statistical techniques to automatically detect failures in operation;

- sampling a subset of operational input according to a certain belief (imitating the sample distribution in the dataset [34], or preferring the selection of failing samples [19]) and manually labeling the samples.

The first solution is preferable when the accuracy of the ML system must be evaluated online, during operation as in the *deployment stage* in MLOps. A sampling strategy is preferable when new samples must be manually labeled, for instance, to retrain the ML system (like in the *experimental stage* in MLOps) to spot the most interesting examples (e.g. the ones causing misclassification). Due to the manual labeling, the cost of the second solution is higher, and it would be desirable to run it only when necessary (e.g. the operational accuracy significantly drops).

2.4.1 Automated oracles

The oracle problem in ML testing is one of the main challenges tackled by researchers Zhang *et al.* [75]. 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) [64]. 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 running the test input of multiple implementations of the same algorithm (based on a predefined percentage threshold). 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.

On the same line, Pei *et al.* leverage multiple Deep Learning systems as cross-referencing oracles in their DeepXplore framework for white-box testing [48]. In particular, they compute a neuron coverage for measuring the parts of the system under test exercised by test inputs and consider multiple Deep Learning systems with similar functionality as cross-referencing oracles to avoid manual checking. The aim is to find relations between neurons activations and failing behaviors.

Wang *et al.* [69] 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 between 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 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.* [11] propose a criterion for failure prediction of CNN based on True Class Probability criterion. This criterion is learned by a confidence neural network (ConfidNet) built upon a classification model. True Class Probability is shown to be effective in performing failure prediction on classification and segmentation problems.

Great interest is also in misbehavior prediction of Deep Neural Networks for autonomous driving problems through automatic oracles [26]. Stocco *et al.* propose SelfOracles to detect unsupported driving scenarios based on Deep Neural Networks behavior at run time [65]. 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 of the sample considered.

Xiao *et al.* [73] recently proposed SelfChecker (SC) for both failure detection of CNN 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 [69], ConfidNet[11], and SelfOracle[65]).

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 the state-of-the-art techniques in failure prediction.

The discussed techniques do not account for the potential deviation of the operational context from pre-deployment one. For this reason, the cited techniques are expected to have bad performance in presence of unexpected phenomena like label shift [16].

Other techniques are proposed that only partially address the oracle problem, namely mutation testing and metamorphic testing. Mutation testing is proposed by Ma *et al.* to evaluate test data quality [40]. They define a set of source-level mutation operators to inject faults into the sources of a DL model, like training data and programs, and model-level mutation operators to inject faults directly into models. They consider mutation to generate new samples starting from an input, whereby the expected output is already known. However, as the mutation approach requires knowledge of the label, it does not allow to submit test cases whose expected output is unknown.

Metamorphic testing exploits many relations (called metamorphic relations) between changes of both input and output over different executions. Every time a relation is violated, a failure is detected. For instance, Tian *et al.* [68] consider metamorphic relations to generate tests and evaluate

the output of DNNs in the autonomous vehicles domain. Xie *et al.* [74] propose a metamorphic mutation strategy to generate new semantically preserved tests, leveraging multiple coverage criteria to guide the test generation.

In the application of metamorphic testing to ML systems, and specifically to IC systems [13], the metamorphic relations are used to generate new images starting from those with a known label (e.g., by manipulating the order or the values of the features) preserving the semantic of the image (the label remains unchanged), or with a label certainly different from the original one. Hence, it is often combined with mutation testing to automate the generation of test cases. However, since this way of applying metamorphic testing assumes that tests are generated from images whose label is already known, it is not applicable to test arbitrary images whose label is not known.

2.4.2 Sampling techniques

In traditional software systems, probabilistic sampling is used in the context of operational testing to estimate the expected reliability of a system after deployment. In operational testing, test suites are built by selecting or generating tests according to the expected operational profile, which is a probabilistic characterization of the expected usage. It was the core technique of Cleanroom software engineering [43, 12, 10, 35], as a means to certify the software against a given mean time to failure (MTTF), and then of the Software Reliability Engineering Test process proposed by Musa at AT&T [46]. With the years, researchers looked for more efficient sampling strategies to improve the accuracy of the estimate at lower cost. Cai et *al.* developed *Adaptive Testing*, still based on the operational profile, but foreseeing adaptation in the assignment of test cases to input partitions [38, 37, 6, 7]. Recently, Pietrantuono et *al.* stressed the use of (unequal probability) sampling to improve the efficiency of the estimates [50], for-

malizing several sampling schemes to this aim [49]. Pietrantuono *et al.* also apply sampling algorithms to select tests for the operational reliability assessment of Microservice Architectures [51, 52]. In particular, they propose a sampling algorithm for rare populations to select as many failing tests as possible. Failing tests are considered the most informative ones and enable a cost-effective estimate of reliability. The dis-proportional selection is then balanced by the estimator to preserve the estimator's unbiasedness.

In line with operational reliability assessment, recently researchers have proposed sampling-based strategies for operational accuracy assessment of CNN. Li et *al.* [34] presented CES (Cross-Entropy Sampling) as the state of the art approach using operational testing to this aim. CES samples images based on the output of the m neurons in the last hidden layer. This information is assumed to be more robust to the operation context drift and highly correlated with the prediction accuracy since it is derived from the linear combination of this layer's output. As the classical operational testing, CES aims to select a small data sample that accurately represents the population. A representative sample would roughly contain the same proportion of examples causing misprediction as the operational dataset.

However, the mere imitation of the expected input can be inefficient, especially with very accurate CNN, because a great effort is made to manually label correctly classified images to get an acceptable estimate of the operational accuracy. Considering the cost of labeling, it is evident that maximizing the sampling of images related to misclassifications, while still getting an unbiased estimate of the operational accuracy, is preferable.

2.5 Discussion

Looking at the ML literature, testing of ML models is mostly seen as a pre-release activity, e.g.: testing for evaluation of accuracy on a verification dataset (*acceptance testing*), testing against adversarial examples

(*robustness testing*).

This thesis focuses on the evaluation of the accuracy by monitoring the ML model after the release in its operational environment. The data required to evaluate the operational accuracy can be monitored during operation. The work refers explicitly to *assessment* rather than testing because it is not required to resubmit operational data to the ML model. In particular, the automatic pseudo-oracles take in input the output of the ML model and compute *online* an estimate of the operational accuracy without human intervention. The sampling strategies need to sample the most significant examples to allow a human oracle to compute *offline* estimates of the operational accuracy on a reduced set of examples.

Although the automatic oracles do not need human intervention, reducing the cost of applying, they are characterized by the presence of a high number of false positives [75]. This issue depends on the probabilistic nature of the knowledge used to evaluate the output of the ML model under assessment. On the opposite, the offline techniques rely on "deterministic" feedback of the human oracle, which is costly and time-consuming, but it guarantees the absence of false positives, providing more faithful estimates.

A way to reduce the costs of applying and maximize the benefits is to combine the online and offline assessment in an Accuracy Assessment Cycle, as shown in Figure 2.4. The idea is to have at each cycle a "low-cost" estimate of the accuracy provided by the *online assessment*, and to trigger a "high-cost" (but more faithful) estimate of the *offline assessment* only when required (e.g. the operational accuracy provided by the automatic pseudo-oracle drops under a certain threshold).

The following two Chapters describe the techniques proposed for both the online and offline assessment that aim to overcome the limitations mentioned about the related work. In particular, the benefits and the costs of the proposed techniques are discussed, highlighting the pros and cons of each approach. In the last Chapter, the hybrid Accuracy Assessment

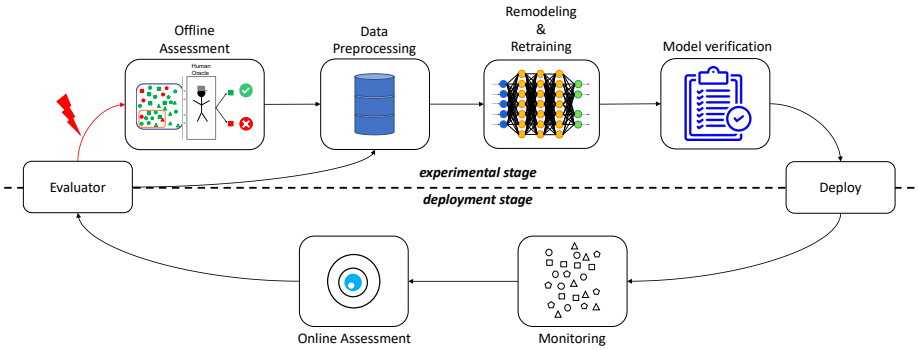


Figure 2.4. Proposed accuracy assessment cycle

Cycle shown in Figure 2.4 is described and simulated.

Chapter 3

Online operational accuracy assessment

This chapter presents ICOS and PAOC as oracles for online CNN operational accuracy assessment. The idea is that the input space can be partitioned to reduce the complexity of the classification problem, simplifying the detection of failures of the CNN under assessment. ICOS exploits "manual" partitioning based on the characteristics of the operational environment. Instead, PAOC is based on automatic partitioning, clustering the most similar classes based on the examples in the training set. Although manual partitioning is more effective than automatic, it is applicable only when specific features of the operational environment are available. These features represent peculiar characteristics of the operational profile, like the way images are generated, and they are unavailable for training images.

The objective is to define techniques able to assess the quality of a CNN in the operational environment, to support paradigms, like MLOps, aiming to know how a system is performing, avoiding the manual labeling

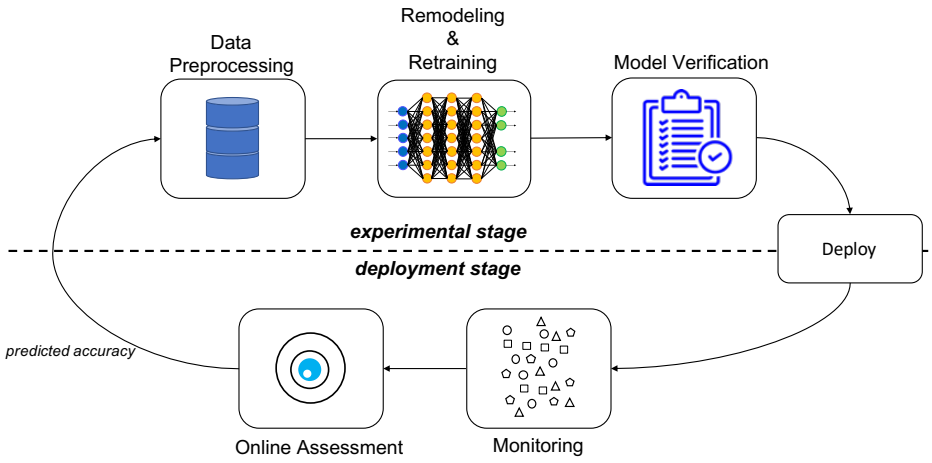


Figure 3.1. ML systems life cycle with online assessment

of the operational input.

The online assessment performed via automatic oracle can be integrated into the life cycle presented in the previous chapter (Figure 2.2). In particular, the *Online Assessment* phase can be placed in the *deployment stage*: the automated oracle computes the predicted accuracy on the data coming from monitoring of the CNN during the 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. A graphical representation is reported in Figure 3.1.

3.1 ICOS: *Image Classification Oracle Surrogate*

3.1.1 Overview

The *Image Classification Oracle Surrogate* addresses the oracle problem in the IC domain inferring a set of invariants from input data, training

data, and the specific ML algorithm adopted, representing assertions that a correct response by the CNN should never violate. As reported in Figure 3.2, when submitted to ICOS, both the image and the predicted output of the CNN are checked against the invariants: if at least one invariant is violated, the output is *fail*, otherwise *pass*. When a failure is detected by one type of invariant, the remaining ones are not further considered.

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

$$H \text{ :- } C_1 \wedge \cdots \wedge C_n, B_1 \wedge \cdots \wedge B_m \tag{3.1}$$

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 form suits the three types of invariants considered. The following sections detail the invariants.

As an example, the following clause is considered:

$$\begin{aligned}(outcome = Fail) \text{ :- } (pixel_1 > 25) \wedge \cdots \wedge (pixel_n < 250),\\ (predicted\_label \neq 3) \wedge (predicted\_label \neq 6)\end{aligned} \tag{3.2}$$

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

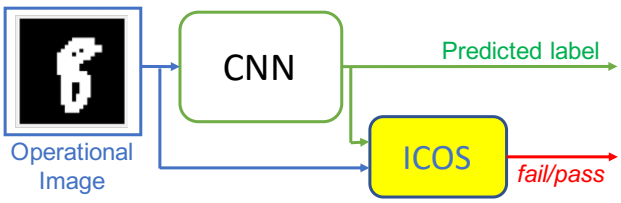


Figure 3.2. ICOS workflow

250 are constraints.

3.1.2 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) which are unavailable for training images.

Let us consider a system receiving input from two different cameras: the first one pointed in a garden, and the second pointed in the sky. The probability the first camera could photograph an airplane, or the second one could photograph a dog is actually zero.

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 can be described regarding the classification of handwritten digits, a very common task in the image classification research [31]. In particular, a system with two input forms is considered: 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. Consequently, the operational input can be 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 which 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 can be updated as follows:

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

These characteristics represent the upfront knowledge related to the system and its domain and do not depend on the training data and/or the CNN architecture. This condition implies greater robustness against phenomena like conceptual drift and distribution shift in general.

This invariants are defined as *input-data-dependent invariants* (IDIs). 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.

Google researchers studied the advantages of user-defined rules in Deep Neural Networks [58] incorporating a rule encoder directly into the models. They envisage a general improvement, also in terms of domain adaptation using the rule strength, becoming robust to phenomena like distribution shift.

3.1.3 Training-data-dependent invariants

The second source of knowledge considered is the training data. A CNN is trained on a *training set* containing information about the expected behavior in operation. ICOS automatically infers knowledge from

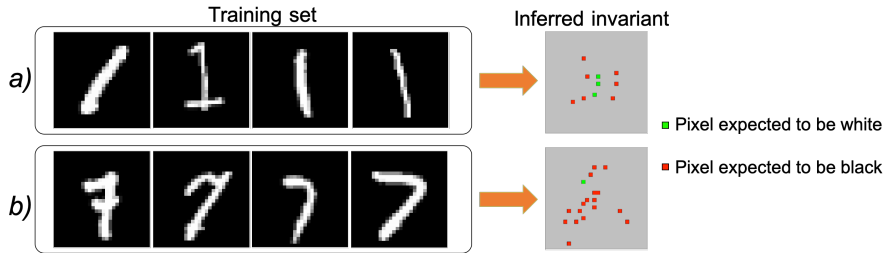


Figure 3.3. Two training-data-dependent invariants for MNIST

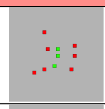
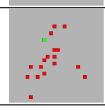
the training set through *explainable* ML algorithms, such as *decision rules* or *decision trees*. It relates incorrect outputs to the observed inputs, encoding the inferred relation as an ordered list of invariants.

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, Figure 3.3 shows two invariants extracted form the training set of the MNIST dataset. Training data in Figure 3.3 *a)*, labelled 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 Figure 3.3 *b)*. Table 3.1 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 sig-

Table 3.1. Two misclassifications detected by the TDIs of Figure 3.3

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

nificantly 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.* [53], generated from training data, and used as pseudo oracles for ML programs testing.

The efficacy of TDIs mainly depends on the *representativeness* of the training set with respect to operational input. The error probability of a classifier function $h : X \to Y$ (that predicts the label $y \in Y$ given the input object $x \in X$), is the probability that it does not predict the correct label on a given input [57]. This is strictly dependent on the quality of the training set.

In principle, if T is an ideal perfect training set, T^* is *representative* if $Acc(T^*) \approx Acc(T)$, where $Acc(X)$ denotes the classification accuracy obtained using X as training set [5]. Our ability to exploit the training set to detect failures depends on how close $Acc(T^*)$ is to $\approx Acc(T)$.

TDIs can work well when the training set is sufficiently representative, otherwise, they are likely to lead to many *false positives*.

3.1.4 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.* [42] 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 as a relevant feature since 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 [14, 70, 71]. ICOS extracts the patterns observed in the nominal case from the probability values (for instance, it derives that values are never very close to each other when the output is correct), 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 *algorithm-dependent invariants*, called *algorithm-related* failures. F_{ADI} may 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 3.2 shows six examples from the MNIST dataset, along with the CNN predictions and with the response of ICOS based on the type of violated invariant. 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).

Table 3.2. Examples of ICOS output for MNIST

Image	**Label**	**CNN output**	**ICOS**	**Confusion matrix element**	**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

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. The latter two types of invariants are automatically inferred, and represent the actual added-value of ICOS to increment completeness over the manually encoded ones. 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 a confidence depending on how many violations are raised, is left to future investigation).

3.2 PAOC: *Partitioning-based Automatic Oracle for CNN*

3.2.1 Overview

PAOC is an oracle surrogate aiming to evaluate the output of an Image classifier when (unlabeled) operational data are submitted. As ICOS, the evaluation of the output of a CNN relies on a certain belief, which in this case is extracted only from the dataset used for the training. In particular, PAOC's belief relies on how images can be partitioned based on their common characteristics.

The objective is to evaluate the *operational accuracy* of a CNN using operation data without human intervention.

3.2.2 Strategy

As for ICOS, the idea consists of extracting *likely invariants* from available sources of information, to be used to evaluate the CNN output. The underlying assumption is that the input space can be partitioned based on common characteristics among images of the dataset.

Figure 3.4 shows a demonstrative example of a dataset of four images representing a dolphin, a dog, an airplane, and a country house. As shown for ICOS, a human being can reasonably partition the images based on their semantics: a partition (P1) containing animals (the dog and the dolphin), and a partition (P2) containing inanimate objects (the house and the plane).

In the case of automatic image classification, the partitioning can be defined based on specific values of pixels, like the dominant color, or brightness and contrast. An alternative partitioning is shown in Figure 3.5, where images are partitioned based on the dominant color in the background: the dolphin and the airplane have blue as the dominant color, the dog and the

Figure 3.4. Partitioning Example (1)

country house have green as the dominant color.

Figure 3.5. Partitioning Example (2)

In summary, the images contained in the training dataset can be partitioned based on common characteristics. The partitioning will group the most similar classes, adding a new label to each image into the training set indicating which partition they belong to. So, PAOC can determine if the label predicted by a CNN for a certain image is correct or not based on the partition it should belong to. Moreover, new classifiers can be trained

on the images of a specific partition, focusing on a reduced set of labels.

3.2.3 Partitioning

Partitioning aims to enrich the training set with new labels representing for each image the corresponding partition. This allows training new classifiers to determine the partition of each operational image. *Clustering* is the most common technique to group the samples in a set; it is widely used for unsupervised classification of images, usually exploiting k Nearest Neighbours (kNN) algorithm [9, 20].

Figure 3.6 shows the partitioning process, consisting of 3 main steps: *clustering*, *partitioning*, and *labeling*.

Clustering aims to find k clusters of samples in the CNN training dataset, without considering their class. k has to be considered lower than the number of labels l; $k \leq l/2$ is suggested. Since it is an unsupervised task, different clusters may contain images belonging to the same class.

As the final objective of the process is to generate k partitions with distinct sets of labels (namely, the intersection of two partitions has to be empty), in the *partitioning* step, partitions are built iteratively according to Algorithm 1. For each *cluster* the number of *occurrences* of the *labels* is computed; as long as there are labels left unassigned to any *partition*, the *label* is assigned to the *partition* corresponding to the *cluster* with the highest number of *occurrences* for that label.

During the *labeling* step, the label encoding the corresponding partition is attached to each image into the training dataset. The output of the partitioning process is a new dataset, called *Partitioned dataset*, which

Figure 3.6. Partitioning process

Algorithm 1 Labels partitioning

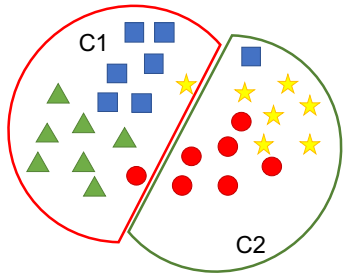
k: number of clusters; C_i: i^{th} cluster; $compute\_occurences(C_i)$: returns a data frame with three vectors representing the *cluster*, the *label*, and the *number of occurrences* for that label in the cluster, while the number of rows depends on the number of different labels in the clusters; P_j: j^{th} partition

1: **for** i in $(0, k)$ **do**
2: $DF.append(compute\_occurrences(C_i))$
3: **end for**
4: **while** $!M.empty()$ **do**
5: $index = find\_max(M[occurrences])$
6: $j = DF[cluster][index]$
7: $label\_t = DF[label][index]$
8: $P_j.add(label\_t)$
9: $DF.remove(DF[label] == label\_t)$
10: **end while**

contains two labels for each image, concerning respectively the class and the corresponding partition.

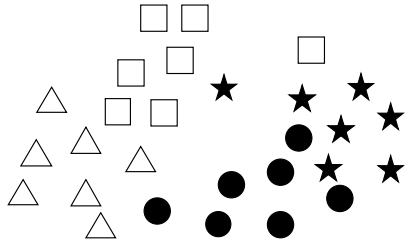
An example of the partitioning process is reported in Figure 3.7, where a dataset with four labels is partitioned into two partitions (P1 and P2).

Figure 3.7 (a) shows the result of clustering. As shown in the table, each cluster has a certain number of occurrences for each class. The partitioning step is run as described in Algorithm 1, and the results are reported in Figure 3.7 (b), where the new labels (which represent a sort of superclass grouping the selected set of original labels) are shown (*white* and *black*).



Cluster	Label	Occurrences
1	▲	7
1	■	6
2	●	6
2	★	6
1	●	1
1	★	1
2	■	1

(a) Clustering



Color	Partition
White	P1
Black	P2

(b) Partitioning and labeling

Figure 3.7. A contrived example for the Partitioning Algorithm

3.2.4 Architecture

PAOC is inspired by the classification via hierarchical ensemble approaches [61], commonly used for Multi-class problems. In particular, the following hierarchy of classifiers is considered:

- A *Partition Assigner*: trained on the Partitioned dataset, it is able to assign the corresponding partition to the operational images.
- A set of *Partition Explorer*: defined for each partition, they are specialized in the classification of the samples considering only the labels belonging to its partition.

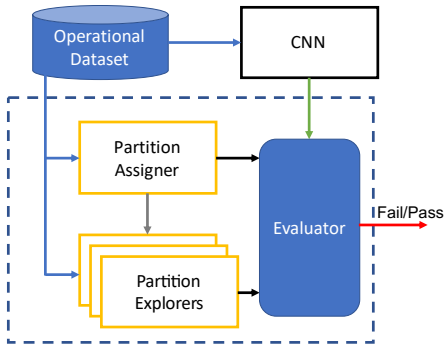


Figure 3.8. PAOC workflow

Figure 3.8 depicts a graphical representation of the workflow of PAOC. The right Partition Explorer is triggered by the output of the Partition Assigner. The decision process executed by the Evaluator for each image in the operational dataset is explained by Algorithm 2.

Algorithm 2 Operational images evaluation process

i: current image from the operational dataset; class: class predicted by the CNN; p: partition provided by the Partition Assigner; c: class provided by the Partition Explorer; outcome: PAOC output

```
1: class = CNN(i);
2: p = PartitionAssigner(i);
3: if class ∉ p then
4:     outcome = FAIL;
5: else
6:     c = PartitionExplorer_p(i);
7:     if class ≠ c then
8:         outcome = FAIL;
9:     else
10:        outcome = PASS;
11:    end if
12: end if
```

PAOC differs from traditional hierarchical classifiers on how the classification is made. In fact, to detect a misclassification, the "hierarchy" can

be executed only partially since the *Partition Explorers* are triggered only if the *Partition Assigner* does not find a failure.

3.3 Experimentation

This section reports an experimental evaluation of the proposed techniques. A comparison with two baselines is provided: Cross Referencing Oracle (CRO), implementing Multiple Implementation Testing [64], and SelfChecker (SC), as the most performing oracle surrogate in the current literature [73].

3.3.1 Research Questions

The experimental evaluation targets the following research questions (RQs):

- **RQ1** (effectiveness): *How effective are ICOS and PAOC at measure the operational accuracy compared to CRO and SC?*
- **RQ2** (invariants selection): *How does invariants selection influence the ICOS performance?*
- **RQ3** (stability): *How do ICOS, PAOC, and SC behave in presence of label shift?*

The three RQs aim to show the effectiveness of the proposed techniques estimating the accuracy achieved by the CNN monitored in operation (RQ1), considering how ICOS'performance can change due to invariants tuning (RQ2), and how much robust they are to unexpected conditions in operation (RQ3).

3.3.2 Evaluation Metrics

To answer the Research Questions, the following metrics are considered:

- **Mean Squared Error** (MSE): $MSE(\hat{\phi}) = \frac{1}{30}\sum_{i=1}^{30}(\hat{\phi}_i - \phi_i)^2$, ϕ_i is the actual operational accuracy computed on the operational dataset sampled during the i^{th} repetition, and $\hat{\phi}_i$ is the accuracy computed through oracle predictions (*predicted accuracy*) on the same operational dataset.

- **Offset**: $offset_i = |\hat{\phi}_i - \phi|$, where ϕ_i is the actual operational accuracy computed on the operational dataset sampled during the i^{th} repetition, and $\hat{\phi}_i$ is the predicted accuracy on the same operational dataset.

- **True Positives Rate** (TPR): $TPR = TP/(TP+FN)$ refers to the proportion of CNN predictions lebeled as failing by the oracle out of those who actually have failed;

- **False Positive Rate** (FPR): $FPR = FP/(TN+FP)$ refers to the proportion of false alarms over the total number of actual correct classifications;

- **F1-score** (F1): $F1 = (2\times TP)/((2\times TP)+FN+FP)$ refers to the harmonic mean of the precision and recall.

The MSE aims to show how the oracles perform when automatically assessing the operational accuracy of a CNN. The use of the TPR aims to show the number of failures correctly classified by the automatic oracles. The FPR well shows the proportion of False Positives, which represents useful information choosing an oracle surrogate.

The goal is to achieve a high TPR and F1, and a low MSE and FDR.

Table 3.3. List of datasets

Dataset	Number of Images	Number of Classes
MNIST	60,000	10
CIFAR10	60,000	10
CIFAR100	60,000	100

3.3.3 Experimental subjects

The evaluation is performed on nine different Convolutional Neural Networks (CNN). In particular, three CNN models are considered for each of the following three datasets: MNIST [32] (A, B, C), CIFAR10 (D, E, F) and CIFAR100 (G, H, I) [29].

Table 3.3 reports the characteristics of the three considered datasets. Table 3.4 reports the number of layers and parameters for each CNN.

Table 3.4. List of experimental subjects

CNN	Dataset	# of Layers	# of Parameters
A	MNIST	7	6,237
B		6	97,114
C		8	545,546
D	CIFAR10	13	1,084,234
E		10	258,762
F		12	550,570
G	CIFAR100	16	15,047,588
H		9	564,484
I		13	1,465,220

Usually, strategies like *transfer learning* are used to achieve high accuracy levels [59]. Recht *et al.* showed that by exploiting previously unseen test sets to evaluate the accuracy of the CNN, the claimed accuracy (obtained on usual test sets) drops range from 3% to 15% on CIFAR10 and

from 11% to 14% on ImageNet [55]. Since the datasets considered are public, and there is no guarantee that the online available models are not trained on the full dataset, to avoid biases, each CNN is trained "from scratch" by separating training, verification, and test sets. For the verification set, it is meant the set of samples used to evaluate the CNN and to compute the generalization error. For the test set, it is meant the unlabeled images used to sample the operational dataset in each experiment.

The training set size is set to 40% of the available dataset size for MNIST and CIFAR10 (28,000 and 24,000 respectively). For CIFAR100 a training set of 40,000 images is considered to have at least 400 examples for each class. The verification set size is 2,500 images for MNIST and CIFAR10; for CIFAR100 it is 4,000. The test set size is set to 20,000 images for MNIST and CIFAR10; for CIFAR100 it is 3,000. Operational dataset images are randomly selected (without replacement) from the test set (in the number of 39,500 for MNIST, 33,500 for CIFAR10, and 15,000 for CIFAR100) remaining after training and verification set construction. As each experiment is repeated 30 times – for statistical significance - the random selection procedure is performed before each repetition. The labels of the selected images are removed to emulate the case of unknown expected output.

3.3.4 ICOS implementation

Invariants 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 3.1.2), where each source produces output belonging to a certain subset of labels.

In this way, the operational dataset can be partitioned in subsets assumed to be disjoint (from a class perspective). For the evaluation, the

following three partitions are considered 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:

- set of all animals (*dog, frog, horse, bird, cat, deer*);
- set of all non-animals (*airplane, car, ship, truck*).

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

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

These partitioning criteria are defined as *fine partitioning* (*fp*). It means that for images selected from the partition with all digits with curves, the output must be (0, 3, 6, 8, 9); otherwise, a violation occurs.

To show the impact of partitioning (a different partitioning can provide different results), a second criterion, which just split the test set into two partitions, is considered since it can be applied to all the datasets:

- the first partition P_1 is made by:
  - $(0, 1, 2, 3, 4)$ for MNIST
  - $(airplane, automobile, bird, cat, deer)$ for CIFAR10
  - the first 50 classes $(apple, ..., mountain)$ for CIFAR100
- the second partition is made by P_2.
  - $(5, 6, 7, 8, 9)$ for MNIST
  - $(dog, frog, horse, ship, truck)$ for CIFAR10
  - the last 50 classes $(mouse, ..., worm)$ for CIFAR100

This partitioning criterion is defined as *equal partitioning* (*ep*).

Indeed, *fp* and *ep* criteria are selected just to show the role played by IDIs in ICOS. Different partitioning can provide different results.

In the rest of the thesis, ICOS_{fp} refers to ICOS implementing *fine partitioning*, while ICOS_{ep} implements *equal partitioning*.

Training-data-dependent invariants The training-data-dependent rules are derived applying the **C4.5** algorithm [54], an implementation of decision trees known for its ability to generate classifiers with easy interpretation by human beings [2]; the setting is in Table 3.5. The *Gini index* is used as the quality measure to compute the splits to derive the rules.

Table 3.5. C4.5 parameters 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 obtained represents conditions satisfied by samples in the training set. Each time a condition is violated a failure is detected. As an instance, the invariant related to Figure 3.3.a is:

$$fail \text{ :- } \$514\$ <= 253 \wedge \$406\$ > 157 \wedge \$539\$ <= 111 \wedge$$
$$\$411\$ <= 61 \wedge \$347\$ <= 2 \wedge \$206\$ <= 0 \wedge \$327\$ <= 7$$
$$\wedge \$522\$ <= 0 \wedge \$489\$ > 56 \wedge \$350\$ > 165, output \neq 1$$

This invariant states that when the values of pixels (shown between "$" symbols) are below or above a certain thresholds, if the output of the CNN is different from 1 a failure occurred. The pixels of each image are sorted from left to right and from top to bottom.

The exact list depends on the values of *confidence* and *support*. For RQ1, the minimum confidence value is set at $C = 0.99$ for all the models and datasets and, since the discriminative power of the rules depends on the support on which such a confidence is obtained, low-support rules have been filtered out with these criteria: for MNIST, it is considered the minimum support, under $C \geq 0.99$, for which at least the 50% of submitted images are evaluated; for CIFAR10 and CIFAR100, whose rules have a considerable lower average support, the median support over the rules is considered with $C \geq 0.99$. The resulting values are: $S = 140$ for MNIST, $S = 14$ for CIFAR10, and $S = 8$ for CIFAR100. [1]

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, and $S = 8$ for CIFAR100;

[1] These parameters can be fine-tuned for improving performance on a case-by-case basis. In the experiments, the default configuration is used. In RQ2, further different configurations are considered.

Table 3.6. Random forest parameters configuration

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

- *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.

- *Criterion 3*: Select all the invariants, regardless their confidence ($C = 0$) and support ($S = 0$).

The influence of the possible orderings of TDI has been explored with different datasets and models. The order of the invariants impacts the results when considering the whole set of TDI without filtering with confidence and support constraints. Since the order does not affect significantly on average (for some CNN the TDIs with low support show better effectiveness finding mispredictions, for other CNN invariants with higher support are better) the results after the filtering, the random order of TDI is considered to avoid additional biases in the results.

Algorithm-dependent invariants ADIs extraction uses the KNIME implementation of Random Forest [4], with default parameter values (as reported in Table 3.6), and considering the SUT verification set as training set. In particular, the output of the last layer of each CNN is collected for each element of the verification set. This new data are used as 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.

3.3.5 PAOC implementation

In the following subsection, a description of how the Partitioned dataset is built is reported, and *Partition Assigner* and *Partition Explorer* implementations are described.

Partitioned dataset

The Partitioned dataset is defined by applying clustering to the images inside the training dataset. In particular, the clustering algorithm adopted is kNN. This algorithm has been executed 15 times with different seeds to find a robust partitioning of the labels in the training set. The number of occurrences for each label inside the clusters is computed as the average on the repetitions[2].

The partitions are computed as described in Section 3.2.3.

For MNIST the following 3 partitions are extracted:

- $P1 = (9, 4, 7)$
- $P2 = (3, 5, 8)$
- $P3 = (0, 1, 2, 6)$

For CIFAR10 the following 2 partitions are extracted:

- $P1 = (2, 3, 4, 5, 7)$
- $P2 = (0, 1, 6, 8, 9)$

For CIFAR100 the following 3 partitions are extracted:

- $P1 = (52, 68, 60, 17, 37, 49, 71, 59, 12, 89, 96, 81, 76, 13, 85, 47, 56, 90, 31, 19, 95, 23, 58, 32, 30, 72, 15, 5, 67, 38, 94, 3, 55, 8)$

[2]The match of clusters resulting from each repetition is performed manually.

- $P2 = (20, 24, 86, 41, 29, 61, 16, 28, 57, 99, 91, 53, 9, 39, 22, 7, 0, 40, 10, 69, 48, 25, 36, 87, 11, 84, 2, 44, 78, 79, 6, 80)$
- $P3 = (1, 62, 92, 33, 51, 64, 70)$

Once the partitions are defined, the Partitioned dataset is built by relabeling each image in the training and verification sets with the partition corresponding to the original label.

Partition Assigner

The Partition Assigner is implemented with a CNN trained on the Partitioned dataset. For the experimentation, modified versions of CNN C (for MNIST), D (for CIFAR10), and G (for CIFAR100) are considered since they work on a different number of labels.

Unlabeled inputs are classified according to the following criterion:

$$outcome(s) = (Predicted\_label(s) \in Assigned\_Partition(s)) \quad (3.3)$$

where $Predicted\_label$ is the output of the CNN monitored, and $Assigned\_Partition$ is the output of the Partition Assigner. A failure is detected when $outcome = false$, otherwise the image is sent to the corresponding Partition Explorer.

Partition explorers

As the Partition Assigner, Partition Explorers are implemented considering CNN C, D, and G, with customizations depending on the number of labels into the partition.

The following criterion is used to classify unlabeled input:

$$outcome(s) = (Predicted\_label(s) == Explorer\_label_i(s)) \quad (3.4)$$

where $Explorer\_label_i$ is the explorer of the partition i triggered by the Partition Assigner. When $outcome = false$ a failure is detected, otherwise, the image is considered correctly classified.

3.3.6 Baselines

Cross Referencing Oracle

Cross-referencing oracle (CRO) is implemented like Srisakaokul *et al.* for *multiple-implementation testing* [64]. In particular, it is implemented as a *majority oracle* with the three CNN of Table 3.4 for each dataset, adjudging as correct output the most voted value. When the three CNN fully disagree, CRO refrains to make a decision (the output is considered *Pass*).

SelfChecker

SelfChecker (SC) is an automatic oracle for *in deployment* evaluation of CNN [73]. In particular, SC aims to evaluate the final output provided by the CNN monitored considering features extracted from the internal layers. SC also suggests advice (alternative prediction) in case of misclassification detected.

The workflow of SC is reported in Figure 3.9. In particular, firstly the training set is used to compute layer-wise density distributions for each layer using *kernel density estimation* (KDE). After that, a selection of layers is computed to find the optimal combination exploiting 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 a misclassification.

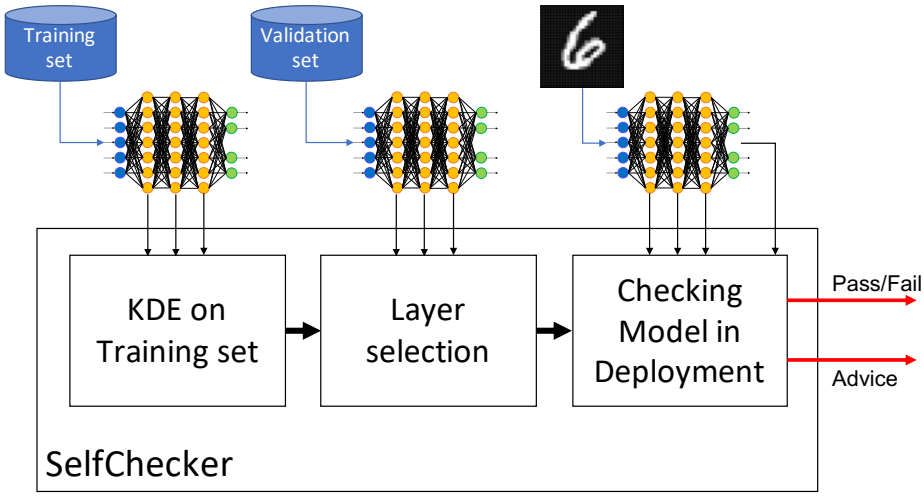


Figure 3.9. SelfChecker

3.4 Results

3.4.1 RQ1: effectiveness

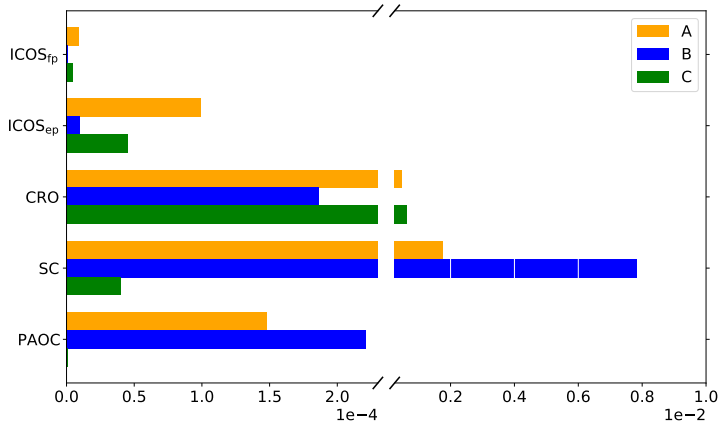
The effectiveness of the oracle surrogates is measured considering the MSE. As reported in Section 3.3.2, it is computed based on the *predicted accuracy* calculated on the output of each oracle, and the *actual accuracy* computed using the "true labels" removed for the experimentation.

The results in terms of MSE for each datasets are reported in Figure 3.10.

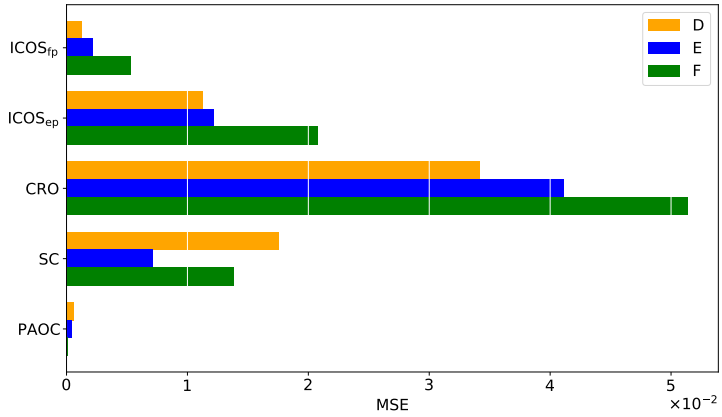
For the MNIST dataset (Figure 3.10a), ICOS and PAOC show the best values of MSE. In particular, ICOS_{fp} shows the best values for CNN A and B, while PAOC shows the best value for CNN C.

For CIFAR10 dataset (Figure 3.10b), PAOC outperforms all the other techniques in terms of MSE. ICOS_{fp} shows the second-best values, while ICOS_{ep} shows values worse than SC, except for CNN E.

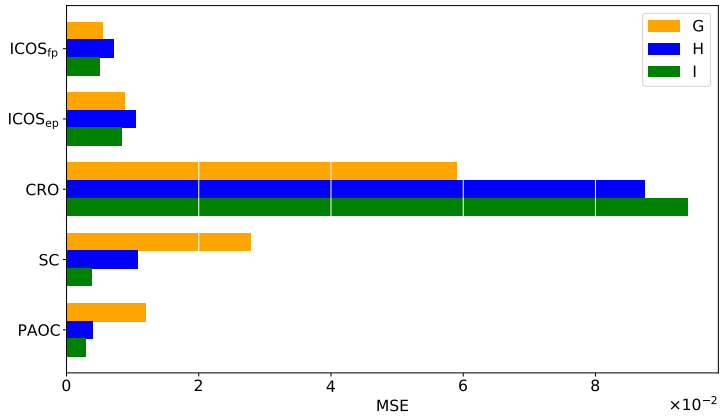
For CIFAR100 dataset (Figure 3.10c), the MSE showed by ICOS,



(a) MNIST



(b) CIFAR10



(c) CIFAR100

Figure 3.10. RQ1 (effectiveness): Mean Squared Error

PAOC and SC are very similar, except for CNN G, where the MSE of SC is higher than the others.

Overall, ICOS_{fp} and PAOC are robust in terms of MSE, showing low values considering both complex CNN (in terms of layer and parameters) and complex datasets (in terms of dimension of the images and the number of classes).

Since the MSE is an average value, the statistical significance of the results is assessed considering the *offsets*, computed for each repetition, each automatic oracle, and each subject. The Friedman test [25] is run to assess if there is a statistical difference among the offset showed by the five automatic oracles. The test returns $p\text{-}value = 2.5500e-173$, rejecting the null hypothesis "there is no difference among offsets".

Figure 3.11 reports the critical differences resulting by the application of the *post hoc* Nemenyi's test [8] on the offset values. Automatic oracles with no significant difference are grouped using a bold horizontal line – the more distant two oracles are (the distance being the average ranking) the smaller the p-value for the null hypothesis of equal performance. In the figure, the oracles are ordered from the worse to the best due to the chosen metric (higher values of offset correspond to worse values of predicted accuracy).

The critical differences show that there is no significant difference only between ICOS_{fp} and PAOC, which are confirmed to be the most performing automatic oracles for the assessment of the operational accuracy.

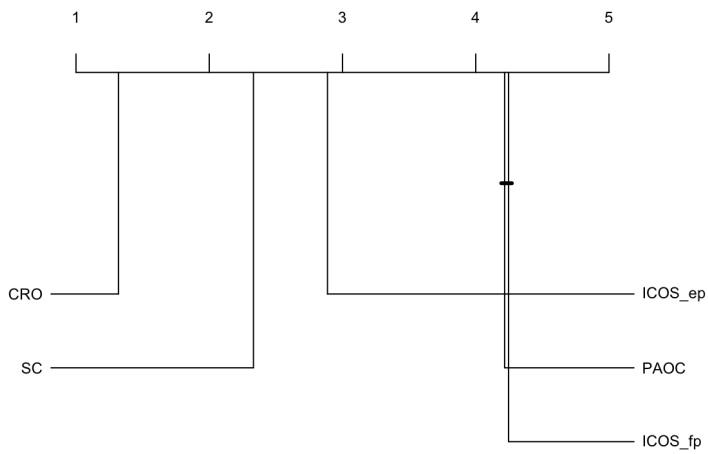


Figure 3.11. Plot of *post hoc* Nemenyi's test on offset values of automatic oracles

The results are also reported in terms of TPR, FPR and F1, metrics commonly used to evaluate oracles performance. The results are shown in Tables 3.7, 3.8, and 3.9, where the best values for each CNN are reported in **bold**.

The results in Table 3.7 show that SC achieves the best values of MSE for 5 CNN, while ICOS achieves the best values 4 times, 3 with ICOS_{ep} and 1 with ICOS_{fp}. About ICOS, the performance is influenced by the partitioning, in particular *fine partitioning* is more effective for MNIST images, and less effective for CIFAR10 ones. Curiously, although SC often shows the best values of TPR, its MSE is high.

According to the values reported in Table 3.8, CRO achieves the best FPR for all models. However, CRO shows the lowest values of TPR. For this reason, the MSE is very high compared to the other techniques considered. For this reason, the second-best values for each model are reported in *italic*.

SC achieves the best values of FPR for CNN C, H, and I, in correspon-

Table 3.7. RQ1: TPR

	TPR ↑				
CNN	ICOS_fp	ICOS_ep	CRO	SC	PAOC
A	0.7840	0.6252	0.5042	**0.7997**	0.7545
B	0.7641	0.6609	0.4078	**0.8364**	0.6161
C	**0.7704**	0.6495	0.3570	0.7236	0.5752
D	0.6143	**0.8487**	0.3621	0.8317	0.7609
E	0.6623	**0.8498**	0.4282	0.7926	0.7807
F	0.6921	**0.8913**	0.4142	0.8138	0.7347
G	0.7843	0.8474	0.2945	**0.8903**	0.7397
H	0.8168	0.8622	0.4166	**0.8893**	0.7892
I	0.7990	0.8476	0.3505	**0.8700**	0.7542

dence with its best values for the MSE. These values depend on the strong ability of SC to detect failures with a low number of false positives, in particular for complex models. On the opposite, the high presence of False Positives, in particular for simpler models, causes a strong divergence of the *Predicted accuracy*, resulting in high values of MSE. ICOS and PAOC show better values in terms of MSE due to the better balancing between TPR and FPR.

A consideration is that SC is preferable when the final purpose is to spot true failures of the CNN, instead of assessing the accuracy provided in operation.

The last metric considered is the F1, which expresses the balancing between precision and recall. The values reported in Table 3.9 confirm the considerations deducted observing the relations among MSE, TPR, and FPR. In fact, ICOS and PAOC show the best values of F1 respectively in 5 (3 for ICOS_{fp}, and 2 for ICOS_{ep}) and 2 cases. SC shows the best values for CNN H and I, which correspond to the best values of MSE, while it shows the worse values for CNN A and B, where it significantly underperforms compared to the other techniques in terms of MSE.

Table 3.8. RQ1: FPR

	FPR ↓				
CNN	ICOS_fp	ICOS_ep	CRO	SC	PAOC
A	*0.0075*	0.0077	**0.0040**	0.0532	0.0243
B	0.0085	*0.0083*	**0.0071**	0.0958	0.0266
C	0.0076	0.0077	**0.0023**	*0.0052*	0.0184
D	0.2166	0.2168	**0.0440**	0.3460	*0.1296*
E	0.2441	0.2448	**0.0379**	0.2450	*0.1533*
F	0.2854	0.2851	**0.0294**	0.2709	*0.1466*
G	0.2209	*0.2190*	**0.0391**	0.3073	0.2960
H	0.2810	0.2817	**0.0230**	*0.2607*	0.2647
I	0.2619	0.2615	**0.0195**	*0.1946*	0.2608

3.4.2 RQ2: sensitivity analysis on ICOS invariants

The main characteristic that distinguishes ICOS from the other techniques is the possibility to perform tuning of the TDIs to achieve high TPR still having an acceptable number of False Positives. In Figure 3.12 three different tuning of ICOS_{fp} are considered to evaluate how the failure detection ability changes.

The configurations considered are the following:

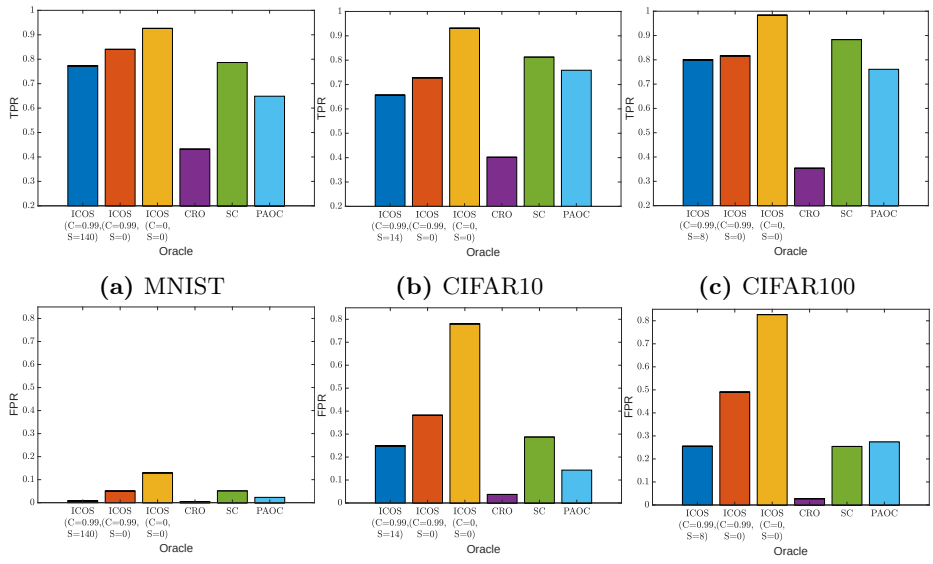
- *Criterion 1*: Select invariants with high confidence ($C = 0.99$) and high support, greater than 140 for MNIST, $S = 14$ for CIFAR10, and $S = 8$ for CIFAR100;
- *Criterion 2*: Select invariants with confidence greater than 0.99 ($C = 0.99$), regardless of support ($S = 0$); this is expected to give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

Table 3.9. RQ1: F1

	F1 ↑				
CNN	ICOS_fp	ICOS_ep	CRO	SC	PAOC
A	**0.8086**	0.7007	0.6318	0.5455	0.6650
B	**0.7581**	0.6928	0.4956	0.3276	0.4900
C	**0.7895**	0.7082	0.5024	0.7841	0.5724
D	0.5801	0.7212	0.4777	0.7175	**0.7832**
E	0.6200	0.7312	0.5560	0.7083	**0.7587**
F	0.6282	**0.7423**	0.5538	0.6927	0.7253
G	0.7069	**0.7442**	0.4095	0.7143	0.6364
H	0.7435	0.7705	0.5637	**0.7931**	0.7359
I	0.7365	0.7631	0.4985	**0.8100**	0.7088

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

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



(a) MNIST **(b)** CIFAR10 **(c)** CIFAR100

(d) MNIST **(e)** CIFAR10 **(f)** CIFAR100

Figure 3.12. RQ2: ICOS sensitivity to invariant selection criteria

3.4.3 RQ3: stability analysis

An analysis of stability is performed on ICOS, PAOC, and SC to evaluate the performance in presence of *label shift*[3]. This phenomenon makes the images of the training dataset less representative of the operational ones. The training dataset extracted from MNIST has been mutated, as by Li et al. [34], to simulate the label shift. In particular, couples of labels are switched while the operational dataset is unchanged. Three different label switches are considered: (2,7), (5,6), and (6,8). The CNN considered for this analysis is C, representing the best case of SC (it shows the best MSE). The three versions of CNN C trained on the mutated datasets are named C_1, C_2, and C_3. As shown in Figure 3.13, the performance of the

[3]CRO is omitted from RQ3 because it is unable to address the problem of *label shift* due to its design. The misclassifications cannot be detected because all the CNN would be trained on the same mutated training set with shifted labels.

three techniques depend on the specific mutation adopted:

- in the first case (C_1), ICOS outperforms PAOC and SC thanks to the IDIs, that can detect failures due to the label shift;
- in the second case (C_2), ICOS$_{ep}$ degrades its performance due to the ineffectiveness of its IDIs to the label shift;
- in the last case (C_3), the three techniques perform badly in the same way.

The results show that ICOS is the only technique robust against some label shifts thanks to the IDIs, which represent specific characteristics of the operational environment. Although the defined IDIs are very simple, they are useful to detect failures of the CNN in case of a strong distribution shift, with consequent label shift. SC and PAOC are not robust against label shift since they rely on the knowledge encoded into the training set.

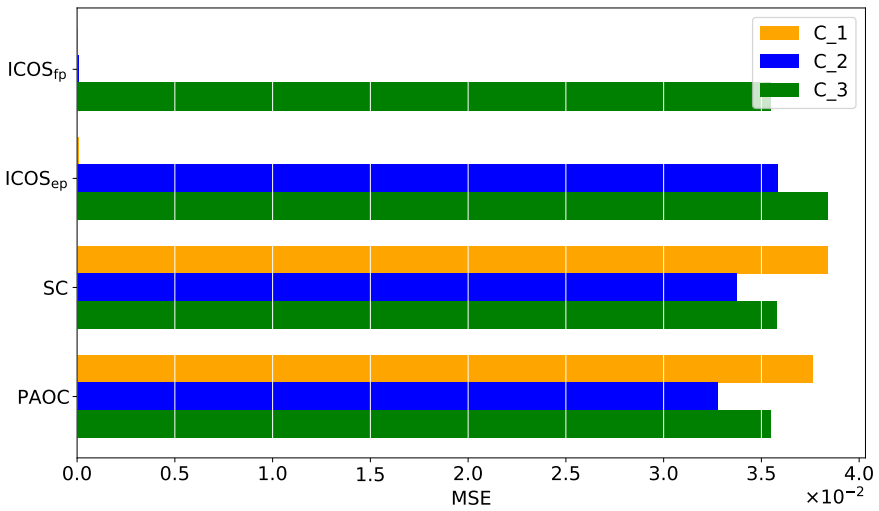


Figure 3.13. RQ3 (stability): MSE in case of *label shift*

3.5 Discussion

This chapter presented ICOS and PAOC as new contribution for automatic oracles for operational accuracy assessment of CNN. Their effectiveness in estimating operational accuracy has been demonstrated by comparison with CRO and SC, representing the best automatic oracle in the state of the art. The results show that despite the high number of False Positives that affect automatic oracles, the *predicted accuracy* is a faithful representation of the actual accuracy of the CNN monitored. Studying the performance of the pseudo-oracles in the case of *label shift*, ICOS shows how a rule derived directly from the operational domain makes it more robust to unexpected conditions in the operational domain.

Chapter 4

Offline operational accuracy assessment

This chapter presents DeepEST, a technique for the offline CNN assessment of the operational accuracy [19]. It aims to estimate the accuracy of CNN by selecting the most interesting images (the ones causing *misclassifications*) collected during the operational phase. The aim is to reduce the size of huge operational datasets, very costly to be manually labeled.

This assessment strategy can be integrated into the *experimental stage* of the life cycle presented in Chapter 2 as *Offline Assessment* phase. It takes data collected in operation with the CNN predicted labels as input and provides the estimated accuracy and the sampled images (labeled by human classifier) to the Data Preprocessing phase. As for the Online Assessment phase, the estimated operational accuracy can be used in the Data Preprocessing and Remodeling & Retraining phases to correct/improve ML system behavior. A graphical representation of the life cycle is reported in Figure 4.1.

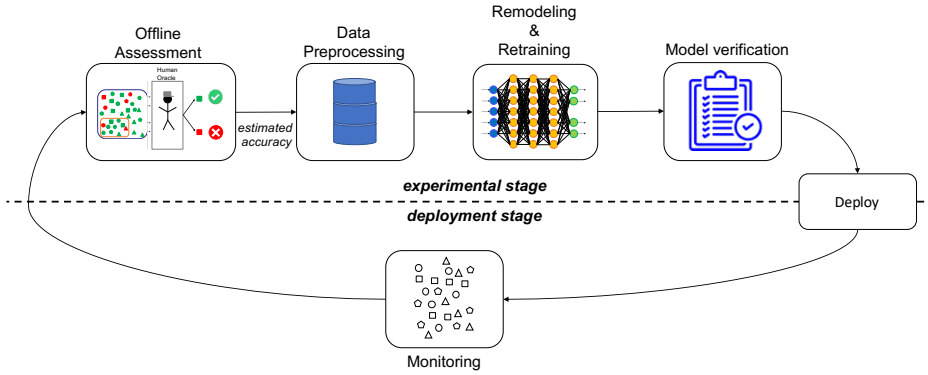


Figure 4.1. ML systems life cycle with Offline Assessment

4.1 Sampling-based assessment

The idea of the sampling-based assessment is to exploit sampling techniques to reduce the operational dataset to a smaller subset of representative examples. Based on the sampling process and sampled examples an estimate of the operational accuracy provided by CNN is provided.

The estimate of operational accuracy is influenced by the way images are sampled. In particular, two strategies can be considered:

- sampling the examples building a subset representative[1] of the population of examples in the operational dataset;
- sampling the most interesting examples (e.g. misclassified images) and balancing the selection bias during the computation of the estimate.

In the sampling theory, the sample is the set of n examples (namely images) $S = \{s_1, \ldots, s_n\}$, having a binary outcome, *Fail* or *Pass*. Outcomes are a series of independent Bernoulli random variables z_{s_i} such that

[1] A subset of examples is representative if it contains the same proportion of mispredictions as the operational dataset.

$z_{s_i} = 1$ if the CNN predicts the correct label for s_i, $z_{s_i} = 0$ otherwise. The operational accuracy of the CNN is the parameter of interest, θ. The aim is to obtain an estimate $\hat{\theta}$ with two desirable properties: *unbiasedness* – i.e., the expectation of the estimate $\mathbb{E}[\hat{\theta}]$ should be equal to the true value θ - and *efficiency* – for the given the sample size, the variance of $\hat{\theta}$ should be as small as possible (implying a highly-confident, stable estimate). The probability that $z_{s_i} = 1$ corresponds to the true (unknown) proportion: $\theta = \frac{\sum_{s=1}^{N} z_{s_i}}{N}$, with N being the population size (i.e., the size of the operational dataset).

4.2 Auxiliary variables

In sampling theory, auxiliary variables are supposed to have a correlation with the variable of interest, which, in this case, is operational accuracy. Auxiliary variables are used to reduce the variance of estimates obtained through sampling. The final effect of adopting those variables is to guide sampling through the most interesting examples (misclassifications). For this reason, values of the auxiliary variable have to be associated with each operational example encoding the amount of interest for that example. Since the main interest is in misclassifications, the auxiliary variables considered in this thesis encode the probability of a wrong classification by the CNN on each operational image. In particular, they are:

- the *confidence* value of each classification provided by the CNN;
- the *distance* between the images in the operational dataset and the ones in the training dataset.

The confidence (CS) aims to exploit the output of the last layer of the CNN monitored, namely a probability vector encoding how much the CNN

is "convinced" of the label provided as output. The assumption is that low values of confidence are highly related to failing examples.

The *distance* is based on the result by Kim *et al.* [28], showing that inputs more "distant" from training data are more likely to cause mispredictions. In particular, they propose two different metrics: *Distance-based Surprise Adequacy* (DSA) and *Likelihood-based Surprise Adequacy* (LSA). These are computed by the *Activation Trace* (AT), namely a vector of activation values of each neuron of a certain layer corresponding to a certain input; In this case, the AT is computed with reference to the last CNN activation layer.

LSA uses Kernel Density Estimation (KDE) to estimate the probability density of each activation value, obtaining the surprise of a new input with respect to the estimated density. LSA is computed as a measure of rareness:

$$P_{LSA} = -log(\hat{f}(x)) \tag{4.1}$$

where $\hat{f}(x)$ is the KDE applied to the new input x. LSA can be used both in the case of classification and regression models.

DSA is computed starting from the Euclidean distance between the AT of the new input x and the ATs observed during training:

$$P_{DSA} = \frac{dist_a}{dist_b} \tag{4.2}$$

where $dist_a$ is the distance between the ATs of the new input x and its nearest neighbour belonging to the same class x_a, $dist_b$ is the distance between the ATs of x_a and its nearest neighbour belonging to a different class x_b.

The effectiveness of an auxiliary variable strongly depends on the CNN and the training/operation dataset: for instance, for a distance metric, the belief that examples far from the one in the training set are more likely to cause a misprediction is not an absolute truth. In particular, if an example

is very similar to many others in the training set according to the distance metric, but has a different label, distance will be not a good metric to select it. Similarly, the confidence is a good proxy *if* the CNN is well trained for the operational context. In general, relying on a single auxiliary variable may work well in some settings and bad in others. Combining multiple beliefs is a choice that is expected to improve the stability of results across multiple settings.

Based on this, a further variant of our algorithm, named DeepEST$_C$, is defined considering as auxiliary variable the combination of confidence and distance:

$$P_C = P_{CS} \times (1 - norm(P_{DSA})) \tag{4.3}$$

where P_{CS} is the confidence value, $norm(P_{DSA})$ is the DSA value normalized [0, 1][2] and P_C is probability of correct prediction. The intuition behind this is that the probability of a correct prediction is related to both the confidence of the CNN and the *drift* of the example from what was seen during training. In fact, P_{CS} is related to the probability that the CNN does a correct prediction *according to what was seen during training* (in other words: it is the probability of correct prediction with perfect training); P_{DSA} is a proxy for the probability of the wrong prediction related to how far the example is from what seen during training – hence due to the *imperfection* of training. If confidence P_{CS} is high *and* the example is close to the training dataset (i.e., P_{DSA} is small), there is a high chance of correct prediction.

[2]In DeepEST$_C$, DSA is preferred to LSA since it has been shown to have better performance for the deeper layer [28].

4.3 DeepEST: Deep neural networks Enhanced Sampling Technique

4.3.1 Overview

The idea behind DeepEST [19] is to assess the CNN accuracy favoring the sampling of misclassified images. Since mispredictions are usually rare compared to correct examples, the strategy adopted by DeepEST is to exploit a sampling algorithm for rare populations, inspired to the Adaptive Web Sampling [24], to both spot as many failures as possible, and, at the same time, provide an unbiased estimate of the accuracy provided in operation. To achieve this task, the sampling is performed according to a piece of information, namely an auxiliary variable, which encodes the probability of failure of each input collected in operation.

Figure 4.2 shows the workflow conceived for DeepEST. All the unlabeled images submitted to the CNN in operation (*operational data*) are collected in a dataset. Starting from the output of the CNN and from the training data, the auxiliary variables are computed for each new operational image. Thus the DeepEST samples the operational data according to the chosen auxiliary variable. Finally, each image is labeled by a human oracle. The output of DeepEST consists of an operational accuracy estimate and a set of labeled operational images, which can be used to retrain the CNN.

4.3.2 Sampling strategy

The sampling strategy adopted by DeepEST focuses on selecting the most interesting examples by exploiting an auxiliary variable. A *weight* $w_{i,j}$ between any pair of images i and j of the operational dataset is defined starting from the auxiliary variables. The space of examples is explored adaptively following the weights. For instance, assuming to use the nor-

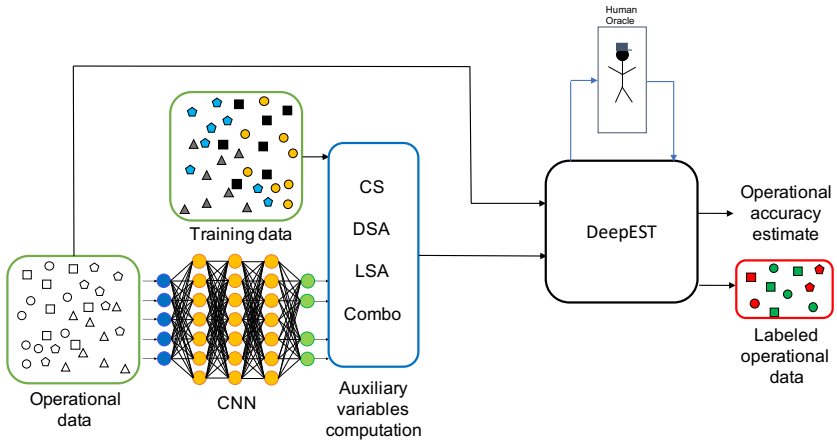


Figure 4.2. DeepEST workflow

malised DSA *distance*: if the example i has distance P_{DSA_i} (representing the belief that i causes a misprediction), and a pre-defined threshold τ is exceeded (i.e., i has a sufficiently high distance compared to the others), then all the $w_{i,j}$ values ($\forall j$ of the operational dataset) are set to their distance P_{DSA_j}; otherwise $w_{i,j} = 0$. This way, a strong-enough belief about example i causing misprediction entails the activation of all the weights toward i. The latter is used, as explained hereafter, for sampling, and makes the algorithm follow the distance criterion to spot potential clusters of failing examples.

The DeepEST sampling strategy is sketched in Algorithm 3. Assuming n examples to be selected from the operational dataset for the assessment, the algorithm selects one image per step. The first input is selected by simple random sampling, namely, initially all examples have equal probability of being selected. Then, one of two sampling schemes is used to select next example: *weight-based sampling* (WBS), or *simple random sampling* (SRS). Example i is selected from the operational dataset at step k with

probability $q_{k,i}$ given by:

$$q_{k,i} = r \cdot \frac{\sum_{j \in s_k} w_{i,j}}{\sum_{h \notin s_k, j \in s_k} w_{h,j}} + (1 - r) \cdot \frac{1}{N - n_{s_k}} \tag{4.4}$$

where:

- r: probability of using WBS (hence, probability of using SRS = 1-r; r is set to 0.8 in our implementation);
- s_k: current sample (all examples selected up to step k);
- $w_{i,j}$: weight relating example j in s_k to example i;
- n_{s_k}: size of the current sample s_k;
- N: size of the operational dataset.

For both WBS and SRS, the selection is *without replacement*.[3] WBS selects an example i proportionally to the sum of weights $w_{i,j}$ of already selected examples toward i – the chance of taking i depends on the current sample, favouring the identification of clusters of failing examples *if* the auxiliary variable (hence, $w_{i,j}$) is well-correlated with mispredictions. As this is not always the case, the WBS "depth" exploration is balanced by SRS, chosen with probability (1-r), for a breadth exploration of the example space. This diversification in the search is useful to escape from unproductive cluster searches. The steps are repeated until the budget $n \leq N$ is over.

At step 1, the probability that a randomly selected example will cause a misprediction is estimated as the outcome y_1 (1 in case of misprediction, 0 otherwise).

[3] Without replacement sampling schemes generally give smaller variance than their with replacement counterpart on the same sample size [36].

Algorithm 3 DeepEST sampling strategy

OD: operational dataset; i: current example; S: sample; n: number of examples; r: probability of WBS

1: $S = \emptyset$
2: $i = SRS\_sampling(OD)$; //first example by SRS
3: $OD = OD \setminus i$; //remove example from dataset
4: $S = S \cup i$; //add to sample
5: $y_1 = labelling\_and\_checking(i)$
6: **for** $k = 2; k <= n$; k++ **do**
7: $rs = random(0, 1)$
8: **if** $rs < r$ **then**
9: $i = WBS\_sampling(OD)$; $OD = OD \setminus i$;
10: **else**
11: $i = SRS\_sampling(OD)$; $OD = OD \setminus i$;
12: **end if**
13: $S = S \cup i$
14: $y_i = labelling\_and\_checking(i)$
15: $z_k =$ *Equation* 4.5
16: **end for**
17: $\hat{\theta} =$ *Equation* 4.6; //compute final estimate

At step k>1, example i, whose outcome is y_i, is selected with probability $q_{k,i}$ according to Eq. 4.4, and the estimator of the misprediction probability is that by Hansen-Hurwitz [21]:

$$z_k = \frac{1}{N}\left(\sum_{j \in s_k} y_j + \frac{y_i}{q_{k,i}}\right) \tag{4.5}$$

where the y_j values are the outcome of the examples already selected. z_k is an unbiased estimator of the expected misprediction probability at step k; the final estimator of the expected *operational accuracy* of the CNN is

1 minus the average of the z_k values:

$$\hat{\theta} = 1 - \frac{1}{n}(y_1 + \sum_{k=2}^{n} z_k) \tag{4.6}$$

where n is the number of sampled images.

4.4 Experimentation

4.4.1 Implementation

DeepEST is implemented mostly in Java. The implementation of the distance metric in DeepEST$_{DSA}$ and DeepEST$_{LSA}$ is the same used by Kim *et al.* [28]. Their Python scripts have been used to compute DSA and LSA values.

These are computed considering the last *activation layer* of each CNN. The threshold τ needed for the weights definition are set as follows:

- DeepEST$_{DSA}$: $\tau = mean(DSA) + 2 \times Std(DSA)$;
- DeepEST$_{LSA}$: $\tau = mean(LSA) + Var(LSA)$.

The threshold for *confidence*, used by DeepEST$_{CS}$ and DeepEST$_{C}$, is set to 0.7, assuming that lower confidence values are more related to misprediction (i.e., the weights are activated when the confidence is less than $\tau = 0.7$).

4.4.2 Baselines

Simple random sampling

Simple random sampling (SRS) is the first baseline approach. SRS draws independent and identically distributed examples directly from the

operational dataset. The estimator of the expected operational accuracy is reported in Equation 4.7.

$$\hat{\theta} = \frac{1}{n} \sum_{i=1}^{n} y_i \tag{4.7}$$

Where n is the number of examples selected, and y_i is 1 when the image is correctly classified by the CNN or 0 otherwise.

SRS is the simplest and cheapest way for carrying out the assessment. For the experimentation, the implementation provided by Li *et al.* [34] is used.

Cross-entropy Sampling (CES)

Li *et al.* [34] present Cross Entropy-based Sampling (CES), a technique that performs sampling using the output of the m neurons in the last hidden layer. This information is assumed to be more robust to the operation context drift, and to be highly correlated with the prediction accuracy since it is directly derived from the linear combination of this layer's output. The CES sampling algorithm (Algorithm 4) builds the sample firstly selecting as random an initial set of examples, and then it selects the remaining examples trying to minimize the average cross-entropy between the probability distribution of the m-dimensional representation of the output of neurons computed on the operational dataset and the selected images.

The objective is to sample a set of images as much as possible representative of the operational dataset, namely if it contains the same proportion of mispredictions as the operational dataset.

For CES, the authors demonstrate that the estimator is the same as SRS (Equation 4.7).

For the experimentation, the same configuration as the original article [34] is adopted. The size of the initial sample is p=30, enlarged by a group

Algorithm 4 CES sampling algorithm

Input: O: operational dataset, n: sample size.
Output: S: sampled examples for labeling ($|S| = n$).

1: Selecting randomly p examples as the initial sample S.
2: **while** $|S| < n$ **do**
3: Randomly select l groups of examples, $Q_1, ..., Q_l$ from O. Each group contains $min(q, n - |S|)$ examples.
4: Choose the group that minimizes the cross entropy, i.e.,

$$Q^* = \min_{Q_i} \overline{CE}(S \cup Q_i), i = 1, ..., l \tag{4.8}$$

5: $S \leftarrow S \cup Q^*$
6: **end while**

Q^* of q=5 examples at each step. The number of random groups from which Q^* is selected is $L = 300$.

4.4.3 Research questions and experiment design

The empirical evaluation is designed to answer the following research questions.

- **RQ1** (effectiveness): *How does DeepEST perform in estimating a CNN operational accuracy and simultaneously finding inputs causing misprediction (i.e., failing examples)?*

- **RQ2** (sensitivity): *How does the performance of DeepEST vary with the sample size?*

- **RQ3** (stability): *How does DeepEST behave in presence of label shift?*

4.4.4 Subjects

The subjects considered for the experimentation are the same as in Section 3.3.3, and the same training, verification, and operational datasets sizes are considered.

The number of examples is set to 200 for each session, then varied to answer RQ2. The experiments are repeated 30 times for statistical significance.

4.4.5 Evaluation Metrics

The metrics considered for the experimentation are the following:

- **Mean Squared Error** (MSE): $MSE(\hat{\theta}) = \frac{1}{30}\sum_{i=1}^{30}(\hat{\theta}_i - \theta)^2$, where θ is the true operational accuracy computed on the whole operational dataset, and $\hat{\theta}_i$ is the accuracy estimated by the sampling technique at repetition i (*estimated accuracy*);
- **Offset**: $offset_i = |\hat{\theta}_i - \theta|$, where θ is the true operational accuracy computed on the whole operational dataset, and $\hat{\theta}_i$ is the accuracy estimated by the sampling technique at repetition i.
- **Average Number of Failing Points** ($\overline{NFP}$): is the average number of failing examples sampled in a single session defined as $\overline{NFP} = Mean(NFP_i)$ with NFP_i being the number of failures at repetition i.
- **Variance of Failing Points** (VFP): is the variance of the number of failing examples detected over the 30 repetitions, it is defined as $VFP = \frac{1}{n-1}\sum_{i=1}^{n}(NFP_i - \overline{NFP})^2$

4.5 Results

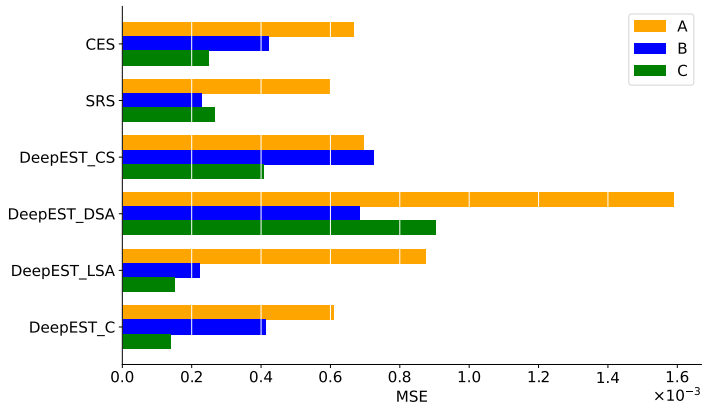
4.5.1 RQ1: effectiveness

To answer the first research question, the MSE is graphically represented in Figure 4.3, and the details of MSE, $\overline{NFP}$, and VFP are reported in Tables 4.1, 4.2, and 4.3. The best values for each CNN are reported in **bold**.

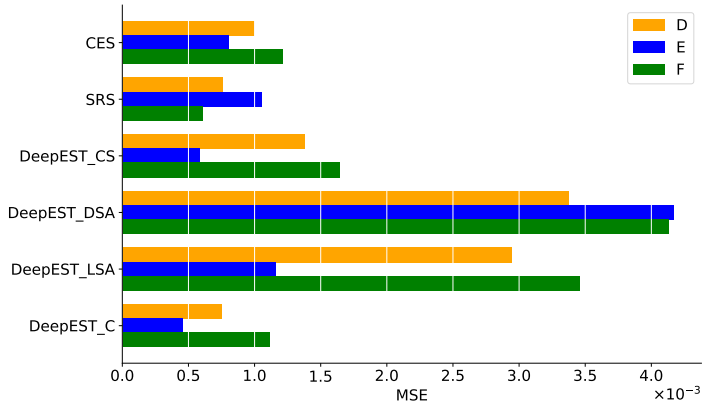
Table 4.1. RQ1: results (MNIST)

CNN	*Technique*	*Auxiliary Variable*	*MSE*	$\overline{NFP}$	*VFP*
A	CES	N/A	6.67E-04	22.2	20.2
	SRS	N/A	**5.98E-04**	18.7	24.0
	DeepEST	CS	6.96E-04	**87.4**	28.4
		DSA	1.59E-03	64.6	39.6
		LSA	8.73E-04	37.2	**18.6**
		C	6.09E-04	82.6	33.1
B	CES	N/A	4.22E-04	11.3	16.5
	SRS	N/A	2.28E-04	11.0	**9.0**
	DeepEST	CS	7.24E-04	**85.8**	24.5
		DSA	6.84E-04	56.7	40.0
		LSA	**2.22E-04**	24.7	14.5
		C	4.12E-04	80.5	52.9
C	CES	N/A	2.48E-04	14.0	**9.8**
	SRS	N/A	2.65E-04	13.5	10.9
	DeepEST	CS	4.07E-04	70.8	34.0
		DSA	9.02E-04	**97.8**	32.7
		LSA	1.51E-04	32.9	28.3
		C	**1.38E-04**	48.3	34.8

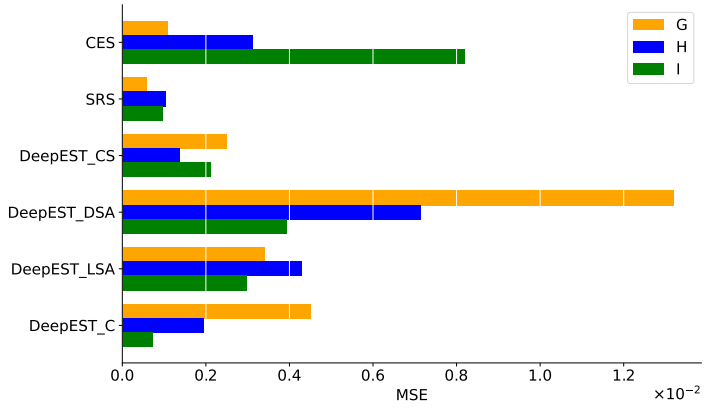
Observing the tables and the detail of the MSE in the corresponding figure, SRS and DeepEST$_C$ are the best strategies due to the higher number of best values achieved (4 each). Only for CNN B the best MSE is achieved by DeepEST$_{LSA}$.



(a) MNIST



(b) CIFAR10



(c) CIFAR100

Figure 4.3. RQ1 (effectiveness): Mean Squared Error

Table 4.2. RQ1: results (CIFAR10)

CNN	*Technique*	*Auxiliary Variable*	*MSE*	$\overline{NFP}$	*VFP*
D	CES	N/A	9.96E-04	56.2	40.6
	SRS	N/A	7.57E-04	56.0	**30.3**
	DeepEST	CS	1.38E-03	94.4	48.4
		DSA	3.37E-03	**100.2**	35.6
		LSA	2.94E-03	58.8	40.4
		C	**7.47E-04**	81.7	56.5
E	CES	N/A	8.03E-04	38.5	**20.2**
	SRS	N/A	1.05E-03	42.4	43.3
	DeepEST	CS	5.81E-04	83.8	42.6
		DSA	4.17E-03	**99.7**	39.8
		LSA	1.16E-03	68.4	24.5
		C	**4.58E-04**	64.9	40.5
F	CES	N/A	1.21E-03	70.0	50.1
	SRS	N/A	**6.07E-04**	69.7	**25.1**
	DeepEST	CS	1.64E-03	**111.3**	37.4
		DSA	4.13E-03	105.5	30.9
		LSA	3.46E-03	79.4	48.7
		C	1.11E-03	97.9	46.9

About the number of failures detected, DeepEST is the best technique, in particular with the CS and DSA auxiliary variables achieving all the best values (respectively 4 and 5 times). The ability to find failures showed by these two auxiliary variables overcame the capacity of the estimator balancing the sampling. This condition causes a higher MSE estimating the operational accuracy of the CNN.

In terms of VFP, SRS achieves the best values for more times (4). The VFP increases with less accurate CNN since failures are very frequent, and the auxiliary variables are less representative of the classification ability of the considered CNN.

CES never shows the best value in terms of MSE. Moreover, it is re-

lated to the worst value for CNN I trained on CIFAR100, showing that minimizing cross-entropy does not provide examples representative of the operational data.

The most interesting result is that although SRS is a very simple technique, it is very stable independently from the different datasets and CNN.

Table 4.3. RQ1: results (CIFAR100)

CNN	*Technique*	*Auxiliary Variable*	*MSE*	$\overline{NFP}$	*VFP*
G	CES	N/A	1.08E-03	66.3	43.5
	SRS	N/A	**5.75E-04**	67.2	**23.7**
	DeepEST	CS	2.50E-03	128.2	41.8
		DSA	1.32E-02	**141.9**	33.5
		LSA	3.40E-03	130.4	34.9
		C	4.50E-03	134.6	31.4
H	CES	N/A	3.12E-03	76.0	41.0
	SRS	N/A	**1.04E-03**	85.5	42.8
	DeepEST	CS	1.37E-03	130.5	45.6
		DSA	7.13E-03	**134.**0	**19.9**
		LSA	4.28E-03	113.3	68.7
		C	1.94E-03	128.2	50.1
I	CES	N/A	8.20E-03	65.8	55.2
	SRS	N/A	9.68E-04	82.2	40.0
	DeepEST	CS	2.12E-03	**133.4**	56.5
		DSA	3.93E-03	116.2	**24.9**
		LSA	2.98E-03	118.7	39.3
		C	**7.27E-04**	92.0	35.3

As in the previous Chapter, a Friedman test is applied to the *offsets* of the six samplers, computed for each repetition, each technique, and each subject. The test returns $p - value = 1.7148e - 35$ rejecting the null hypothesis that there is no difference among techniques offsets.

Figure 4.4 reports the pairwise comparison resulting from the *post hoc* Nemenyi's test, as performed in Chapter 3. As in the previous chapter, the

samplers are ordered from the worse to the best due to the offset metric. The plot confirms the considerations about MSE values, DeepEST$_C$ and SRS are the most performing samplers estimating the accuracy (there is no statistical difference between them). Moreover, DeepEST with CS and LSA auxiliary variable does not differ significantly from CES. DeepEST$_{DSA}$ is the worst sample since it shows the worst offset values.

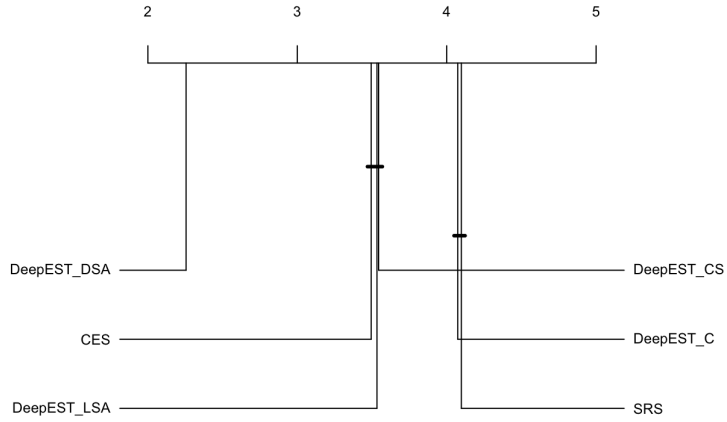


Figure 4.4. Plot of *post hoc* Nemenyi's test on offset values of samplers

The same statistical analysis is performed for the number of failing points. The Friedman test returns $p - value = 9.3124e - 262$ rejecting the null hypothesis that there is no difference among techniques. Figure 4.5 reports the pairwise comparison among the techniques resulting from the *post hoc* Nemenyi's test. Since a higher number of failing points corresponds to better techniques, the ranking of the samplers is in the conventional order. DeepEST with CS and DSA auxiliary variables are confirmed to be the best approaches, while CES and SRS are confirmed to be the worse ones.

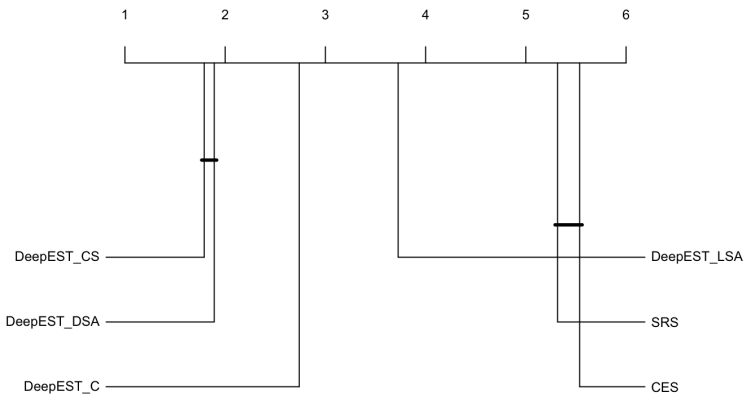
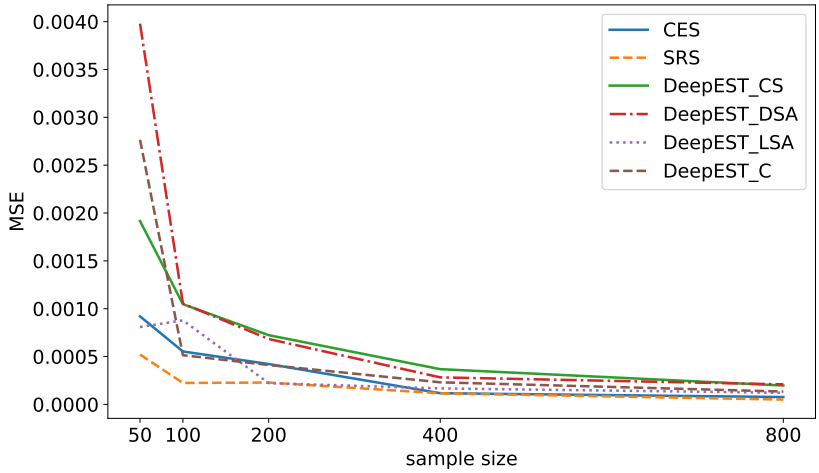


Figure 4.5. Plot of *post hoc* Nemenyi's test on failing points of samplers

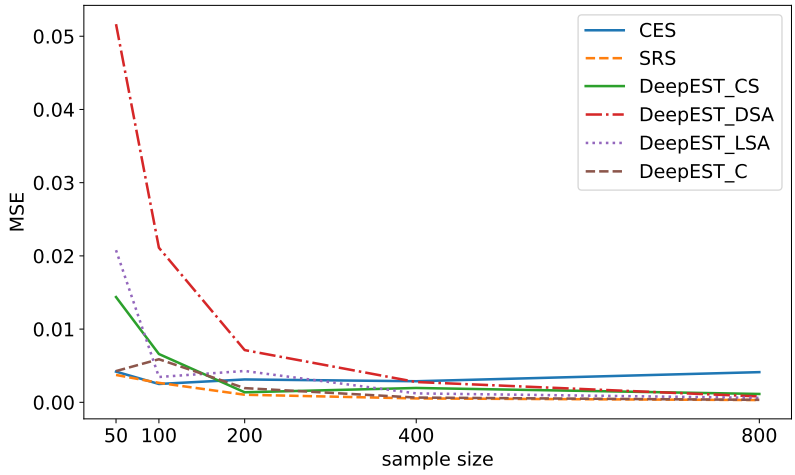
4.5.2 RQ2: sensitivity

To answer this RQ, experiments are run with the sample sizes 50, 100, 200, 400, 800, considering the subject with the highest accuracy (B), and the one with the lowest accuracy (H), to analyze how DeepEST performs when there are very few and many failing examples in the dataset, respectively. Figures 4.6 and 4.7 plot the MSE and the average number of failing points, respectively. Increasing the sample size all techniques, except CES for model H, exhibit a decreasing trend in MSE and an increasing trend in failing examples.

For CNN B (highest accuracy), surprisingly, SRS shows the best performance in terms of MSE becoming better with the increasing sample size. Observing both the MSE and the failing examples detection ability, DeepEST$_C$ is the best approach. In particular, it converges with MSE estimates of both SRS and CES, increasing the sample size, and it has the second-best values in terms of $\overline{NFP}$ (the best approach is DeepEST$_{CS}$) which correspond to more than seven times the number of failures detected by CES and SRS.

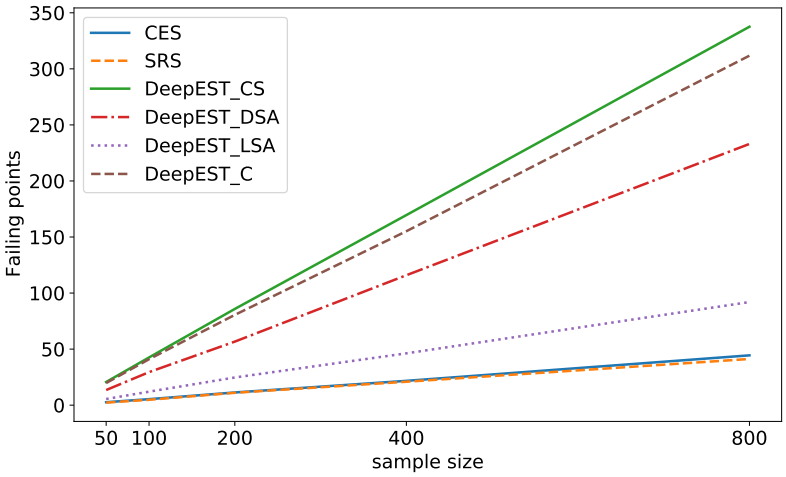


(a) model B

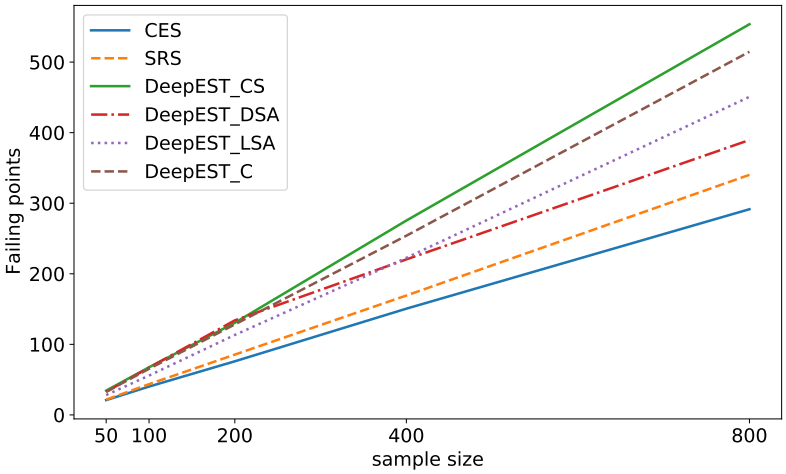


(b) model H

Figure 4.6. RQ2: Sensitivity to sample size (MSE)



(a) model B



(b) model H

Figure 4.7. RQ2: Sensitivity to sample size (average number of failing points)

For CNN H (lowest accuracy), CES and SRS outperform DeepEST$_C$ for 100 sample sizes; from 200 to 800, the estimation by CES starts diverging, while SRS and DeepEST keep good performance. This anomalous trend for CES depends on its lower ability to find failures of the CNN H. The number of failures detected by CES is lower than the ones detected by SRS. Performance in failing examples detection is always clearly in favor of all DeepEST variants. As for the ability to detect failing examples, *confidence* is the best auxiliary variable for DeepEST for CNN H: it presents the best values in all configurations. This time, it can be preferable compared to DeepEST$_C$ due to the similar performance in terms of MSE.

4.5.3 RQ3: stability analysis

As for ICOS and PAOC in Section 3.4.3, a sensitivity analysis is performed to measure how the proposed techniques are robust in the case of operational dataset very different from the dataset used in training, like in the case of label shift. Three versions of CNN C trained on the mutated datasets (label switches (2,7), (5,6), and(6,8)) are considered (C_1, C_2, and C_3). The results in terms of MSE are reported in Figure 4.8, and the detailed results are reported in Table 4.4 with the best values for each variant reported in **bold**.

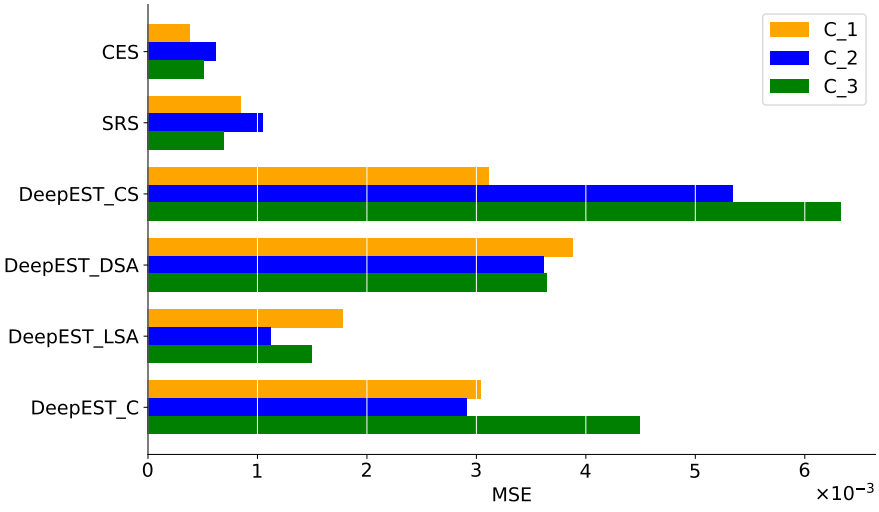


Figure 4.8. RQ3 (stability): MSE in case of *label shift*

The results show that techniques like SRS and CES, which aim to obtain a sample as much as possible representative of the operational dataset, are more robust to label shift, in particular, CES show the best values, thank a very low variance in the number of failing examples detected.

DeepEST, in particular the version with DSA auxiliary variable, is still very strong in terms of failures detected.

Table 4.4. RQ3: results

CNN	*Technique*	*Auxiliary Variable*	*MSE*	$\overline{NFP}$	*VFP*
C_1	CES	N/A	**3.78E-04**	50.6	**15.28**
	SRS	N/A	8.45E-04	50.9	34.16
	DeepEST	CS	3.11E-03	93.9	41.44
		DSA	3.88E-03	**112.1**	53.54
		LSA	1.78E-03	68.1	51.17
		C	3.04E-03	76.8	43.48
C_2	CES	N/A	**6.18E-04**	49.6	**24.05**
	SRS	N/A	1.05E-03	48.0	43.21
	DeepEST	CS	5.34E-03	91.4	50.25
		DSA	3.61E-03	**112.0**	34.65
		LSA	1.12E-03	66.0	34.17
		C	2.91E-03	73.3	55.60
C_3	CES	N/A	**5.08E-04**	49.8	**20.81**
	SRS	N/A	6.87E-04	49.2	27.32
	DeepEST	CS	6.33E-03	91.5	42.19
		DSA	3.64E-03	**115.2**	52.51
		LSA	1.49E-03	64.7	55.53
		C	4.49E-03	72.9	36.16

4.6 Discussion

This section presented DeepEST as a technique for sampling-based assessment of CNN. Its versatility allows practitioners to use the best auxiliary variables to select the images for the assessment. DeepEST ability to find failures is evident from the experiments. It finds up to eight times the number of failures than the baselines. This characteristic makes the estimated accuracy diverge when a *label shift* occurs since images with low interest in terms of auxiliary variable value fail unexpectedly. An interest-

ing finding overall of the three discussed RQs is that SRS, a very simple technique, can perform very robustly when estimating the accuracy of a CNN.

This page intentionally left blank.

Chapter 5

CNN Accuracy Assessment Cycle (AAC)

This chapter proposes a possible combination of the online and offline assessment strategies presented in the previous chapters according to their characteristics and how they can be integrated into the ML systems life cycle. In particular, the objective is to exploit their complementarity to reduce the application cost.

The chapter is structured as follows: in the first section, the cost of the two types of assessment is presented; the definition and implementation of the Accuracy Assessment Cycle are presented in the second section; finally, a simulation of the AAC is reported.

5.1 Assessment cost

The assessment techniques presented in chapters 3 and 4 are very effective in estimating the operational accuracy of CNN. These techniques are characterized by an application cost, which can represent a threat to

their usage in a continuous monitoring perspective.

5.1.1 Online Assessment

The online assessment performed via automatic oracles is characterized by a fixed cost for the knowledge extraction from the training dataset. In particular, each technique presented in Chapter 3 requires a "training" cost, defined as follows:

- ICOS requires training for TDIs and ADIs;
- PAOC requires training for Cluster Assigner and Explorers;
- SC requires to compute layer-wise density distributions for each layer, and each image in the training data;
- CRO requires training each alternative implementation.

CRO and PAOC can be considered more costly in terms of training compared to ICOS and SC, but ICOS and SC 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 cost for manual definition and parameters tuning can be considered low as they occur *una tantum*, while the training cost is necessary only when the training and verification sets are modified. Compared to the cost of training the CNN, the training cost can be considered acceptable since they can be executed mostly simultaneously.

5.1.2 Offline Assessment

The offline assessment via sampling techniques is characterized by the manual labeling cost, very high due to the need for human intervention each time they are executed.

Except for Simple Random Sampling, all the other sampling techniques are characterized by a cost for the computation of the auxiliary information.

The weight of this computation depends on the considered auxiliary information:

- the confidence does not require any computation and is collected directly during the monitoring of the ML system;
- Surprise Adequacy metrics (DSA and LSA) require heavy computation since they need to compare each operational image with the ones in the training set;
- Cross-entropy (related to CES technique) must be computed at each sampling step.

Despite the high cost, the average MSE of the sampling-based techniques is very low, even in the case of label shift (like for SRS), and in particular, it is lower than the MSE of the automatic oracles. The difference between the MSEs is one order of magnitude (on average).

Moreover, the high cost of the manual labeling of the operational images is balanced by the possibility to use the labeled samples to enlarge the training dataset, providing more representative images of the operational environment.

5.2 Accuracy Assessment Cycle

The way to reduce the assessment cost by exploiting all the advantages of the presented strategies is to combine them in an Accuracy Assessment Cycle, as reported in Figure 5.1.

The following phases are defined:

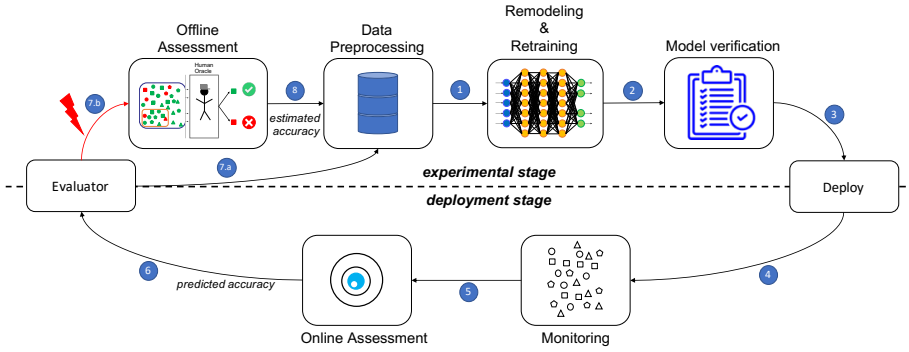


Figure 5.1. Accuracy Assessment Cycle (AAC)

- **Data Preprocessing**: the starting phase of the cycle, where the training dataset and the verification datasets can be updated both considering new labeled images (if available) and based on the accuracy estimates provided by the previous phases (if available). The aim is to build a training set representative of the observed operational domain.

- **Remodeling and Retraining**: the model is trained from scratch (first iteration or in case of remodeling) or according to the output of the previous phases using the new training set in input from the Data Processing phase.

- **Model verification**: in this phase, the accuracy of the ML system trained in the previous phase is computed on the verification dataset provided by the Data Preprocessing phase.

- **Deploy**: the CNN is deployed into the execution environment with all the instrumentation needed to collect new (unlabeled) samples.

- **Monitoring**: all the new samples are collected coupled with the prediction of the CNN deployed and additional information proper of

the environment (image sources, user typologies, operational profile, etc.)

- **Online Assessment**: an automatic oracle is used to classify each output of the deployed CNN as *Pass* or *Fail*. The oracle predictions are used to compute the *predicted accuracy* of the CNN in operation.
- **Evaluator**: it triggers a sampling session each time the accuracy of the CNN deployed decreases significantly compared to the one estimated on the verification dataset before the release of the CNN (7.b). Otherwise, the sampling-based assessment is skipped (7.a).
- **Offline assessment**: in this phase, a set of images is sampled from the operational data and labeled by a "human oracle", and an estimate of the accuracy (*estimated accuracy*) achieved in operation is computed.

The idea to reduce the cost is to consider the online assessment for a continuous evaluation of the operational accuracy provided by the ML system and to reduce the cost of manual labeling, retraining, and remodeling, performing them only if required.

5.3 Simulation of the Accuracy Assessment Cycle

5.3.1 Implementation

The simulation of an AAC is performed using the following implementation. The automatic oracle considered is ICOS due to both its effectiveness in estimating the accuracy of the CNN on previously unseen data and also for the robustness against label shifts. The sampling technique considered is SRS since it is a simple technique and is robust against the label shift. Moreover, SRS has been preferred to DeepEST because the purpose of this simulation is to have an accurate assessment, and there is

no interest in having a high number of failing examples. DeepEST represents a more efficient solution when the purpose is to rapidly improve the CNN accuracy since it can spot the most interesting images (the ones causing misclassification).

The dataset considered for the simulation is MNIST. In particular, 1000 samples have been considered for training, 500 for the verification set, and 1000 unlabeled images (for each cycle) as the operational dataset.

The configuration chosen for ICOS is *fp* (as in Section 3.3.4), assuming that the input of each partition is generated by different input sources. The sample size considered for SRS is 500 since in the sensitivity analysis (Section 4.5.2) has been shown that for budgets greater than 400, SRS estimate converges to the true accuracy.

5.3.2 Simulation

The AAC is simulated running eight consecutive cycles. The *Online Assessment* is executed in each cycle. The *Offline Assessment* is executed only when the accuracy estimated through ICOS is lower than 0.05 compared to the one estimated on the verification set before the release. Each time the sampling-based assessment is performed, the new manually labeled samples are sent to the Data Preprocessing phase. In this phase, the new images can be integrated into the training and verification sets. The proportion between new and old samples in the training dataset can be varied according to the accuracy estimates provided by the last cycles. In Figure 5.2 a graphical representation of the cycles is provided and Table 5.1 shows the details of each cycle. The first three cycles represent the nominal conditions, namely when the training and validation set well represent the operational dataset. As expected, the predicted accuracy computed through ICOS does not trigger any alarm in the first three cycles.

Starting from the fourth cycle, a label shift is simulated by switching labels 2 and 7 in the operational data. Thanks to IDIs, the failures are

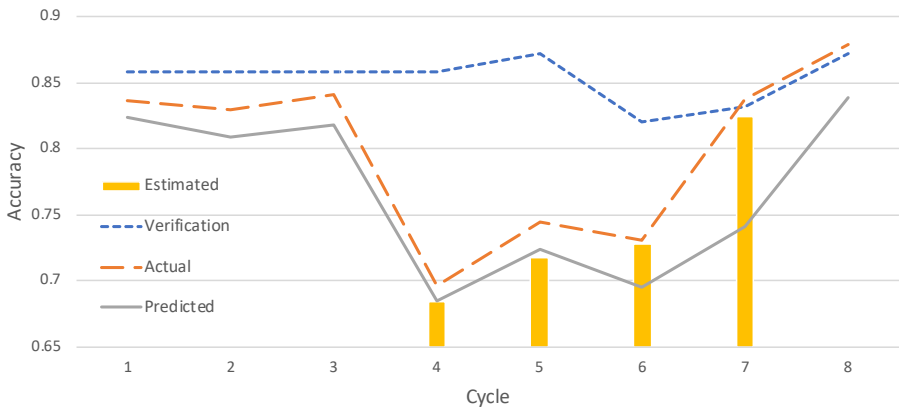


Figure 5.2. Plot of the results for each cycle

correctly detected. The accuracy drop detected thanks to the predicted accuracy triggers the offline assessment. The estimated accuracy confirms that the operational dataset is starting to diverge from the one observed in the previous cycles. Thus, the new samples are inserted into the training dataset, and the model is retrained accordingly.

During cycles 5 and 6, the accuracy drop is still correctly detected by ICOS, and the rest of the cycle is repeated as previously explained.

In cycle 7, ICOS provides a False Alarm, which triggers the sampling-based assessment phase. Observing the value of the estimated accuracy during the data processing phase, it is evident that the false alarm provided by ICOS (largely due to the TDIs) depends on a problematic design of the training set of the CNN. For this reason, in the last cycle, only the images collected in the last four cycles are used for training. After this retraining, the accuracy of the CNN on the current operational profile grows up significantly, and the predicted accuracy returns converge to the true value.

Table 5.1. Detailed results per cycle

	Accuracy				
Cycle	***Verification***	***Actual***	***Predicted***	***Estimated***	***Alarm***
1	0.858	0.837	0.824	N/A	No
2	0.858	0.830	0.809	N/A	No
3	0.858	0.841	0.818	N/A	No
4	0.858	0.696	0.685	0.684	True
5	0.872	0.744	0.724	0.718	True
6	0.820	0.731	0.695	0.728	True
7	0.832	0.838	0.741	0.824	False
8	0.872	0.879	0.839	N/A	No

5.3.3 Considerations

Figure 5.2 shows that the accuracy computed before the release can be very different from the one achieved in operation (*Actual*), in particular in cycles 4, 5, and 6.

The accuracy predicted by ICOS follows the actual accuracy with the operational data, except in cycle 7, where the bias in the training set causes many false positives.

The estimate provided by SRS is very useful to evaluate the accuracy provided in operation, and its high cost (due to manual labeling) is avoided in many cycles thanks to the predicted accuracy provided through the automatic oracle.

Chapter 6

Conclusions

Machine Learning models represent a very general solution for very specific problems. The literature about automatic oracles shows how ML models can be used to detect failures of other ML models seeing the same training data from different perspectives. PAOC is an extreme example of this, since it uses a slightly modified version of the same CNN under assessment, with very performing results.

A common practice in the current research is to try to solve the oracle problem in a general manner. This objective contrasts with the specificity of the task to be solved - dedicated solutions are expected to be more efficient. In fact, features of the operational domain represent important information to faithfully assess the accuracy of ML systems once they are deployed, and this aspect is not explored enough in the literature. A more interesting contribution for companies is to define a methodology that can be applied in any operational context and can improve the failure detection, and consequently the faithfulness of the accuracy assessment. Although operational features can be too specific for classifiers, troubling the generalization ability of the ML system, they can be very useful to evaluate the output of such systems on the fly (like IDI for ICOS).

The power of ML models has been demonstrated to go further than the mere classification task. As for human beings, physical symptoms can hide structural problems; similarly, ML models can be analyzed to generate new features (e.g. confidence, DSA, LSA) demonstrated to help evaluate the operational behavior since they encode the faithfulness of the output provided.

This thesis focuses on the Image Classification task, which is one of the most studied classification tasks in the literature. IC is well known to be a very complex task (even for human beings) for many reasons, like intra-class variation, variation in terms of scale, rotation, blurring, illumination, and so on.

In this work, the characteristics of the ML systems are considered to address the operational accuracy assessment problem in the IC domain adopting two different strategies. Contextually, three techniques have been proposed: ICOS and PAOC, as techniques for the online assessment via automatic oracles, and DeepEST, for the offline assessment via sampling.

The experiments performed on nine different CNN and three popular datasets (MNIST, CIFAR10, and CIFAR100) show that each strategy presents pros and cons depending on its specific nature:

- *Online assessment* via automatic oracles provides a continuous evaluation of the operational accuracy of the CNN, but it requires implementation and training cost to be added to the CNN life cycle.

- *Offline assessment* via sampling provides an on-demand faithful estimate of the operational accuracy of the CNN and a new set of labeled images very representative of the operational domain. On the opposite, it involves high costs due to the manual labeling of selected samples.

The mentioned characteristics of the two adopted strategies allow combining them by defining an Accuracy Assessment Cycle. This cycle aims to

continuously assess the operational accuracy of the ML systems, reducing the cost of the offline assessment. As shown in the last chapter, the automatic oracle can follow the operational accuracy variations of the CNN, triggering the offline assessment only when required, minimizing human intervention.

Steps forward can be done about the implementation of the Accuracy Assessment Cycle. The life cycle conceived as a loop helps engineers to collect features about the operational environment to specialize the ML systems performing the task they need in the way they need. This research could also help to find specific strategies for the improvement of the CNN in the loop. Techniques like DeepEST can spot a high number of failing examples, which, along with operational features, can help improve the performance of ML systems also in corner cases.

A possible advancement of the Accuracy Assessment Cycle can also integrate the automatic improvement both at the experimental and deployment stage. As shown in the last Chapter, rarely is required to change a performing model in case of unexpected phenomena like label shift. Often, additional training or training from scratch of the model incorporating the operational examples in the training set can be sufficient to improve the operational accuracy. For this reason, in line with MLOps perspectives, strategies for the online auto-improvement of ML models can be based on the "probabilistic" output of the Automatic Oracles. Moreover, automating data preprocessing, the offline retraining step can be run without human intervention when remodeling is not required.

The assessment, and consequently the improvement, of the accuracy of the ML systems can be of interest beyond the IC domain, and, more in general, beyond the classification domain. In particular, the AAC can find applications in the Autonomous Driving domain, also impacting industrial practices. The steering angle prediction, a regression problem, represents a possible application. For this purpose, both online and offline

assessment require specific customization: IDI must be defined in the Autonomous Driving context (e.g. based on traffic rules); TDI and ADI must be reconceived to work on continuous values; new auxiliary variables to guide sampling must be found (e.g. autoencoders output) since confidence and DSA are not directly computable for ML models for regression.

This process poses new challenges related to the collection and evaluation of operational features, which can be useful but need very specific reasoning to infer useful constraints. An interesting research direction is to apply inferential engines on operational features to automatically extract operational constraints which can help focus the task on the final objective.

Bibliography

[1] S. Alla and S. K. Adari. *What Is MLOps?*, pages 79–124. Apress, Berkeley, CA, 2021.

[2] J. M. Alonso, A. Ramos-Soto, C. Castiello, and C. Mencar. Explainable AI Beer Style Classifier. In *SICSA Reasoning, Learning and Explainability Workshop*. CEUR, 2018.

[3] R. Ashmore, R. Calinescu, and C. Paterson. Assuring the machine learning lifecycle: Desiderata, methods, and challenges. *ACM Comput. Surv.*, 54(5), may 2021.

[4] M. R. Berthold, N. Cebron, F. Dill, T. R. Gabriel, T. Kötter, T. Meinl, P. Ohl, C. Sieb, K. Thiel, and B. Wiswedel. KNIME: The Konstanz Information Miner. In *Studies in Classification, Data Analysis, and Knowledge Organization (GfKL 2007)*. Springer, 2007.

[5] T. Borovicka, M. Jirina, and P. Kordík. Selecting representative data sets. In *Advances in Data Mining Knowledge Discovery and Applications*, pages 43–70. IntechOpen, 2012.

[6] K. Cai, C. Jiang, H. Hu, and C. Bai. An experimental study of adaptive testing for software reliability assessment. *Journal of Systems and Software*, 81(8):1406–1429, 2008.

[7] K. Cai, Y. Li, and K. Liu. Optimal and adaptive testing for software reliability assessment. *Information and Software Technology*, 46(15):989–1000, 2004.

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

[9] C. Cariou and K. Chehdi. Unsupervised nearest neighbors clustering with application to hyperspectral images. *IEEE Journal of Selected Topics in Signal Processing*, 9(6):1105–1116, 2015.

[10] R. H. Cobb and H. D. Mills. Engineering software under statistical quality control. *IEEE Software*, 7(6):45–54, 1990.

[11] C. Corbière, N. THOME, A. Bar-Hen, M. Cord, and P. Pérez. Addressing failure prediction by learning model confidence. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[12] P. Allen Currit, Michael Dyer, and Harlan D. Mills. Certifying the reliability of software. *IEEE Transactions on Software Engineering*, SE-12(1):3–11, 1986.

[13] A. Dwarakanath, M. Ahuja, S. Sikand, R. M. Rao, R. P. J. C. Bose, N. Dubash, and S. Podder. Identifying Implementation Bugs in Machine Learning Based Image Classifiers Using Metamorphic Testing. In *Proc. 27th ACM SIGSOFT Int. Symposium on Software Testing and Analysis (ISSTA)*, pages 118–128. ACM, 2018.

[14] Y. Feng, Q. Shi, X. Gao, J. Wan, C. Fang, and Z. Chen. Deepgini: Prioritizing massive tests to enhance the robustness of deep neural networks. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2020, page 177–188, New York, NY, USA, 2020. Association for Computing Machinery.

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

[16] S. Garg, Y. Wu, S. Balakrishnan, and Z. C. Lipton. A unified view of label shift estimation. In *34th Conference on Neural Information Processing System (NeurIPS)*, 2020.

[17] Google. MLOps: Continuous delivery and automation pipelines in machine learning.

[18] A. Guerriero. Reliability Evaluation of ML systems, the oracle problem. In *IEEE Int. Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 127–130, 2020.

[19] A. Guerriero, R. Pietrantuono, and S. Russo. Operation is the Hardest Teacher: Estimating DNN Accuracy Looking for Mispredictions. In *IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 348–358. IEEE, 2021.

[20] J. Guo and X. Wang. Image classification based on surf and knn. In *2019 IEEE/ACIS 18th International Conference on Computer and Information Science (ICIS)*, pages 356–359, 2019.

[21] Morris H. H. and William N. H. On the Theory of Sampling from Finite Populations. *The Annals of Mathematical Statistics*, 14(4):333–362, 1943.

[22] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015.

[23] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[24] D. G. Horvitz and D. J. Thompson. A generalization of sampling without replacement from a finite universe. *Journal of the American Statistical Association*, 47(260):pp. 663–685, 1952.

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

[26] G. Jahangirova, A. Stocco, and P. Tonella. Quality metrics and oracles for autonomous vehicles testing. In *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*, pages 194–204, 2021.

[27] M. I. Jordan and T. M. Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.

[28] J. Kim, R. Feldt, and S. Yoo. Guiding Deep Learning System Testing Using Surprise Adequacy. In *41st International Conference on Software Engineering*, ICSE, pages 1039–1049. IEEE, 2019.

[29] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical Report TR-2009, University of Toronto, 2009.

[30] N. Kühl, M. Goutier, L. Baier, C. Wolff, and D. Martin. Human vs. supervised machine learning: Who learns patterns faster? *CoRR*, abs/2012.03661, 2020.

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

[32] Y. LeCun and C. Cortes. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/, 2010.

[33] Z. Li, X. Ma, C. Xu, and C. Cao. Structural coverage criteria for neural networks could be misleading. In *41st Int. Conference on Software Engineering: New Ideas and Emerging Results*, ICSE-NIER, pages 89–92. IEEE, 2019.

[34] Z. Li, X. Ma, C. Xu, C. Cao, J. Xu, and J. Lü. 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)*, pages 499–509. ACM, 2019.

[35] R. C. Linger and H. D. Mills. A case study in cleanroom software engineering: the IBM COBOL Structuring Facility. In *12th Int. Computer Software and Applications Conference*, COMPSAC, pages 10–17. IEEE, 1988.

[36] S. L. Lohr. *Sampling: Design and Analysis.* Duxbury Press, 2nd edition, 2009.

[37] J. Lv, B. Yin, and K. Cai. Estimating confidence interval of software reliability with adaptive testing strategy. *Journal of Systems and Software*, 97:192–206, 2014.

[38] J. Lv, B. Yin, and K. Cai. On the asymptotic behavior of adaptive testing strategy for software reliability assessment. *IEEE Transactions on Software Engineering*, 40(4):396–412, 2014.

[39] L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu, J. Zhao, and Y. Wang. Deepgauge: Multi-granularity testing criteria for deep learning systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ASE 2018, page 120–131, New York, NY, USA, 2018. Association for Computing Machinery.

[40] L. Ma, F. Zhang, J. Sun, M. Xue, B. Li, F. Juefei-Xu, C. Xie, L. Li, Y. Liu, J. Zhao, and Y. Wang. DeepMutation: Mutation Testing of Deep Learning Systems. In *29th Int. Symposium on Software Reliability Engineering (ISSRE)*, pages 100–111. IEEE, 2018.

[41] L. Ma, F. Zhang, M. Xue, B. Li, Y. Liu, J. Zhao, and Y. Wang. Combinatorial testing for deep learning systems, 2018.

[42] S. Ma, Y. Liu, G. Tao, W. Lee, and X. Zhang. NIC: Detecting Adversarial Samples with Neural Network Invariant Checking. In *26th Network and Distributed System Security Symposium (NDSS)*, 2019.

[43] H. D. Mills, M. Dyer, and R. C. Linger. Cleanroom software engineering. *IEEE Software*, 4(55):19–24, 1987.

[44] C. Murphy, G. Kaiser, L. Hu, and L. Wu. Properties of Machine Learning Applications for Use in Metamorphic Testing. In *20th Int. Conference on Software Engineering and Knowledge Engineering (SEKE)*, pages 867–872, 2008.

[45] C. Murphy, G. E. Kaiser, and M. Arias. An approach to software testing of machine learning applications. In *19th Int. Conference on Software Engineering and Knowledge Engineering (SEKE)*, pages 167–172, 2007.

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

[47] A. Odena and I. Goodfellow. TensorFuzz: Debugging neural networks with coverage-guided fuzzing. In *Proc. 36th Int. Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, 2019.

[48] K. Pei, Y. Cao, J. Yang, and S. Jana. Deepxplore: Automated whitebox testing of deep learning systems. *Commun. ACM*, 62(11):137–145, oct 2019.

[49] R. Pietrantuono and S. Russo. On adaptive sampling-based testing for software reliability assessment. In *27th Int. Symposium on Software Reliability Engineering*, ISSRE, pages 1–11. IEEE, 2016.

[50] R. Pietrantuono and S. Russo. Probabilistic sampling-based testing for accelerated reliability assessment. In *IEEE Int. Conference on Software Quality, Reliability and Security*, QRS, pages 35–46. IEEE, 2018.

[51] R. Pietrantuono, S. Russo, and A. Guerriero. Run-time reliability estimation of microservice architectures. In *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*, pages 25–35, 2018.

[52] R. Pietrantuono, S. Russo, and A. Guerriero. Testing microservice architectures for operational reliability. *Software Testing Verification and Reliability*, 30(2), 2020.

[53] Y. Qin, H. Wang, C. Xu, X. Ma, and J. Lu. SynEva: Evaluating ML Programs by Mirror Program Synthesis. In *Int. Conference on Software Quality, Reliability and Security (QRS)*, pages 171–182. IEEE, 2018.

[54] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[55] B. Recht, R. Roelofs, L. Schmidt, and V. Shankar. Do ImageNet Classifiers Generalize to ImageNet? In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of Machine Learning Research (PMLR)*, volume 97, pages 5389–5400, 2019.

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

[57] Shai S. and Shai B. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, USA, 2014.

[58] S. Seo, S. Ö. Arik, J. Yoon, X. Zhang, K. Sohn, and T. Pfister. Controlling neural networks with rule representations. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

[59] M. Shaha and M. Pawar. Transfer Learning for Image Classification. In *Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, pages 656–660, 2018.

[60] N. Sharma, V. Jain, and A. Mishra. An analysis of convolutional neural networks for image classification. *Procedia Computer Science*, 132:377–384, 2018. International Conference on Computational Intelligence and Data Science.

[61] C. Silla and A. Freitas. A survey of hierarchical classification across different application domains. *Data Mining and Knowledge Discovery*, 22:31–72, 01 2011.

[62] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. Driessche, T. Graepel, and D. Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354–359, 10 2017.

[63] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[64] S. Srisakaokul, Z. Wu, A. Astorga, O. Alebiosu, and T. Xie. Multiple-implementation testing of supervised learning software. In *AAAI Workshops*. Association for the Advancement of Artificial Intelligence, 2018.

[65] A. Stocco, M. Weiss, M. Calzana, and P. Tonella. Misbehaviour prediction for autonomous driving systems. In *Proc. of the 42nd Int. Conference on Software Engineering*, ICSE. ACM, 2020.

[66] F. Sultana, A. Sufian, and P. Dutta. Advancements in image classification using convolutional neural network. In *Fourth Int. Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)*, pages 122–129. IEEE, 2018.

[67] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[68] Y. Tian, K. Pei, S. Jana, and B. Ray. DeepTest: Automated Testing of Deep-neural-network-driven Autonomous Cars. In *40th Int. Conference on Software Engineering (ICSE)*, pages 303–314. ACM, 2018.

[69] H. Wang, J. Xu, C. Xu, X. Ma, and J. Lu. Dissector: Input validation for deep learning applications by crossing-layer dissection. ICSE '20, page 727–738, New York, NY, USA, 2020. Association for Computing Machinery.

[70] M. Weiss and P. Tonella. Fail-safe execution of deep learning based systems through uncertainty monitoring. In *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*, pages 24–35, 2021.

[71] M. Weiss and P. Tonella. Uncertainty-wizard: Fast and user-friendly neural network uncertainty quantification. In *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*, pages 436–441, 2021.

[72] W. Wu, H. Xu, S. Zhong, M. R. Lyu, and I. King. Deep Validation: Toward Detecting Real-World Corner Cases for Deep Neural Networks. In *49th Annual IEEE/IFIP Int. Conference on Dependable Systems and Networks*, DSN, pages 125–137. IEEE, 2019.

[73] Y. Xiao, I. Beschastnikh, D. S. Rosenblum, C. Sun, S. G. Elbaum, Y. Lin, and J. S. Dong. Self-checking deep neural networks in deployment. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 372–384, 2021.

[74] X. Xie, L. Ma, F. Juefei-Xu, M. Xue, H. Chen, Y. Liu, J. Zhao, B. Li, J. Yin, and S. See. Deephunter: A coverage-guided fuzz testing framework for deep neural networks. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA, pages 146–157. ACM, 2019.

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

[76] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S.z Khurshid. DeepRoad: GAN-Based Metamorphic Testing and Input Validation Framework for Autonomous Driving Systems. In *33rd ACM/IEEE Int. Conference on Automated Software Engineering (ASE)*, pages 132–142. ACM, 2018.

This page intentionally left blank.

Author's publications

J1 R. Pietrantuono, S. Russo, A. Guerriero,
Testing Microservice Architectures for Operational Reliability,
Software Testing, Verification and Reliability (STVR),
Vol. 30, Issue 2 (2020)
DOI: 10.1002/stvr.1725

J2 A. Bertolino, G. De Angelis, A. Guerriero, B. Miranda, R. Pietrantuono, S. Russo,
DevOpRET: Continuous Reliability Testing in DevOps,
Journal of Software: Evolution and Process (JSEP) (2020)
DOI: 10.1002/smr.2298

C1 M. Camilli, A. Guerriero, A. Janes, B. Russo, S. Russo,
Microservices Integrated Performance and Reliability Testing,
3rd International Conference on Automation of Software Test (AST),
Pittsburgh, PA, 17-18 May 2022, ACM/IEEE (to appear)

C2 A. Guerriero, R. Pietrantuono, S. Russo,
Operation is the hardest teacher: estimating DNN accuracy looking for mispredictions,
43nd International Conference on Software Engineering (ICSE),
Madrid, Spain, 23-29 May 2021, pp. 348-358, IEEE
DOI: 10.1109/ICSE43902.2021.00042

C3 A. Bertolino, A. Guerriero, B. Miranda, R. Pietrantuono, S. Russo,
Learning-to-Rank vs Ranking-to-Learn: Strategies for Regression Testing in Continuous Integration,
42nd International Conference on Software Engineering (ICSE),
Seoul, South Korea, 06-11 July 2020, ACM
DOI: 10.1145/3377811.3380369

C4 A. Guerriero,
Reliability Evaluation of ML systems, the oracle problem,
31st IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW),
Coimbra, Portugal, 12-15 Oct. 2020, pp. 127-130, IEEE
DOI: 10.1109/ISSREW51248.2020.00050

C5 A. Guerriero, R. Mirandola, R. Pietrantuono, S. Russo,
A Hybrid Framework for Web Services Reliability and Performance Assessment,
1st International Workshop on Governing Adaptive and Unplanned Systems of Systems,
30th IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), IEEE
Berlin, Germany, Oct. 28, 2019, pp. 185-192,
DOI: 10.1109/ISSREW.2019.00070

C6 R. Pietrantuono, S. Russo, A. Guerriero,
Run-time Reliability Estimation of Microservice Architectures,
29th IEEE International Symposium on Software Reliability Engineering (ISSRE), Memphis, TN, 15-18 Oct. 2018, IEEE
DOI: 10.1109/ISSRE.2018.00014
(recipient of the **ISSRE 2018 BEST RESEARCH PAPER AWARD**)