Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 222 (2023) 499–508

Procedia Computer Science

www.elsevier.com/locate/procedia

International Neural Network Society Workshop on Deep Learning Innovations and Applications (INNS DLIA 2023)

EFFECT: An End-to-End Framework for Evaluating Strategies for Parallel AI Anomaly Detection

Matthias Stammler*, Julian Hoefer, David Kraus, Patrick Schmidt, Tim Hotfilter, Tanja Harbaum, Jürgen Becker

Karlsruhe Institute of Technology, Institut fuer Technik der Informationsverarbeitung, Engesserstraße 5, 76131 Karlsruhe, Germany

Abstract

Neural networks achieve high accuracy in tasks like image recognition or segmentation. However, their application in safety-critical domains is limited due to their black-box nature and vulnerability to specific types of attacks. To mitigate this, methods detecting out-of-distribution or adversarial attacks in parallel to the network inference were introduced. These methods are hard to compare because they were developed for different use cases, datasets, and networks. To fill this gap, we introduce EFFECT, an end-to-end framework to evaluate and compare new methods for anomaly detection, without the need for retraining and by using traces of intermediate inference results. The presented workflow works with every preexisting neural network architecture and evaluates the considered anomaly detection methods in terms of accuracy and computational complexity. We demonstrate EFFECT's capabilities, by creating new detectors for ShuffleNet and MobileNetV2 for anomaly detection as well as fault origin detection. EFFECT allows us to design an anomaly detector, based on the Mahalanobis distance as well as CNN based detectors. For both use cases, we achieve accuracies of over 85 %, classifying inferences as normal or abnormal, and thus beating existing methods.

Keywords: AI security; inference monitoring; layer tracing; evaluation framework; anomaly detection

1. Introduction

In recent years, Convolutional Neural Networks (CNNs) achieved higher accuracy in image recognition and segmentation tasks, than human respondents or traditional methods. Using dedicated hardware accelerators, neural networks have also entered embedded applications [6]. Many applications bear additional requirements regarding safety

This work was funded by the German Federal Ministry of Education and Research (BMBF) under grant number 16ME0454 (EMDRIVE). The responsibility for the content of this publication lies with the authors.

* Corresponding author. Tel.: +49 721 608-41322
E-mail address: stammlerkit.edu

1877-0509
10.1016/j.procs.2023.08.188

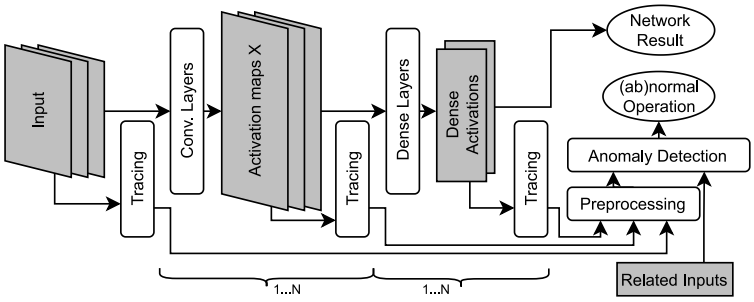

Fig. 1. Schematic of network tracing. Relevant data is marked gray, calculations and results are shown in white.

and security, e.g., in autonomous driving [18, 13]. In such safety-critical environments, mechanisms have to detect anomalies that occur during CNN inference. Those anomalies manifest themselves in different aspects: For example, an out of distribution (OOD) anomaly can occur, when the network has to deal with input data not present during training. Inevitably, a misclassification happens, since the network will always predict one of the classes it has been trained on. Similar behavior can be observed when the network operates in yet unknown situations like weather or daytime.

Besides naturally occurring problems, misclassification can also be the consequence of a malicious intent. Those misclassifications, which are referred to as *adversarial attacks*, can be evoked through, e.g., adding noise to the input image, that is not noticeable to the human eye. Other attacks mislead the network by adding special shapes on top of the input image, triggering a prediction desired by the attacker. Both occurrences can be regarded as an anomaly. Detecting these anomalies with neural networks, by tracing intermediate results, has been proven to be successful in [17, 1, 16]. Figure 1 shows a structure, where intermediate features are used to detect an input anomaly in parallel to the CNN inference. Many strategies use different datasets or metrics for evaluation, making a comparison difficult. Therefore, we introduce EFFECT, a tool, enabling comparison of newly developed methods with different anomaly detection strategies, known in literature. Our approach takes the anomaly detection accuracy as well as the introduced computational overhead into account and, makes a grounded comparison available. The contribution in this paper is thus:

- We present EFFECT, an end-to-end framework to evaluate and compare strategies to detect anomalies during CNN inference based on layer traces
- Using EFFECT, we implement and test new strategies to detect anomalies, showing Mahalanobis distance and CNNs as the best for our use cases
- We show an evaluation and comparison of state-of-the-art methods and our newly proposed combinations, using EFFECT on two use cases: (A) Detecting abnormal system behavior as well as (B) detecting fault origin.

2. Related work

Anomalies in neural networks can either be caused unintentionally by the environment or by hardware faults or be a result of intentionally induced attacks.

Out of distribution samples can be caused by inputs that were not part of the training dataset, or when dealing with dataset shifts when operating in conditions unknown to the network, such as weather or daytime. Detecting these samples can be done, e.g., with [10]. By using temperature scaling [2], this OOD detector observes the impact of the slight alterations to the input on the output. If the output drastically changes, the input likely is an OOD sample. While providing good results, this approach only works for OOD samples and needs multiple inferences for its decision. In addition to that, [9] allows for out of distribution detection, but only focuses on dens layers and needs training information for configuration.

Hardware faults, like random bit flips or stuck at faults, can be detected by [17]. In this approach, intermediate feature representations are traced and anomalies are detected using a pretrained neural network. The same approach is used to detect a noisy image in [16], where additional environmental information is considered and used for classifi-

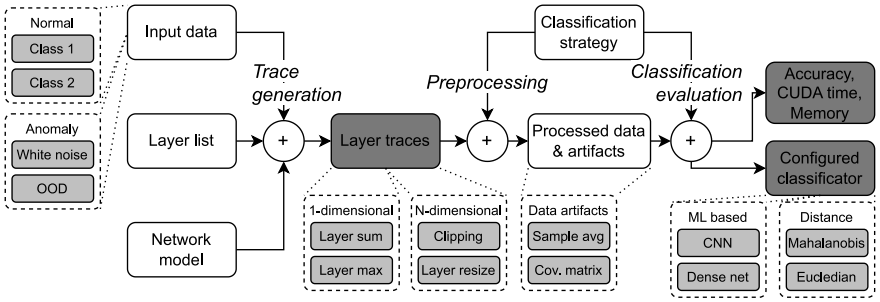

Fig. 2. Schematic diagram of framework workflow, starting with network model, layer list and input data. Arrows point towards (intermediate) generated artifacts. Gray boxes mark examples for the concepts shown in white boxes. Dark gray boxes signify generated artifacts suitable for further use.

cation. By calculating the distance between a fully connected layer of one inference to a given example, [5] introduces a method to label neural network decisions as problematic, by calculating the Euclidean distance between the current activations and a predefined sample. In contrast to [17, 16], this method only traces one and not many layers. The methods in [17, 16, 1] show good results but can neither be compared to each other nor to other newly introduced approaches, because of different datasets and metrics.

In addition to that, [17, 16] lack explainability, lying in the nature of its methods. Using machine learning methods to detect anomalies during inference of other machine learning methods only moves the problem and does not mitigate it. Besides the detection of anomalies, some approaches exist that try to train a network to be resilient against out-of-distribution or adversarial inputs [19, 7], or alter the structure to filter out appropriate input aspects. While providing acceptable results, these approaches only move the problem to a careful selection of training data.

Overall, the common problem with all methods is the lack of comparability between them, due to divergent metrics and different datasets. There exists no framework to compare them against each other or newly introduced anomaly detection methods. EFFECT closes this gap, by providing a framework for easy comparison and configuration of new and existing methods like [9, 17, 16, 1].

Algorithm 1 Framework overview

Require: Matching list M, classification list C, network model n, dataset D, traced layers L

function EVALUATE_STRATEGIES(M, C, n, D, L)
  $F_{\text{train}}, F_{\text{test}} \leftarrow$ TRACE_GENERATION(n, D, L)
  for $m \in M$ **do**
    $A \leftarrow$ PREPROCESSING(m, F_{train})
    for $c \in C$ **do**
      if COMPATIBLE(m, c) **then**
        $R \leftarrow$ EVAL_STRATEGY(c, m, A, F_{test})
      end if
    end for
  end for
end function

Algorithm 2 Trace generation

Require: Traced network n, complete dataset D, traced layers L

function TRACE_GENERATION(n, D, L)
  $D_{\text{train}}, D_{\text{test}} \leftarrow$ SPLIT(D)
  for $l \in L$ **do**
    $F_{\text{train},l} = \{f = n(d, l) \mid d \in D_{\text{train}}\}$
    $F_{\text{test},l} = \{f = n(d, l) \mid d \in D_{\text{test}}\}$
  end for
  $\forall\, f_{\text{train}} \in F_{\text{train},l} : F_{\text{train}} \leftarrow F_{\text{train}} \cup \{f_{\text{train}}\}$
  $\forall\, f_{\text{test}} \in F_{\text{test},l} : F_{\text{test}} \leftarrow F_{\text{test}} \cup \{f_{\text{test}}\}$
  return $F_{\text{train}}, F_{\text{test}}$
end function

3. Tracing framework concept

EFFECT describes a framework and end-to-end workflow for testing, evaluating, choosing and configuring different inference classification strategies for defined faults or normal operation of preexisting neural networks. All strategies share the same fundamental structure, using traces of intermediate feature representations and classifying them in parallel to the running inference. This chapter describes the three framework steps. In section 4, EFFECT gets evaluated by choosing strategies to distinguish network inferences into *normal* or *abnormal* runs, described in section 5.1. Afterward, the framework is then used a second time to test strategies detecting the fault origin and reason

for an *abnormal* inference (section 5.2). Algorithm 1, fig. 2 and the following chapters describe the EFFECT workflow in detail and shows the strengths of this framework. Providing high flexibility while simultaneously automating the train and test setup, the workflow encapsulates the steps as far as possible, to allow an automated test of a broad range of classification strategies.

EFFECT consists of three steps, *trace generation*, *preprocessing* and *classification evaluation*. A trained *network model* and dataset specifies the given use case. To specify real world application, *trace generation* is used to generate inference traces based on specified watched layers. The same layers are used during training, testing and evaluation. As shown in algorithm 1, a training dataset F_{train} and a test dateset F_{test} of feature map traces is generated, by doing an inference for every input dataset element $d \in D$ and saving the outputs of the specified layers in L. Trace generation keeps the original classification of the inference inputs for the corresponding traces in regard to test and train data, as well as predefined classes. Afterward, neither the original dataset D, nor the neural network model n is needed for *preprocessing* or *classification evaluation*, thus eliminating the need to retrain or modify the existing network. For every classification strategy $c \in C$, and for every dimension matching strategy $m \in M$, the following steps are executed. If the classification is compatible with the dimension matching strategy, the classification is prepared in the *preprocessing* step. Here, the training dataset F_{train}, consisting of the output traces marked for training, is used in junction with the dimension matching d to prepare a classification strategy c. Afterward, in the *classification evaluation* step, the test traces F_{test} are run through the classificator c using the dimension matching strategy m to create the results $R = (a, m_{\text{conf}}, t_{\text{CUDA}}, n_{\text{mem}})$. EFFECT traces the accuracy a, the confusion matrix m_{conf}, the CUDA time t_{CUDA} as well as the used memory n_{mem}. The three steps are explained in more detail below.

3.1. Trace generation

Traces are signified as feature activation maps F of previously specified layers in L. For every input of the training dataset D (split into predefined classes), an inference takes place for the given model n and the traces are saved. Depending on the configuration, the inference output might also play a role for classifying the traces and can be saved into F. Algorithm 2 shows the algorithm for *trace generation*. Required are a network model n, for which EFFECT should evaluate AI operation classification strategies, a dataset of inputs D, split into subsets representing the different operational classes, as well as a list of traced layers L. For each element in the dataset, an inference run of the existing network is done. During each inference, samples are fed through the machine learning model and intermediate feature traces, sorted by specified layer and split into the given classes, are collected. Creating traces f for each of the monitored layers in L based on the given datasets D comprises step one. Inside the framework, the trace for every feature activation map includes the complete data. No dimension matching takes place during *trace generation*. The generated traces, split into test and training, are then saved for later use.

3.2. Preprocessing

This section necessitates *dimension matching*, to equalize the dimensions of the traced feature maps for further classification, including dimension reduction if desired. For *further calculations*, knowledge over the used classification strategy is necessary. The algorithm describing this is listed in algorithm 3.

When generating traces of AI models, the output dimensions might not always be the same. In `MobileNet` for example, the first convolutional layer has an output dimension of $112 \times 112 \times 32$, while the fifth convolutional layer has an output dimension of $28 \times 28 \times 32$ [15], resulting in a dimensional mismatch when trying to concatenate feature maps. Equalizing the dimensions is necessary to proceed for calculating classification artifacts as well as for classification when using those artifacts. Matching the dimension of inputs between samples of different width and height can either be done by scaling the length and width of smaller or larger samples to the same value, or e.g., reducing the first two dimensions to a single value per feature map. The second strategy is used in, e.g., [16]. Here, each of the values of each activation map are summed, with this sum used as input into the anomaly detector. Evaluated dimension matching strategies are highly dependent on the use case. *Preprocessing* also includes training neural network based classification strategies. For every feature maps, `generate_artifacts` gets called, generating a set of artifacts A, necessary for classification, section 4.2 shows examples for dimension matching and the required artifacts for each strategy.

3.3. Classification evaluation

Classification evaluation takes place as the final and last step of EFFECT. Algorithm 4 shows this step. Depending on the used classification c, the function `classify` gets called, classifying the test sample f_{test}. Depending on the classifier, additional artifacts A might be necessary, such as pretrained neural networks or covariance matrices. Afterward, the results are collected. For confusion matrices and accuracy, this includes comparing it to the true class c_{true}. Additionally, timing information as well as used data points are evaluated. The configured classifier is saved, allowing for seamless integration and easy deployment, once a classifier is chosen.

Algorithm 3 Preprocessing

Require: dimension matching m, train dataset F_{train}
function PREPROCESSING(m, F_{train})
  $A = \{\}$
  for $f_{\text{train}} \in F_{\text{Train}}$ **do**
    $A \leftarrow A \cup \{\text{GENERATE\_ARTIFACTS}(m, f_{\text{train}})\}$
  end for
  return A
end function

Algorithm 4 Classification evaluation

Require: classification strategy c, dim. matcher m, classification artifacts A, test dataset F_{test}
function EVAL_STRATEGY(c, m, A, F_{test})
  for $f_{\text{test}} \in F_{\text{test}}$ **do**
    $c_{\text{pred}} = \text{CLASSIFY}(c, f_{\text{test}}, m, A)$
    $c_{\text{true}} = \text{GET\_CLASS}(f_{\text{test}})$
    $R \leftarrow R \cup \{\text{REGISTER\_RESULT}(c_{\text{pred}}, c_{\text{true}})\}$
  end for
  return R
end function

4. Anomaly Detection Strategies used for Evaluation

To show the capabilities of EFFECT, two different use cases are evaluated, (A) detecting anomalies in system behavior as well as (B) detecting the origin of the faults. The combination of seven different *classification* strategies with five different *dimension matching* strategies is shown in section 4.1 and section 4.2 as well as in Table 1 and Table 2.

For the use case of anomaly detection, these combinations were not found in literature except for a few combinations. These are: (a) with a specified threshold in [17]; (a,1) in [1]; (a,7) in [16]; as well as tracing every input with strategy (5) in [9]. In Table 3 and Table 4, these combinations are marked with footnotes.

Evaluation is done for two network models (`MobileNet` [15] and `ShuffleNetV2` [11]), tracing two convolutional layers each. Choosing an early and a late layer provides information about basic as well as advanced features. With different width and height between chosen layers, this provides the reason for the dimension matching strategies presented in section 4.2. Simulating *normal* system behavior, 10,000 correctly classified samples from the ImageNet [14] dataset are used. Representing a hardware fault, 10,000 white noise samples are fed through the deep learning models. Simulating inference on unknown inputs, ImageNet-o (representing out of distribution samples) and ImageNet-a (representing adversarial attacks) samples from [8] are fed through both models.

In a first experiment, the framework is used to select a strategy discriminating between *normal* and *abnormal* samples. In the second experiment, EFFECT is used, to choose a strategy for the detection of *abnormal* system behavior origin and classifying the traces into *normal* system behavior as well as *white noise*, *adversarial* or *out of distribution*. In neither case, a retraining nor altering of the networks was necessary. The evaluation is done on an NVIDIA RTX A6000, using driver version 520.61.50 and CUDA version 11.8.

4.1. Classification

Classification strategies specify the algorithm, deciding if the trace of an inference can be classified as normal or abnormal. They also necessitate the use of dimension matching strategies, and calculated artifacts. Some combinations between classification as well as the used dimension matching might not be possible. Table 1 gives an overview of the classification strategies, presented below. The table also shows compatible dimension matching as well as necessary input dimensions and classification artifacts. Classification strategies are numbered from (1) to (7) for easier reference.

(1) Euclidean distance: Calculate the Euclidean distance between the averages of different classes and the current sample and select the class, whose average has the lowest distance between the sample and it's average.

Table 1. Overview of classification strategy, input sizes, for classification necessary data calculated and compatible dimension matching strategies, outliers are marked in gray.

Classification	Input dimension	Calculation	Necessary artifacts	Dim. matching
(1) Euclidean distance	$N_W \times N_H \times N_s$	$c = \underset{c}{\operatorname{argmin}}\, d_c = \underset{c}{\operatorname{argmin}} \lvert f_{\text{avg},c} - f \rvert$	Class avg.	(a) - (e)
(2) Cross entropy	$N_W \times N_H \times N_s$	$H(Q) = -\frac{1}{N} \sum_{c=1}^{N} \log Q_c(f)$	Class avg.	(a) - (e)
(3) KL divergence	$N_W \times N_H \times N_s$	$D_{KL} = -\sum_{f \in F_{\text{train}}} Q_f(f) \log\left(\frac{Q_f(f)}{Q_c(f)}\right)$	Class avg.	(a) - (e)
(4) K nearest neighbor	$N_W \times N_H \times N_s$	Class of nearest k neighbors	-	(a) - (e)
(5) Mahalanobis dist.	$1 \times N_s + N_s \times N_s$	$d_M(x, Q_c) = \sqrt{(f - f_{\text{avg},c})^T S_c^{-1} (f - f_{\text{avg},c})}$	Avg. & cov. mat.	(a), (b)
(6) CNN based	$N_W \times N_H \times N_s$	Inference of CNN model	parameters	(c) - (e)
(7) Dense net	$1 \times N_s$	Inference of DNN model	parameters	(a), (b)

(2) Cross entropy: Calculate the entropy between the given sample and the average of the classes, where f is the currently processed sample and Q_c is the distribution over class c.

(3) Kullback–Leibler divergence [3]: Calculate the Kullback-Leibler divergence between the average of each class and the current sample. where f is the currently processed sample, Q_f the distribution over the sample and Q_c is the distribution over class c.

(4) K nearest neighbor: Classifying the sample based on k nearest neighbors of a subset of all training data, using Euclidean distance. In this paper, 1 % randomly selected samples are chosen.

(5) Mahalanobis distance [12]: Calculate the Mahalanobis distance between the averages of different classes and the current sample and select the class with the lowest distance, where f is the currently processed sample, $f_{\text{avg},c}$ is the average sample value, and S_c is the covariance matrix of the average of F_{train} of the current class c.

(6) CNN based classification: Train a convolutional neural network on activation traces with input sizes of $N_W \times N_H \times N_s$, where N_W is the width, N_H is the height, and N_S is the amount of traced feature maps. Using two convolutional and two dense layers and Sigmoid Linear Unit (SiLU) [4] activation function.

(7) Dense net classification: Train a fully connected neural network on activation traces with input sizes of $1 \times N_S$, where N_S is the amount of traced feature maps. Using two layers and Sigmoid Linear Unit (SiLU) [4] activation function.

4.2. Dimension matching

Resulting from the different prerequisites on the classification strategies in section 4.1, choosing the right strategy for matching the dimensions of different layer traces is important for successful classification. The evaluated dimension matching strategies are marked from (a) to (e), where the first two strategies reduce every feature map into a single value and strategies (c) - (e) reduce or enlarge feature maps to a common size, respectively. An overview is given in Table 2, where possibilities for combination with the compatible classification strategies as well as the input and output sizes are listed. A description of those strategies is given below. The representation of one trace sample is specified with t and consists of a set of feature map representations f_i. It is specified as $t = \{f_1, f_2, ..., f_N\}$, with N specifying the amount of traced feature maps.

(a) Layer Summation: Summation of all values inside one feature map into a single value, where $a_{i,j,k}$ denotes the activation value at indexes (j, k) in feature map f_i.

(b) Layer maximum extraction: Extract the maximum value of the activation, with $a_{i,j,k}$ denoting the activation value at indexes (j, k) in feature map f_i.

(c) Layer up-scaling: Perform bilinear interpolation on all feature activation maps to fit specified dimension sizes, where $f_i \in f$ denotes the ith feature map in the current sample, f specifies the processed sample, resize(f_i, c_x, c_y) the bilinear resizing function, c_x and c_y specify the target size in x and y dimension, respectively.

Table 2. Overview of dimension matching strategies, corresponding output sizes and compatible classification strategies.

Dimension matching	Calculation	Output size	Compatible classification
(a) Summation	$f = \{f_i \mid f_i = \sum_{j,k} a_{i,j,k}$ and $1 \leq i \leq \mathrm{N}\}$	1×1	(1) - (5), (7)
(b) Max. extraction	$f = \{f_i \mid f_i = \max_{j,k} a_{i,j,k}\ , 1 \leq i \leq \mathrm{N}\}$	1×1	(1) - (5), (7)
(c) Up-scaling	$f = \{f_i \mid \mathrm{resize}(f_i, c_x, c_y) \mid 1 \leq i \leq \mathrm{N}\}$	$W_{\mathrm{Max}} \times H_{\mathrm{Max}}$	(1) - (4), (6)
(d) Clipping	$f = \{f_i \mid \mathrm{clip}(f_i, c_x, c_y) \mid 1 \leq i \leq \mathrm{N}\}$	$W_{\mathrm{Min}} \times H_{\mathrm{Min}}$	(1) - (4), (6)
(e) Zero padding	$f = \{f_i \mid \mathrm{pad}(f_i, d_x, d_y) \mid 1 \leq i \leq \mathrm{N}\}$	$W_{\mathrm{Max}} \times H_{\mathrm{Max}}$	(1) - (4), (6)

(d) Layer clipping: Clip the middle part of activation with larger sizes to fit the size of the smallest activation, where $f_i \in f$ denotes the ith feature map in the current sample, f specifies the processed sample and $\mathrm{clip}(f_i, c_x, c_y)$ a function for returning the middle $c_x \times c_y$ entries of feature map f_i.

(e) Zero padding: Pad the perimeter of smaller feature activation maps with zeros, where $f_i \in f$ denotes the ith feature map in the current sample, f specifies the processed sample and $\mathrm{pad}(f_i, c_x, c_y)$ a function, returning a matrix with the dimension $d_x \times d_y$ and the middle entries equal to f_i. Every other entry is equal to zero.

Necessary artifacts: In addition to matching the dimensions of feature activation, an information compression is necessary for some classification strategies. These include averaging over all samples in one class. By interpreting every sample as a random variable, it is possible to additionally calculate the covariance matrix over all samples.

5. Results

EFFECT evaluates given strategies for inference run classification and dimension matching in three aspects. For each combination of *dimension matching* and *classification*, accuracy, confusion matrix, runtime as well as memory usage are tracked and the given graphs and tables are generated by the framework. Accuracy is calculated as $Acc = N_{\mathrm{correct}}/N_{\mathrm{total}}$. For computational complexity, GPU time is a good indicator. Omitting disk loading time and any CPU processing necessary, it only focuses on calculations inside the GPU itself. Memory consumption for necessary calculated data is presented as `float32` data points. Mini-batching is omitted for inference in this framework to be as close to a mobile use case as possible, using dedicated hardware accelerators, where mini-batching of incoming data is uncommon.

5.1. Anomaly detection

The samples are split into two classes, representing *normal* and *abnormal* input images. Correctly classified input images from ImageNet's validation dataset [14] are called *normal*. Traces generated from ImageNet-o and ImageNet-a samples [8] as well as traces generated from white noise are called *anomalies*. 75 % of all data used as training data and 25 % used as test data. For `MobileNet`, 96 feature maps and for `ShuffleNetV2`, 1048 feature maps are traced per sample. In contrast to conventional classification methods, ML classificators highly depend on parameter optimization to the given use case of traced layers and dimension matching strategies, resulting in a large design space. For the following chapters, we focus on an architecture with two dense layers, either after two convolutional layers or in sequence to dimension reduction methods, resulting in a one dimensional output. Accuracy is presented in percent, runtime in µs and memory consumption in the number of `float32` data points.

Accuracy: Table 3-A shows the classification results for traces passing through the two traced networks using accuracy as metric. For *Mahalanobis distance*, only dimension matching resulting in a single value per feature map is applicable ((a) and (b)). For *machine learning methods* the columns are split into fully connected networks (6) and CNNs (7). The experiment was repeated three times, with the highest accuracy value presented. `MobileNet` is marked light gray. Rows depicting `ShuffleNetV2` are white. Dark gray cells mark incompatible combinations. Classifications, where all samples are classified the same, thus failing the task, are marked with an X. Inconsistencies between percentages result from slightly unbalanced datasets. Low percentages not marked dark gray show results, not exclusive to a single class. A trend emerges, where (1) *Euclidean distance* is strong, while some *machine learning based* strategies (6) & (7) also show success in some combinations. The overall trend is constant for `ShuffleNetV2` and `MobileNet`. For `MobileNet`, the strongest combination is (c,6) and (d,6) (CNNs with *upscaling* or *clipping*)

for machine learning approaches, and (b,5) (*Mahalanobis distance* with *maximum extraction*) for conventional approaches. For `ShuffleNetV2`, (b,7) is the strongest machine learning approach.

Execution time: For parallel validation of an inference, using a parallel hardware module, reducing the computational complexity and choosing an approach that minimizes resource utilization is necessary to avoid large resource overhead. In fig. 3-A, the total CUDA time used to classify every test sample is shown. The Y axis is logarithmic, with shaded bar heads showing additional time used for `ShuffleNetV2` traces. No additional bar signals identical runtime. CUDA time is dependent on *dimension matching* as well as *classification* strategy. Reducing feature maps to a single value (*dimension matching* (a) and (b)) always results in a lower CUDA time. *Classification strategies* (1) - (3), (6) and (7) have lower CUDA time than (4) and (5). *K nearest neighbor* and *Mahalanobis distance* take longer for every *dimension matching* strategy.

Memory usage: In addition to computational complexity, reflected by CUDA time, the used memory consumption is a governing factor for usability in mobile and embedded applications. Table 4-A shows the memory consumption of every dimension matching strategy. Because the traced layers per evaluated model have different sizes, the total amount of used memory is different between both models. For every strategy, *Layer summation* and *maximum extraction* use the least amount of memory. With 2.7 G data points being necessary, *k nearest neighbor* for `ShuffleNetV2` in combination with *zero padding* or *up-scaling* uses the most memory. Classificators based on *machine learning models* take advantage of not needing class examples, saving parameters instead. This results in a memory usage lower than *k-nearest-neighbor* and *Mahalanobis distance* based classificators, only classificators (1) - (3) use less memory.

5.2. Detecting fault origin

EFFECT is used in a second experiment to find fitting classifiers for detecting fault origin, again without the need for altering or retraining the networks. The samples are split into four different classes, *normal*, *out of distribution*, *adversarial* as well as *white noise*. Correctly classified input images from ImageNet's validation dataset [14] are called *normal*, traces generated from ImageNet-o [8] are called *out of distribution*, ImageNet-a samples [8] are called *adversarial* and traces generated from white noise are called *white noise*. Thus, the used dataset is the same, only abnormal samples are now split into three classes instead of one.

Accuracy: When comparing Table 3-A with Table 3-B, classification accuracy decreases. While some strategies have a large overall reduction in accuracy for both traced networks (*Euclidean distance*), other strategies still achieve acceptable performance for one network (*Mahalanobis distance*). The confusion between adversarial samples from ImageNet-a, ImageNet-o and samples from ImageNet is the high for all strategies, but was irrelevant for the previous use case. Detecting out of distribution samples from ImageNet-o has less confusion with *normal* samples. The least confusion is in regard to white noise. For every classification strategy, the confusion with white noise is lowest.

Execution time: Figure 3-B shows the used CUDA time for fault origin detection. Execution time doubled for conventional methods, namely (1) - (3) and (5), because the necessary amount of calculations doubles with twice the classes. For *k nearest neighbor*, the execution time stays nearly constant, because the dataset is still 1 % of the total data. For deep learning methods, the execution time stays similar, because the dimension of a network don't scale with the amount of possible classification results.

Memory usage: Compared to the anomaly detection presented in section 5.1, the memory usage for classifications of fault origin is twice as large for classification strategies (1) - (3) and (5), *Euclidean distance*, *entropy*, *KL divergence* and *Mahalanobis distance*, for *k nearest neighbor*, the memory usage is identical, because the total amount of samples is the same, and 1 % of those are used for classification. For machine learning models, the same structure is used for anomaly detection (described in section 5.1) and fault origin detection. Other than the last fully connected layer, this results in nearly identical parameters sizes, with a deviation of only around 1 k parameters at most.

5.3. Summary

EFFECT provides the user with three aspects to choose the right abnormal AI operation detection method. Table 3, Table 4, as well as fig. 3 are generated using the framework. For both use cases, all compatible combinations between *dimension matching* and *classification* are evaluated. Section 5.1 shows the newly introduced method (b,5) - *Mahalanobis distance* with *maximum extraction* - as the best method for anomaly detection in `ShuffleNetV2`. For `MobileNet` anomaly detection using (c,6) and (c,7) *CNN*s with either *upsclaing* or *clipping* has the highest accuracy.

In fault origin detection (section 5.2), the same strategies show the highest accuracy. Previously shown methods still show high accuracy, but couldn't beat the presented combinations. *Euclidean distance* is always the fastest method and uses the least `float32` data points. All in all EFFECT shows, though highly use case dependent in accuracy, the overall trends regarding accuracy, memory and runtime hold true between use cases.

Table 3. Classification accuracy for MobileNet (M) and ShuffleNetV2 (S) in percent for use cases A and B over all classes, dark gray mark incompatible combinations, X mark inability to differentiate any samples

Dimension matching	Traced model	A: Anomaly detection (1)	(2)	(3)	(4)	(5)	(6)	(7)	B: Fault origin detection (1)	(2)	(3)	(4)	(5)	(6)	(7)
(a) Layer summation	M	72[b]	X	40	69	78[a]		X[c]	62[b]	29	39	65	63[a]		X[c]
	S	71[b]	56	57	65	87[a]		X[c]	58[b]	42	42	65	70[a]		X[c]
(b) Max. extraction	M	76	X	71	74	80		70[c]	64	X	50	68	58		74[c]
	S	73	X	36	74	93		89[c]	66	X	36	67	86		80[c]
(c) Up-scaling	M	85	65	67	75		90		78	65	47	67		84	
	S	78	53	57	38		71		67	43	53	33		65	
(d) Clip-ping	M	85	65	67	74		90		78	65	48	63		84	
	S	72	52	54	36		54		59	42	46	32		53	
(e) Zero padding	M	76	X	38	35		55		67	X	38	X		35	
	S	71	58	66	35		X		59	43	52	29		X	

Combinations not found previously in literature, except for: (a) partially in [9]; (b) in [17] and [1]; (c) in [16]

Table 4. Data points for dimension matching strategy with used classification for use cases A and B, dark cells mark incompatible combinations

Dimension matching	Traced model	A: Anomaly detection (1) - (3)	(4)	(5)	(6)	(7)	B: Fault origin detection (1) - (3)	(4)	(5)	(6)	(7)
(a) Layer summation	M	192[b]	19 k	36 k[a]		34 k[c]	384[b]	19 k	73 k[a]		34 k[c]
	S	2096[b]	221 k	2 M[a]		225 k[c]	4138[b]	221 k	4 M[a]		225 k[c]
(b) Max. extraction	M	192	19 k	36 k		34 k[c]	384	19 k	73 k		34 k[c]
	S	2096	221 k	2 M		225 k[c]	4138	221 k	4 M		225 k[c]
(c) Up-scaling	M	2 M	243 M		138 k		4 M	243 M		139 k	
	S	26 M	2 G		418 k		52 M	2 G		418 k	
(d) Clip-ping	M	37 k	3 M		15 k		75 k	3 M		15 k	
	S	102 k	10.4 M		158 k		205 k	10 M		158 k	
(e) Zero padding	M	2 M	243 M		138 k		4 M	243 M		139 k	
	S	26 M	2.7 G		418 k		52 M	2 G		418 k	

Combinations not found previously in literature, except for: (a) partially in [9]; (b) in [17] and [1]; (c) in [16]

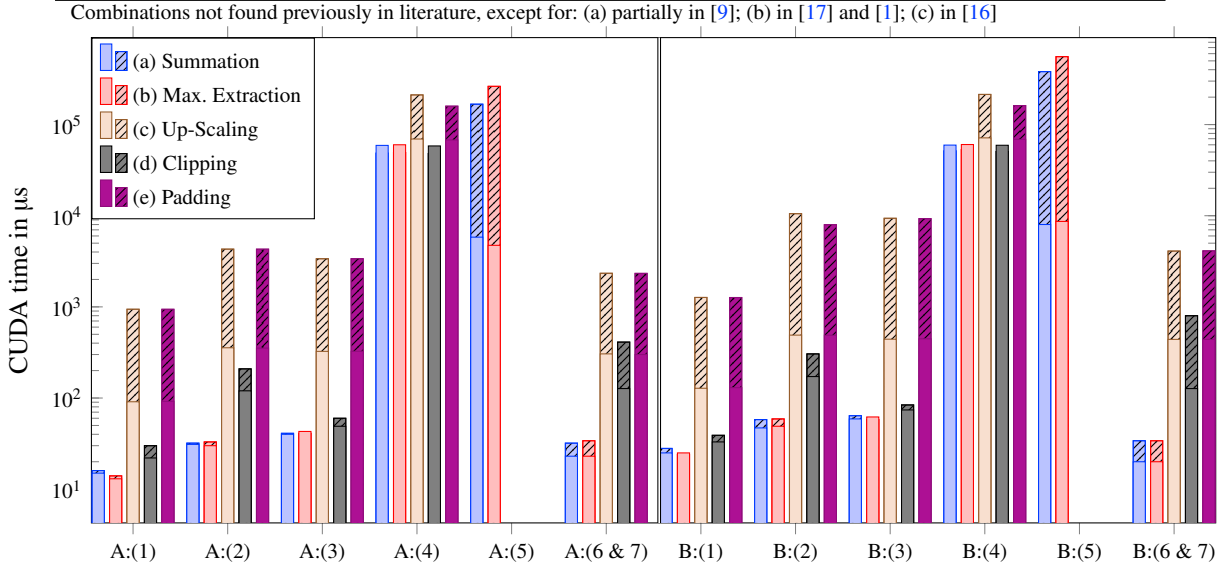

Fig. 3. Graph showing execution time for different classification approaches clustered by matching strategy, for use cases A and B. Y axis is logarithmic. The last bars methods include (6) and (7), dense net (for dimension matching (a) - (b)) and CNN based (for dimension matching (c) - (d)) approaches.

6. Conclusion

This paper presents EFFECT, an end-to-end framework capable of evaluating multiple combinations of *dimension matching* and *classification* strategies for anomaly detection, without the need for altering or retraining existing methods. The detection is based on inference traces using previously defined layers. The capabilities of this framework are demonstrated by evaluating two use cases on two machine learning models, anomaly detection and fault origin detection are tested `MobileNet` and `ShuffleNetV2`. For this, five *dimension matching* strategies and seven *classification* strategies are combined and the accuracy, CUDA time and memory consumption for each combination is presented. EFFECT is then used to find and configure the best combination for both neural network models out of existing strategies and newly introduced methods. For previously introduced methods, the performance could be confirmed, but for `MobileNet`, *CNN based classification* methods yield the highest accuracy in combination with *upscaling* or *clipping* of the generated traces. For `ShuffleNetV2`, the *Mahalanobis distance* yields the highest accuracy. By implementing our own *dimension matching* and *classification* strategies separately, it is easy to test and evaluate new strategies for inference classification based on layer tracing and comparison to existing strategies. EFFECT can thus be used to provide an overview of existing methods and present new methods in standardized ways.

References

[1] Cheng, C.H., Nührenberg, G., Yasuoka, H., 2019. Runtime monitoring neuron activation patterns, in: 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 300–303. doi:10.23919/DATE.2019.8714971.

[2] Chuan Guo, Geoff Pleiss, Yu Sun, Kilian Q. Weinberger, 2017. On calibration of modern neural networks. International Conference on Machine Learning , 1321–1330URL: http://proceedings.mlr.press/v70/guo17a.html.

[3] Csiszar, I., 1975. *I*-Divergence Geometry of Probability Distributions and Minimization Problems. The Annals of Probability 3, 146 – 158. doi:10.1214/aop/1176996454.

[4] Elfwing, S., Uchibe, E., Doya, K., 2018. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. Neural Networks 107, 3–11.

[5] Goodfellow, I.J., Shlens, J., Szegedy, C., . Explaining and harnessing adversarial examples. URL: https://arxiv.org/pdf/1412.6572.

[6] Guo, K., et al., 2019. [dl] a survey of fpga-based neural network inference accelerators. ACM Trans. Reconfigurable Technol. Syst. 12. URL: https://doi.org/10.1145/3289185, doi:10.1145/3289185.

[7] Hendrik Metzen, J., Chaithanya Kumar, M., Brox, T., Fischer, V., 2017. Universal adversarial perturbations against semantic image segmentation, in: Proceedings of the IEEE international conference on computer vision, pp. 2755–2764.

[8] Hendrycks, D., Zhao, K., Basart, S., Steinhardt, J., Song, D., 2021. Natural adversarial examples. CVPR .

[9] Lee, K., Lee, K., Lee, H., Shin, J., 2018. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. Advances in neural information processing systems 31.

[10] Liang, S., Li, Y., Srikant, R., . Enhancing the reliability of out-of-distribution image detection in neural networks. URL: https://arxiv.org/pdf/1706.02690.

[11] Ma, N., Zhang, X., Zheng, H.T., Sun, J., 2018. Shufflenet v2: Practical guidelines for efficient cnn architecture design, in: Proceedings of the European conference on computer vision (ECCV), pp. 116–131.

[12] Mahalanobis, P.C., 1936. On the generalized distance in statistics, National Institute of Science of India.

[13] Masing, L., et al., 2022. Xandar: Exploiting the x-by-construction paradigm in model-based development of safety-critical systems, in: 2022 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 1–5. doi:10.23919/DATE54114.2022.9774534.

[14] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L., 2015. ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV) 115, 211–252. doi:10.1007/s11263-015-0816-y.

[15] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C., 2018. Mobilenetv2: Inverted residuals and linear bottlenecks, in: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 4510–4520. doi:10.1109/CVPR.2018.00474.

[16] Schorn, C., Gauerhof, L., 2020. Facer: A universal framework for detecting anomalous operation of deep neural networks, in: 2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC), pp. 1–6. doi:10.1109/ITSC45102.2020.9294226.

[17] Schorn, C., Guntoro, A., Ascheid, G., 2018. Efficient on-line error detection and mitigation for deep neural network accelerators, in: Gallina, B., Skavhaug, A., Bitsch, F. (Eds.), Computer Safety, Reliability, and Security, Springer International Publishing, Cham. pp. 205–219.

[18] Stang, M., Grimm, D., Gaiser, M., Sax, E., 2020. Evaluation of deep reinforcement learning algorithms for autonomous driving, in: 2020 IEEE Intelligent Vehicles Symposium (IV), pp. 1576–1582. doi:10.1109/IV47402.2020.9304792.

[19] Xie, C., Wu, Y., Maaten, L.v.d., Yuille, A.L., He, K., 2019. Feature denoising for improving adversarial robustness, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 501–509.