Evaluation of NAP-based approaches for monitoring Neural Network classifiers

Islem Touati[1] and Abderraouf Boussif[1]

***Abstract*— This paper presents and evaluates three variant approaches based on the concept of Neural Activation Patterns (NAP) for monitoring neural network (NN) classifiers with both in-distribution (ID) and out-of-distribution (OOD) inputs. The proposed approaches aim to develop monitoring systems to supervise, in real time, the internal behavior of NN image classifiers in order to detect potential anomalies and misclassifications. The approaches rely on the NAP concept, which represents the activation states (i.e., active or inactive) of neurons within an NN in response to a given input or a set of inputs. Two benchmark datasets, MNIST and CIFAR-10, are used for the experimental evaluation under two settings: nominal (i.e., ID) setting and OOD setting, using Fashion-MNIST and SVHN as OOD datasets, respectively. The results provide a comparative assessment of the effectiveness of the proposed monitoring techniques and highlight certain limitations of NAP-based approaches for NN monitoring.**

I. INTRODUCTION

The evaluation and verification of neural networks (NN), as part of their safe design and deployment, have become a major research topic, especially with the recent studies showing their sensitivity and vulnerability to operational conditions (adversarial attacks, environmental conditions, etc.) [1], [2], [3]. Testing, formal verification, and runtime monitoring are three key verification activities for assuring the safe and reliable deployment of NN systems [1], [4]. While testing and formal verification are performed during the system design phase, monitoring (i.e., runtime verification) is an ongoing runtime activity performed during the system operational phase. It involves real-time tracking of inputs, performances, and/or the behavior of the model [5].

In recent years, the monitoring of NN systems has received significant attention from both the academic community and practitioners across a wide range of application domains. This growing interest has produced several surveys that review existing works from different perspectives, including key challenges, monitoring objectives, techniques, and targeted applications [6], [7]. One axis of research in NN monitoring focuses on designing monitors as independent components that operate in parallel with the NN model. These approaches leverage different sources of information (i.e., inputs, outputs, and internal states or behaviors of the network) either individually or in combination, to detect anomalies and assess model reliability.

For NN, the idea is to build a monitor that captures some of the features learned by the network. Then, this monitor is used to continuously observe and supervise the network and compare its behavior against the expected behavior. It is designed to detect deviations, anomalies, or any degradation in performance that may indicate a malfunction or a compromise of the intended behavior. Additionally, it can be particularly useful for detecting out-of-distribution (OOD) inputs, for which the network was not trained, which may yield erroneous results [8], [9].

The approaches studied in this paper fall within the scope of methods that exploit the internal states and behaviors of neural networks, specifically through the use of neural activation patterns (NAPs) [5], [10], [11] to build monitoring mechanisms. Three variants are considered, corresponding to the approaches NAP, NAP$_L$, and NAPath [12], [9]. A NAP represents the activation/inactivation states of neurons in response to a given input or a set of inputs, indicating which neurons in the hidden layers are active or inactive [13]. NAP$_L$ is a simplified variant that focuses solely on the neurons of the last layer, based on the assumption that this layer has the most direct influence on the network's decision. In contrast, NAPath extends the NAP concept by capturing neural paths across layers, i.e., inter-layer activation patterns [9], [2]. Rather than considering neurons independently within the same layer, NAPath preserves structural relationships between neurons across successive layers, thereby encoding richer information about the network's internal behavior.

In this paper, we evaluate these three approaches on two benchmark datasets, MNIST and CIFAR-10, under both nominal (ID) and out-of-distribution (OOD) detection settings. The paper is organized as follows. Section II provides preliminary concepts on NN monitoring, Section III discusses the NAP-based approaches for NN monitoring, Section IV presents and discusses the experimental results, and finally, Section V provides some concluding remarks.

II. NN MONITORING CONCEPTS

In this section, we present the preliminary definitions and concepts related to neural networks and their monitoring along with the associated notations and formulas.

A. Neural Networks

In this paper, we consider CNNs with ReLU activation functions for image classification [14]. Typically, a CNN contains 3 types of layers, Convolutional (*for extracting local spatial features*), Pooling (*for reducing spatial dimensions*), and fully connected (*for mapping features to class scores*). Formally, a CNN classifier is defined as $N : \mathbb{D}_x \rightarrow \mathbb{R}^m$, where $\mathbb{D}_x$ denotes the input space of images and m is

[1]Univ. Gustave Eiffel, COSYS-ESTAS, 20 rue Élisée Reclus, F-59666, Villeneuve d'Ascq, France firstname.lastname@univ-eiffel.fr

the number of classes. For an input image $x \in \mathbb{D}_x$, the network computes a sequence of intermediate representations $v_0(x), v_1(x), \ldots, v_L(x)$, where $v_0(x) = x$ is the input and $v_L(x) = N(x)$ is the final output. Each layer applies a transformation of the form: $v_i(x) = \tau_i(v_{i-1}(x))$, where τ_i denotes the operation of layer i. For ReLU-layers, this takes the form $\tau_i(v) = \sigma(g_i(v))$, where g_i denotes an affine or convolutional transformation (e.g., $g_i(v) = W_i v + b_i$ for fully connected layers or $g_i(v) = W_i * v + b_i$ for convolutional layers), and σ is the ReLU function defined as: $\sigma(z) = \max(z, 0)$. For pooling layers, no activation function is applied, and τ_i reduces to the pooling operation alone.

For an image classification problem with m classes, the network outputs a score vector: $N(x) = (N_1(x), \ldots, N_m(x)) \in \mathbb{R}^m$, where $N_i(x)$ denotes the score associated with class i. The predicted class is then defined as: $c = \arg\max_{1 \le i \le m} N_i(x)$. When there is no ambiguity, we write $N(x) = c$.

B. NN Monitoring

The runtime monitoring task relies on two main phases: (*i*) building the monitor offline, and (*ii*) using the monitor in real-time.

1) **Building the monitor (offline):** this phase is performed during the development of the monitoring system. It consists in defining properties and measurement metrics that are used to implement the core algorithms of the monitor, enabling it to observe and analyze the runtime behavior of the system.
2) **Using the monitor in real-time (online):** after building and validating the monitor, it is deployed in real-time to observe and analyze the behavior of the system. The monitor runs in parallel with the system, raising alarms for potential misbehaviors, erroneous decisions, or performance degradation.

Figure 1 illustrates the general monitoring process, including the two phases described above. In some cases, additional information may be required to build the monitor, such as details about the architecture or the internal state of the system. The output of the monitor can be a confirmation that the system behavior meets the defined requirements, an alarm signaling a misbehavior, or feedback to the system and its controller suggesting possible improvements.

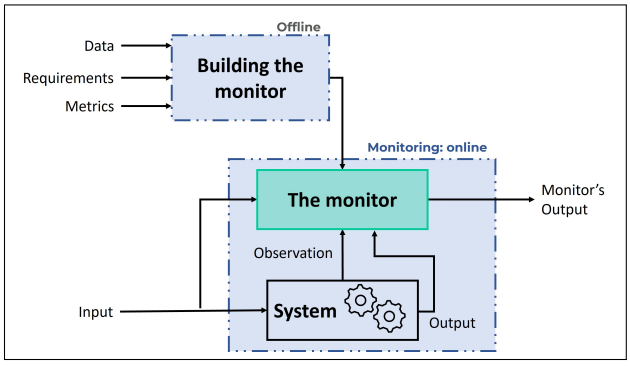

Fig. 1: General process of runtime monitoring.

In the context of NN systems, the monitor building phase relies on extracting learned patterns from the NN model using its training dataset. The monitor, represented by these patterns, is then used to supervise the network decisions at runtime. This phase is essential, as it determines the quality of the extracted patterns that constitute the core of the monitor. Although it is hard to formally prove that these patterns exactly reflect what the network has learned, empirical guarantees are sought by assessing the monitoring system on diverse datasets and scenarios [5].

Most NN monitoring approaches seek to analyze the internal activations of the network. The general idea is to feed the network N with a set of labeled data, e.g., the training set, and to record the values of the hidden neurons for each input. These values are referred to as *hidden neuron activations*. The monitor is then constructed from these activations, where each set of activations encodes a representative pattern for a given class [13], [5], [15]. Hereafter, we present NN monitoring approaches based on the concept of *Neural Activation Patterns*.

III. NAP-based monitoring

A. Neuron Activation Patterns (NAP)

A NAP is a representation of the activation levels of neurons in a neural network in response to a given input or a set of inputs, indicating which neurons (from the hidden layers) are active or inactive [13]. This concept is easier to illustrate in the case of the *ReLU* function, since it is a piece-wise linear function (with two linear regions).

Definition 1: Let N be a network with *ReLU* (denoted ReLU-NN), and let S be the set of its hidden neurons. The set of all active (resp. inactive) neurons on the network N, for an input x, is denoted as A (resp. D) and defined such that:

$$\begin{cases} A = \{n \in S : n(x) > 0\} \\ D = \{n \in S : n(x) = 0\} \end{cases} \quad (1)$$

In definition 1, $n(x)$ denotes the output value of a neuron $n \in S$ when the network has x as input. For a *ReLU*-NN, a NAP of an input x on N (denoted $NAP(x, N)$) is defined as the set of its corresponding active and inactive neurons, i.e., $NAP(x, N) = (A, D)$.

For a set of inputs X_c such that $\forall x \in X_c : N(x) = c$, we define its associated *NAP* as the common NAP between all inputs:

$$\begin{cases} A_c = \{n \in S : \forall x \in X_c, n(x) > 0\} \\ D_c = \{n \in S : \forall x \in X_c, n(x) = 0\} \end{cases} \quad (2)$$

For a set X_c considered as a representative set of inputs of class c, then its NAP computed using (2) represents the NAP of this class, which is denoted as $NAP_c = (A_c, D_c)$.

B. Neuron Activation Patterns Last Layer (NAP_L)

A simplified variant of NAP for monitoring consists in focusing only on the last hidden layer of the network (since it is supposed to be the most impactful layer on the decision of the network). This variant, denoted NAP_L, follows the same construction principle as NAP, with a restriction to the

set of neurons on the last hidden layer. This simplification aims to reduce the computation effort for building and storing the monitored representation while still capturing high-level features relevant for a classification task.

C. Neuron Activation Paths (NAPath)

An interesting extension of NAPs has recently been introduced for the formal verification of neural networks. It consists in exploiting the neural paths across layers (i.e. *inter-layer patterns*) instead of neurons with the same layer (i.e., *intra-layer patterns*). Hence, while NAP captures which neurons are active or inactive independently, NAPath encodes structural information by preserving the relationships between neurons across successive layers.

A neural path $P = \{n_1, \ldots, n_L\}$ is a sequence of neurons such that each neuron n_k belongs to layer l_k for $1 \leq k \leq L$. For a given input x, we distinguish two types of paths:

- *Active path:* all neurons in P satisfy $n_k(x) > 0$.
- *Inactive path:* all neurons in P satisfy $n_k(x) = 0$.

Compared to NAP, NAPath offers a stricter representation of the network patterns by capturing interlayer dependencies, which can lead to a more discriminative characterization of the internal decision process.

Definition 2: A NAPath of an input x using a network N, denoted as $NAPath(x,N)$, is the tuple $(\mathscr{A}, \mathscr{D})$ where $\mathscr{A}$ is the set of all active paths and $\mathscr{D}$ is the set of all inactive paths. Formally, we write: $NAPath(x,N) = (\mathscr{A}, \mathscr{D})$.

To maintain consistency with the definition of NAPs, in Definition 2 we used the notation $\mathscr{A}$ and $\mathscr{D}$ to denote the set of active and inactive paths, respectively. When there is no ambiguity, we omit the argument N and write $NAPath(x) = NAPath_x = (\mathscr{A}, \mathscr{D})$.

Remark 1: It is worth noting that the NAP of an individual input x, denoted as $NAP(x,N) = (A,D)$, is a partition of the set of all neurons S of the network N, such that $A \cap D = \emptyset$ and $A \cup D = S$. However, the NAP of a class c generated using a set of images and denoted as $NAP_c = (A_c, D_c)$ only forms a subset of S, i.e., $A_c \cup D_c \subseteq S$. This is due to the fact that NAP_c is an intersection of the NAPs of the selected images. On the other hand, a NAPath of the same input is a subset of paths of N, and consequently, the set of participating neurons (the neurons in $\mathscr{A}$ and $\mathscr{D}$) is a subset of S. This is due to the fact that some paths of the network may be neither active nor inactive.

Illustrative Example: Figure 2 illustrates the three NAP-based concepts on a small network with two hidden layers. l_1 contains neurons n_{11}, n_{12}, and n_{13}; and l_2 contains n_{21}, and n_{22}. For input x, active and inactive neurons are in green and red respectively. The activation patterns are :

- $NAP(x,N) = \big(A, \quad D\big), \quad A = \{n_{11}, n_{13}, n_{21}\}, \quad D = \{n_{12}, n_{22}\}$
- $NAP_L(x,N) = \big(A_L, \ D_L\big), \quad A_L = \{n_{21}\}, \quad D_L = \{n_{22}\}$
- $NAPath(x,N) = \big(\mathscr{A}, \ \mathscr{D}\big), \mathscr{A} = \{(n_{11}, n_{21}), \ (n_{13}, n_{21})\}, \mathscr{D} = \{(n_{12}, n_{22})\}$

D. Monitoring using NAPs and NAPaths

The proposed monitoring approaches follow the general two-phase process (See Fig 1). In the offline phase, class-wise reference patterns are extracted from the training data (i.e., NAPs, NAP_L and NAPaths). In the online phase, these patterns are used during the runtime execution of the NN models with real inputs. For clarity and brevity, we explain here below the process for the NAPath-based approach, as depicted in Fig. 3; the process remains similar for NAP and NAP_L-based approaches.

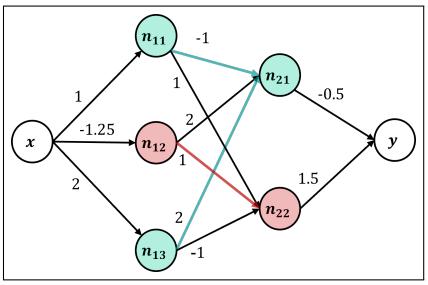

Fig. 2: Illustrative example of NAPs concepts.

1) Offline Phase: Given a neural network N and a labeled dataset X, the training samples are partitioned into subsets X_c for each class c. For each class, a reference NAPath is extracted from the hidden-layer activations. A threshold parameter $\delta \in [0,1]$ controls the selectivity of the construction: a path is considered active (resp. inactive) if it is activated (resp. deactivated) in at least $\delta \times |X_c|$ samples. The resulting class-wise reference patterns, denoted $NAPath_c^{\delta}$, are stored and later used by the monitor.

2) Online Phase: At inference time, the NN monitor is executed in parallel with the network N to supervise its decisions in real-time. In addition to the stored precomputed NAPaths, the monitor receives in real-time the input image x with its corresponding output y issued by the NN, i.e.: $N(x) = y$, and the NAPath ($NAPath_x$) computed on-the-fly. Based on this information, the monitor evaluates the similarity between $NAPath_x$ and each pre-computed reference pattern $NAPath_c = (\mathscr{A}_c, \mathscr{D}_c)$ as follows:

$$Sim(NAPath_c, NAPath_x) = p \cdot \frac{|\mathscr{A}_c \cap \mathscr{A}_x|}{|\mathscr{A}_c|} + (1-p) \cdot \frac{|\mathscr{D}_c \cap \mathscr{D}_x|}{|\mathscr{D}_c|} \tag{3}$$

where $p \in [0,1]$ weights the relative importance of active and inactive paths. If several classes obtain the same maximum similarity, the following noise measure is minimized:

$$Noise(NAPath_c, NAPath_x) = p \cdot \frac{|\mathscr{A}_x| - |\mathscr{A}_c \cap \mathscr{A}_x|}{N_{paths} - |\mathscr{A}_c|} + (1-p) \cdot \frac{|\mathscr{D}_x| - |\mathscr{D}_c \cap \mathscr{D}_x|}{N_{paths} - |\mathscr{D}_c|} \tag{4}$$

where N_{paths} is the total number of paths in the network. The predicted class is then the one whose reference NAPath yields the highest similarity score, with the noise measure used as a tiebreaker.

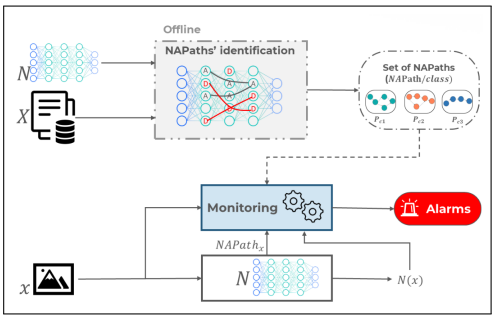

Fig. 3: NAPath monitoring process

E. ID and OOD monitoring

During the runtime monitoring, the NN classifiers (and their monitors) face two kinds of inputs: i) inputs that belong to the distribution on which the NN models were trained *i.e., in-distribution inputs – ID*, and ii) inputs that do not belong to this distribution i.e., *out-of-distribution inputs – OOD*.

1) ID monitoring: For ID monitoring, the monitor's task is to supervise the outputs of the NN classifier and raise an alarm when the output is inconsistent with the reference activation pattern. If the class c yielding the highest similarity differs from the predicted class y, the monitor raises an alarm signaling a potential misclassification.

2) OOD monitoring: OOD monitoring introduces an additional task, which consists in determining whether the input belongs to the considered distribution or not. Only if the input belongs to ID that the monitor proceeds to assess whether the NN classifier's decision is consistent with the reference.

For the evaluation, we consider that for each correctly classified image in the ID test set, two scores are extracted: the maximum similarity Sim_{max} with the class-wise reference NAPaths, and the associated noise level $Noise_{min}$. These scores are then used to build two distributions, from which decision thresholds are derived using percentiles:

- $Sim_{threshold}$ is set at the 5^{th} percentile of the similarity distribution, ensuring that 95% of ID inputs satisfy $Sim_{max} \geq Sim_{threshold}$,
- $Noise_{threshold}$ is set at the 95^{th} percentile of the noise distribution, ensuring that 95% of ID inputs satisfy $Noise_{min} \leq Noise_{threshold}$.

An input x is then classified as ID if:

$$Sim_{max} \geq Sim_{threshold} \quad \text{and} \quad Noise_{min} \leq Noise_{threshold}$$

and OOD otherwise. Intuitively, an ID input tends to produce a high similarity and a low noise level with respect to the reference NAPaths, while an OOD input deviates from these expected patterns.

F. Evaluation Metrics

The monitoring performances are evaluated by considering the numbers of *correct alarms* (TP), *correct no-alarms* (TN), *false alarms* (FP), and *missed alarms* (FN), and their corresponding rates.

$$TPR = \frac{TP}{TP+FN}; \quad FNR = \frac{FN}{TP+FN}; \quad FPR = \frac{FP}{TN+FP}$$

$$TNR = \frac{TN}{TN+FP}; \quad \text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

A typical monitor should maximize TNR and TPR, while keeping FPR and FNR as low as possible.

For the OOD setting, the metrics are defined as follows:

- TP (FN): OOD inputs correctly detected as OOD (incorrectly accepted as ID).
- TN : ID inputs correctly accepted as ID and correctly classified.
- FP: ID inputs either rejected as OOD or accepted as ID but misclassified. Formally: FP = ID rejected as OOD + ID misclassified.

Additionally, we consider the following performance metrics for the evaluation of OOD setting: Accuracy, Precision, Recall, F1-score, Specificity, False Positive Rate (FPR), Negative Predictive Value (NPV), Balanced Accuracy, and Matthews Correlation Coefficient (MCC).

IV. Experiments

A. Benchmark Datasets and Models

1) Datasets: Two datasets are used, MNIST [16] and CIFAR10 [17]. MNIST consists of 60,000 training images and 10,000 test images of handwritten digits (0–9), i.e., 10 classes. Each image is grayscale with a resolution of 28×28 pixels. CIFAR-10 consists of 60,000 color images of 32×32 pixels, divided into 50,000 training images and 10,000 test images across 10 classes. Compared with MNIST, CIFAR-10 exhibits higher visual complexity, larger intra-class variability, and stronger inter-class similarity.

For OOD monitoring, we use Fashion-MNISTas the OOD dataset of MNIST. It consists of 10,000 grayscale images of 28×28 pixels representing clothing items, sharing the same format as MNIST but with a fundamentally different content distribution. For CIFAR10, we use SVHNas the OOD dataset. It consists of color images of street-view house numbers, sharing the same $32 \times 32 \times 3$ format as CIFAR-10 but drawn from a different real-world distribution.

2) CNN Models: For the MNIST dataset, we employ a CNN classifier whose architecture consists of two convolutional blocks (32 and 64 filters, 3×3, each followed by ReLU and 2×2 max-pooling), followed by two fully connected layers (512 and 128 neurons, ReLU). The final layer has 10 outputs. The model achieves 99.17% accuracy on the MNIST test set. For the CIFAR-10 dataset, we use a VGG-based [18] CNN, with multiple convolutional blocks with 3×3 kernels and increasing numbers of filters (64, 128, 256, and 512), each followed by ReLU activations and max-pooling layers. The model achieves 92.17% accuracy on the CIFAR-10 test set.

B. Experimental Protocol

1) Offline Phase: For each approach the monitor is built from the training set by extracting the features (NAPs, NAP_L, and NAPaths) associated with each class, from the hidden-layer activations of the trained CNN. For the *ID setting*, these features are generated with respect to different values of the

controlling parameter $\delta \in \{0.80, 0.85, 0.90, 0.95, 1\}$. For the OOD setting, the value of δ, which corresponds to the best performance, is used.

2) Runtime Phase: The generated monitors are then used to evaluate the classification decisions of the models based on the specified test set of each benchmark. For the ID setting, various values of the monitoring parameter p are considered: $p \in \{0.0, 0.1, \cdots, 0.8, 0.9, 1.0\}$. For the OOD setting, the best configurations identified in the OOD setting are considered.

C. Experimental Results

Figures 4 and 5 depict the accuracy of the monitors on MNIST and CIFAR10, respectively. On MNIST, all three methods quickly reach high accuracy around $\sim 99\%$ when $p > 0.2$, indicating that the approach performs well on simpler tasks. On CIFAR-10, the results are more sensitive to the value of p, and accuracy drops when p increases. NAP$_L$ performs better on CIFAR-10 (keeping $\sim 91.95\%$ accuracy), while NAP shows more variation.

Tables I and II provide the simulation results for MNIST and CIFAR respectively with ID distribution.

MNIST Results (Table I): All three methods achieve high TNR ($\sim 99.5\%$) and maintain false alarm rates below 1%, demonstrating reliable normal case detection. However, anomaly detection performance is limited, with TPR values between 30-40%. Increasing δ improves performance by reducing false alarms and enhancing detection rates, while p has minimal impact. NAPath outperforms NAP and NAP$_L$, reaching TPR values up to 46% compared to 37-41% for the others. NAP shows consistent improvement with higher δ, while NAP$_L$ exhibits lower overall robustness despite stable normal case detection.

CIFAR-10 Results (Table II): Similar to MNIST, all methods maintain high TNR ($\sim 99.3 - 99.7\%$) and low false alarm rates ($< 1\%$). However, CIFAR-10 presents greater anomaly detection challenges, with TPR values in the 4-9% range, reflecting dataset complexity. NAP$_L$ demonstrates superior performance with marginally higher and more stable TPR across parameter variations. Increasing p reduces alarm counts while maintaining detection capability, with the sparsest configuration ($p = 0.8$) achieving the lowest false alarm counts. NAPath shows competitive efficiency with NAP while requiring fewer total alarms.

Overall: Both datasets confirm that all methods reliably identify normal cases but struggle with anomaly detection, particularly on complex datasets. The choice between methods depends on the specific application: NAPath excels on simpler tasks (MNIST), while NAP$_L$ provides better stability on complex tasks (CIFAR-10).

Table III presents the OOD setting results for MNIST/ Fashion-MNIST and CIFAR-10/ SVHN. On MNIST/ Fashion-MNIST, NAP and NAP$_L$ perform strongly: NAP$_L$ achieves the best results with 93% accuracy, detecting 9,302 out of 10,000 OOD samples and an MCC of 0.87, while NAP achieves similar performance (92% accuracy, MCC of 0.84). NAPath, however, lags behind with only 7,548 OOD detections, a recall of 75%, and an MCC of 0.69, despite maintaining good precision (91%). On CIFAR-10/SVHN, detection becomes harder for all methods: NAP$_L$ still leads with 74% accuracy and 18,068 detections (MCC of 0.50), while NAP and NAPath achieve lower recall (64–67%) and MCC scores of 0.44–0.50, reflecting the increased dataset complexity. Across both datasets, NAP$_L$ consistently outperforms the other methods, while NAPath shows weaker anomaly detection despite competitive precision. This relatively weaker performance of NAPath can be attributed to the higher structural rigidity of its representations: by requiring entire paths to be simultaneously active or inactive, NAPath captures a stricter and more sparse set of patterns. This may lead to an overfitting of the reference patterns extracted from the training data and reduce its ability to generalize to unseen ID inputs, and even more so to OOD.

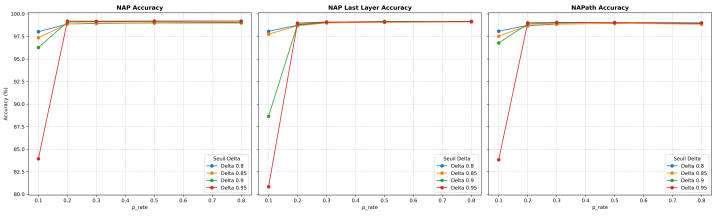

Fig. 4: Accuracy on MNIST w.r.t. p and δ.

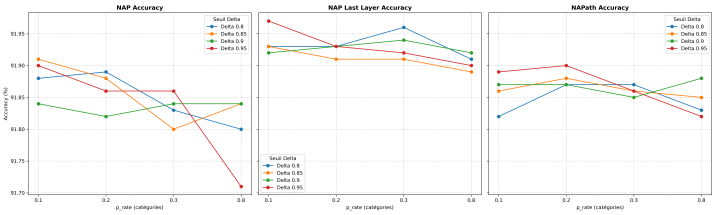

Fig. 5: Accuracy on CIFAR-10 w.r.t. p and δ.

V. CONCLUSIONS

This paper assessed three NAP-based approaches for monitoring NN classifiers under both ID and OOD settings. The results show that NAPs can help detect anomalies and misclassifications by capturing internal network behavior; however, their effectiveness remains limited, especially for complex OOD images. These findings provide a better understanding of the strengths and limitations of internal-state-based monitoring approaches for NN classifiers and highlight the need for complementary methods to improve the robustness and reliability of NN monitoring systems.

REFERENCES

[1] R. Hawkins, C. Paterson, C. Picardi, Y. Jia, R. Calinescu, and I. Habli, "Guidance on the assurance of machine learning in autonomous systems (amlas)," *arXiv preprint arXiv:2102.01564*, 2021.

[2] F. Boudardara, A. Boussif, P.-J. Meyer, and M. Ghazel, "Innabstract: An inn-based abstraction method for large-scale neural network verification," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 35, no. 12, pp. 18 455–18 469, 2023.

[3] ——, "A review of abstraction methods toward verifying neural networks," *ACM Transactions on Embedded Computing Systems*, vol. 23, no. 4, pp. 1–19, 2024.

[4] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

δ	p	NAP					NAP_L					NAPath				
		Alarms	TNR (%)	TPR (%)	FNR (%)	FP (%)	Alarms	TNR (%)	TPR (%)	FNR (%)	FP (%)	Alarms	TNR (%)	TPR (%)	FNR (%)	FP (%)
0.80	0.1	90	99.4	37.3	62.7	0.6	97	99.3	33.7	66.3	0.7	104	99.3	36.1	63.9	0.7
0.80	0.2	83	99.5	36.1	63.9	0.5	72	99.6	37.3	62.7	0.4	86	99.4	34.9	65.1	0.6
0.80	0.3	80	99.5	37.3	62.7	0.5	67	99.6	37.3	62.7	0.4	81	99.5	34.9	65.1	0.5
0.80	0.8	78	99.5	37.3	62.7	0.5	52	99.7	32.5	67.5	0.3	81	99.5	34.9	65.1	0.5
0.85	0.1	89	99.4	37.3	62.7	0.6	110	99.2	37.3	62.7	0.8	115	99.2	39.8	60.2	0.8
0.85	0.2	82	99.5	37.3	62.7	0.5	75	99.5	34.9	65.1	0.5	91	99.4	36.1	63.9	0.6
0.85	0.3	85	99.5	39.8	60.2	0.5	65	99.6	33.7	66.3	0.4	81	99.5	36.1	63.9	0.5
0.85	0.8	80	99.5	37.3	62.7	0.5	57	99.7	31.3	68.7	0.3	93	99.4	37.3	62.7	0.6
0.90	0.1	71	99.6	38.6	61.4	0.4	96	99.4	38.6	61.4	0.6	96	99.4	41.0	59.0	0.6
0.90	0.2	68	99.6	38.6	61.4	0.4	71	99.6	37.3	62.7	0.4	78	99.5	37.3	62.7	0.5
0.90	0.3	67	99.6	37.3	62.7	0.4	59	99.7	36.1	63.9	0.3	73	99.6	37.3	62.7	0.4
0.90	0.8	69	99.6	36.1	63.9	0.4	53	99.7	31.3	68.7	0.3	79	99.5	37.3	62.7	0.5
0.95	0.1	62	99.7	39.8	60.2	0.3	82	99.5	38.6	61.4	0.5	83	99.5	41.0	59.0	0.5
0.95	0.2	65	99.7	41.0	59.0	0.3	72	99.6	39.8	60.2	0.4	86	99.5	45.8	54.2	0.5
0.95	0.3	63	99.7	41.0	59.0	0.3	69	99.6	38.6	61.4	0.4	87	99.5	45.8	54.2	0.5
0.95	0.8	63	99.7	41.0	59.0	0.3	63	99.7	37.3	62.7	0.3	89	99.5	44.6	55.4	0.5

TABLE I: Evaluation results of NAP, NAP_L, and NAPath approaches for MNIST under different values of δ and p.

δ	p	NAP					NAP_L					NAPath				
		Alarms	TNR (%)	TPR (%)	FNR (%)	FP (%)	Alarms	TNR (%)	TPR (%)	FNR (%)	FP (%)	Alarms	TNR (%)	TPR (%)	FNR (%)	FP (%)
0.80	0.1	130	99.3	8.4	91.6	0.7	111	99.5	7.6	92.4	0.5	110	99.4	6.8	93.2	0.6
0.80	0.2	117	99.4	7.7	92.3	0.6	105	99.5	7.2	92.8	0.5	99	99.5	6.5	93.5	0.5
0.80	0.3	115	99.4	7.2	92.8	0.6	102	99.5	7.2	92.8	0.5	95	99.5	6.2	93.8	0.5
0.80	0.8	94	99.5	5.7	94.3	0.5	75	99.7	5.2	94.8	0.3	77	99.6	4.9	95.1	0.4
0.85	0.1	133	99.3	8.8	91.2	0.7	105	99.5	7.2	92.8	0.5	104	99.5	6.7	93.3	0.5
0.85	0.2	130	99.3	8.4	91.6	0.7	103	99.5	7.0	93.0	0.5	98	99.5	6.5	93.5	0.5
0.85	0.3	118	99.4	7.2	92.8	0.6	101	99.5	6.8	93.2	0.5	92	99.5	6.0	94.0	0.5
0.85	0.8	94	99.5	6.0	94.0	0.5	73	99.7	5.0	95.0	0.3	73	99.6	4.8	95.2	0.4
0.90	0.1	138	99.3	8.7	91.3	0.7	106	99.5	7.2	92.8	0.5	113	99.4	7.3	92.7	0.6
0.90	0.2	128	99.3	7.9	92.1	0.7	103	99.5	7.1	92.9	0.5	107	99.5	7.0	93.0	0.5
0.90	0.3	118	99.4	7.4	92.6	0.6	100	99.5	7.0	93.0	0.5	99	99.5	6.3	93.7	0.5
0.90	0.8	90	99.5	5.7	94.3	0.5	74	99.7	5.2	94.8	0.3	70	99.7	4.8	95.2	0.3
0.95	0.1	134	99.3	8.8	91.2	0.7	107	99.5	7.6	92.4	0.5	109	99.5	7.2	92.8	0.5
0.95	0.2	124	99.4	7.9	92.1	0.6	103	99.5	7.1	92.9	0.5	102	99.5	6.8	93.2	0.5
0.95	0.3	116	99.4	7.4	92.6	0.6	96	99.5	6.6	93.4	0.5	96	99.5	6.2	93.8	0.5
0.95	0.8	79	99.5	4.3	95.7	0.5	78	99.6	5.4	94.6	0.4	68	99.6	4.3	95.7	0.4

TABLE II: Evaluation results of NAP, NAP_L, and NAPath approaches for CIFAR-10 under different values of δ and p.

Méthode	δ	p	OOD			IND			Matrice de confusion				Métriques						Jeu
			Total	Détectées	Non détectées	Total	Correctes	Incorrectes	TP	TN	FP	FN	Acc.	Prec.	Rec.	F1	Spec.	MCC	
NAP	0.95	0.8	10000	9183	817	10000	9232	768	9183	9232	768	817	0.92	0.92	0.91	0.92	0.92	0.84	MNIST Fashion-MNIST
NAP_L	0.95	0.8	10000	9302	698	10000	9387	613	9302	9387	613	698	0.93	0.94	0.93	0.93	0.94	0.87	
NAPath	0.95	0.8	10000	7548	2452	10000	9267	733	7548	9267	733	2452	0.84	0.91	0.75	0.83	0.93	0.69	
NAP	0.9	0.1	26032	16731	9301	10000	8438	1562	16731	8438	1562	9301	0.70	0.91	0.64	0.75	0.84	0.44	CIFAR-10 SVHN
NAP_L	0.9	0.1	26032	18068	7964	10000	8625	1375	18068	8625	1375	7964	0.74	0.93	0.69	0.79	0.86	0.50	
NAPath	0.9	0.1	14646	9771	4875	10000	8363	1637	9771	8363	1637	4875	0.74	0.86	0.67	0.76	0.84	0.50	

TABLE III: Evaluation for OOD setting of MNIST and CIFAR10.

[5] C.-H. Cheng, G. Nührenberg, and H. Yasuoka, "Runtime monitoring neuron activation patterns," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 300–303.

[6] V. J. Hodge, C. Paterson, and I. Habli, "Out-of-distribution detection for safety assurance of ai and autonomous systems," *arXiv preprint arXiv:2510.21254*, 2025.

[7] R. S. Ferreira, J. Guérin, K. Delmas, J. Guiochet, and H. Waeselynck, "Safety monitoring of machine learning perception functions: a survey," *Computational Intelligence*, vol. 41, no. 2, p. e70032, 2025.

[8] V. Hashemi, J. Křetínskỳ, S. Rieder, and J. Schmidt, "Runtime monitoring for out-of-distribution detection in object detection neural networks," in *International Symposium on Formal Methods*. Springer, 2023, pp. 622–634.

[9] F. Boudardara, A. Boussif, P.-J. Meyer, and M. Ghazel, "Monitoring of neural network classifiers using neuron activation paths," in *International Conference on Verification and Evaluation of Computer and Communication Systems*. Springer, 2024, pp. 81–96.

[10] R. S. Ferreira, J. Guerin, J. Guiochet, and H. Waeselynck, "Sena: Similarity-based error-checking of neural activations," in *27th European Conference on Artificial Intelligence-ECAI 2023*, 2023.

[11] S. Wang, P. Wang, L. Mihaylova, and M. Hill, "Real-time activation pattern monitoring and uncertainty characterisation in image classification," in *2021 IEEE 24th International Conference on Information Fusion (FUSION)*. IEEE, 2021, pp. 1–7.

[12] F. Boudardara, "Contributions to the verification and monitoring of neural network systems," Ph.D. dissertation, Université Gustave Eiffel, 2024.

[13] C. Geng, N. Le, X. Xu, Z. Wang, A. Gurfinkel, and X. Si, "Toward reliable neural specifications," *arXiv preprint arXiv:2210.16114*, 2022.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, vol. 25, 2012, pp. 1097–1105.

[15] T. A. Henzinger, A. Lukina, and C. Schilling, "Outside the box: Abstraction-based monitoring of neural networks," in *24th European Conference on Artificial Intelligence*, 2020, pp. 2433–2440.

[16] Y. LeCun. (1998) The mnist database of handwritten digits. urlhttp://yann.lecun.com/exdb/mnist/.

[17] A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009.

[18] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.