Monitoring of Neural Network Classifiers using Neuron Activation Paths

Fateh Boudardara, Abderraouf Boussif, Pierre-Jean Meyer, Mohamed Ghazel

Monitoring of Neural Network Classifiers using Neuron Activation Paths

Fateh Boudardara[1][0000−0001−5771−7676], Abderraouf Boussif[1], Pierre-Jean Meyer[1], and Mohamed Ghazel[1]

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

Abstract. To be deployed in safety critical applications, neural network (NN) systems require to be verified during the development phase and then monitored during the runtime phase. The latter phase is essential to closely supervise the performance and the behavior of NNs, particularly when used for safety-related tasks. This paper presents a novel approach for monitoring NN classifiers through real-time supervision of the model's behavior and decisions, to detect potential anomalies. The approach is based on the concept of *Neuron Activation Paths* (NAPath), which allows extracting relevant activated/inactivated paths that link the inputs to the outputs of the network and significantly influence the NN's classification decision. The main idea is to characterize paths for each class using training data, i.e., from the training data set, we group the images of the same class together. Then, for each group of images, we identify their common active and inactive paths, respectively. The sets of active and inactive paths constitute a NAPath, which is used as a signature to feature the corresponding class. The monitoring system then uses these NAPaths online to continuously check whether the paths activated by the image fit the NAPath characterization associated with its classification, as returned by the network. The monitoring system raises alarms if an abnormal decision of the network is detected. We evaluated our approach on a benchmark of neural networks pre-trained on the MNIST data set.

Keywords: Neural network (NN) classifiers · NN monitoring · Neural activation patterns · Neural Activation paths

1 Introduction

Neural networks (NNs) are one of the most successfully used techniques in artificial intelligence-based systems [9]. The ability of NNs to generalize the input-output relationship of a training data set allows its application to tackle various complex problems, such as image classification and object detection and recognition [16,18]. Nowadays, academia and industry are showing more and more interest in applying this technique in some safety-critical domains, such as autonomous transportation systems [3,20]. Considering the safety requirements pertaining to these systems, the question of validating and certifying NN systems becomes central.

Test-based verification (testing for short), formal verification, and runtime monitoring are three key verification activities for ensuring the safe and reliable deployment of NN systems. Testing involves running the trained model on a set of data to measure its performance and validate it with respect to a set of (representative) test cases [9]. Formal verification involves using mathematical techniques to verify and provide formal guarantees that the model satisfies a set of desired properties [12,7,5,4]. While testing and formal verification are performed during the system design phase, monitoring is an ongoing runtime activity performed during the system operational phase. It involves real-time tracking of the inputs, the performances, and the behavior of the model [6].

In this paper, we propose a monitoring approach for NN image classifiers with the ReLU activation function. The proposed approach is used to supervise and control the network's outputs with respect to the input images, towards detecting misclassified images and adding a level of confidence to the NN decision. The approach relies on the concept of neuron activation pathway (NAPath), which we introduce as part of the contribution. NAPath is used to identify relevant activation paths for each class in a NN. A path is defined as a sequence of neurons of the same type of activation (active or inactive) linking the input of the NN to its output. For a NN with a *Relu* activation function, NAPath extracts learned features of the network when it processes images that belong to the same class. We define a feature as a set of paths on the network that are similarly activated or inactivated for the selected images. This work is inspired by the Neuron Activation Pattern (NAP) concept discussed in [8]. While a NAP is a tuple of two sets: active and inactive neurons, we define a NAPath as a tuple $NAPath = (\mathcal{A}, \mathcal{D})$, with:

1. $\mathcal{A}$ being the set of paths that are active for most of the selected images.
2. $\mathcal{D}$ is the set of paths that are inactive for most of the selected images.

The first step of our approach involves identifying active and inactive neurons within the neural network for a set of samples, for each class c. Specifically, a neuron is deemed active for class c if its output value is strictly positive for a considered percentage of input images of this class. Similarly, a neuron is considered as inactive if its output is equal to zero for a considered percentage of samples of this class. Once the neurons have been labeled as active or inactive, the next step involves constructing paths that are exclusively constituted of active or inactive neurons. In essence, an active (resp. inactive) path represents a sequence of active (resp. inactive) neurons linking the input layer to the output layer. These paths are then used to build the NAPath for the given set of images. The main intuition consists in assuming that the inputs belonging to the same class tend to behave similarly and activate the same paths. Therefore, a NAPath can be viewed as a feature associated with that specific class c.

We construct a set of NAPaths, where each NAPath corresponds to a specific class of images. Leveraging these NAPaths, we propose an online monitoring process for image classifiers when they are applied to new unseen images. Our NN monitor relies on the premise that an image x, whose classification output by the network N is class c (denoted as $N(x) = c$), should generate active

and inactive paths similar to the NAPath pre-computed for class c ($NAPath_c$). Therefore, for an arbitrary image x, the network's classification $N(x) = c$ and the NAPath generated by this image x in the network (denoted as $NAPath_x$), the monitor processes this data and compares them to the pre-computed NAPath associated with each class, and can raise two different types of alarms:

1. *Misclassification detection*: if $NAPath_x$ is highly similar to the NAPath associated with a class c' other than the network's classification (class c) of this input, i.e., $N(x) \neq c$, the monitor raises an alarm and suggests a re-classification of this image.
2. *Novelty detection*: if $NAPath_x$ does not share any similar paths with any of the pre-computed NAPaths, the monitor raises an alarm that this input does not fit any feature of any class; thus, it is considered as a novel (or out-of-distribution) sample.

To evaluate the effectiveness of our approach, we performed a series of experiments on neural networks trained for digit classification on the MNIST dataset [15]. The experimental results show how the proposed approach enhances efficiently and the reliability of the classification decision made by the network. The use of NAPaths for runtime monitoring provides more confident and trustworthy predictions. Based on our findings, we believe that our approach shows great promise and can pave the way for further research in the area of monitoring image classifiers.

The remaining of this paper is organized as follows: Section 2 provides definitions and notations related to NNs and introduces the NAPath concept. Section 3 presents the proposed monitoring method. The experiments' setups and the obtained results are given in Section 4. Finally, in Section 5 we provide a review of related works before concluding the paper in Section 6.

2 Preliminary concepts

2.1 Neural networks

In this paper, we consider feed-forward neural networks with *Relu* activation function, and we refer to them as neural networks (NNs). A NN is a set of interconnected neurons, also called nodes, organized in layers. The neurons of layer l_i are connected to all neurons of layer l_{i+1} via weighted edges. The weights of layer l_i are organized as a matrix $W_i \in \mathbb{R}^{|l_i| \times |l_{i-1}|}$. Each layer has a vector bias $b_i \in \mathbb{R}^{|l_i|}$. Using these notations, NNs can be seen as a function $f : \mathbb{R}^n \to \mathbb{R}^m$ that is defined recursively as:

$$\begin{cases} f(x) = v_L(x) \\ v_i(x) = W_i \times \sigma(v_{i-1}(x)) + b_i \\ v_0(x) = x \end{cases} \tag{1}$$

where L is the number of layer of N and σ is a non-linear activation function, which represents the *Relu* function $\sigma(x) = \max(x, 0)$ in this work.

For an image classification problem of m classes, f maps an input-image $x \in \mathbb{D}_x$ [1] to an output $y \in \mathbb{R}^m$, where $y_i = f_i(x)$ is the score of the i^{th} class, and the element of the highest score ($c = \underset{1 \leq i \leq m}{\text{argmax}}(f_i(x))$) is considered as the predicted class for the image x. We write $f(x) = c$ or $N(x) = c$ where there is no ambiguity.

2.2 Neuron Activation Patterns (NAP)

To provide the necessary background for our proposed approach, we first recall the concept of Neuron Activation Pattern (NAP) [8]. 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. It is easier to explain this concept 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:*

$$\begin{cases} A = \{n \in S : n(x) > 0\} \\ D = \{n \in S : n(x) = 0\} \end{cases} \tag{2}$$

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:

$$\begin{cases} A_c = \{n \in S : \forall x \in X_c, n(x) > 0\} \\ D_c = \{n \in S : \forall x \in X_c, n(x) = 0\} \end{cases} \tag{3}$$

For a set X_c considered as a representative set of inputs of class c, then its NAP calculated using Equation 3 represents the NAP corresponding to this class, which is denoted as $NAP_c = (A_c, D_c)$.

2.3 Neuron Activation Paths (NAPath)

As mentioned in the introduction, the NAPath is built upon the concept of NAP; however, instead of solely focusing on the activation levels of individual neurons, it represents learned features through paths. Specifically, a path is a sequence of neurons that connects the input and output layers, passing through all hidden

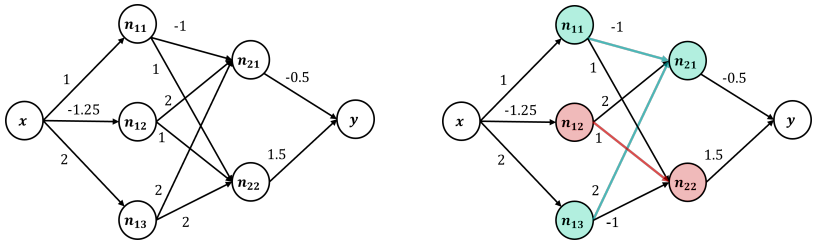
[1] E.g.: $\mathbb{D}_x = \mathbb{R}^{n_1 \times n_2}$ for grayscale images, or $\mathbb{D}_x = \mathbb{R}^n$ if the image is transformed to a vector.

layers. This concept provides a more comprehensive view of the network's learned features, as it allows us to maintain the relation between neurons of different layers. By following the paths, we can better understand the sequence of neural activations that leads to a classification decision.

Formally, a path on a network N of L hidden layers is a set of neurons $P = \{n_1, \ldots, n_L\}$ such that n_k is a neuron of layer l_k for $1 \leq k \leq L$. In our work, we define two types of paths:

1. ***Active path:*** For an input x and a network N of L layers, an active path is a set of neurons $P = \{n_1, \ldots, n_L\}$ such that for all k: $1 \leq k \leq L$, we have $n_k \in l_k$ and $n_k(x) > 0$.
2. ***Inactive path:*** For an input x and a network N of L layers, an inactive path is a set of neurons $P = \{n_1, \ldots, n_L\}$ such that for all k: $1 \leq k \leq L$, we have $n_k \in l_k$ and $n_k(x) = 0$.

Figure 1 presents an example of a network with marked active and inactive paths for the input $x = 1$.



(a) a neural network.

(b) Active (green) and inactive (red) paths.

Fig. 1: NAPath computation process for an input $x = 1$ on a neural network

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

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

Remark 1. It is worth noting that the NAP of an individual input x, denoted as $NAP(N, x) = (A, D)$, 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 $\mathcal{A}$ and $\mathcal{D}$) is a subset of S. This is due to the fact that some paths of the network may be neither active nor inactive.

3 Monitoring with NAPath

In this section, we present our NAPath-based approach for runtime monitoring of NN image classifiers. The approach consists in firstly (i) building offline the monitor using the NAPath concept, and then (ii) using the monitor in runtime operation (in parallel to the NN model). The first phase, which we call it *NAPathing*, is devoted to compute the NAPaths, and the second phase, namely monitoring, is responsible for supervising the classification decision of the network during runtime. The general structure of the monitoring process is presented in Figure 2.

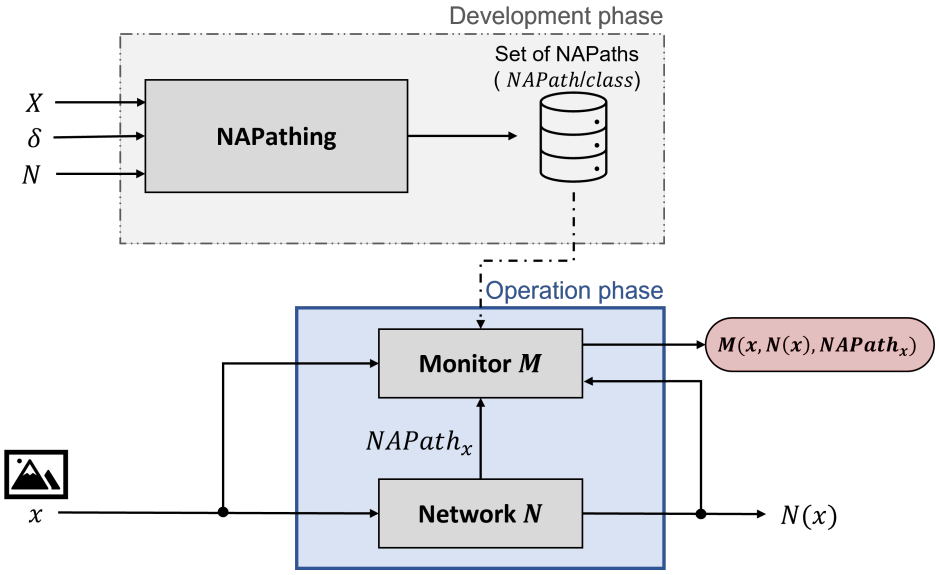


Fig. 2: The general structure of the monitoring system.

3.1 NAPathing phase

This phase is performed during the development phase of the monitor, its objective is to compute the set of NAPaths, i.e., a NAPath for each image class. For this purpose, this phase involves computing the set of active and inactive paths for each class using the training set. The following three parameters are required for this phase:

1. The network N.
2. The set of inputs X from which the NAPaths will be computed, e.g., the training set used to obtain N.
3. A parameter δ (called precision parameter) specifying the threshold of activations or inactivations required for a path to be considered as active or inactive, for a given class, respectively.

The paths' mining is an important step. First, we partition the set X into subsets such that each subset corresponds to one class. Then, for a set X_c representing a class c, we start by filtering this set by eliminating inputs that are misclassified by the network N. Next, we choose a precision parameter δ, used to determine the NAPath (i.e., activated and inactivated paths) for a given class c. The next step consists in assuring that the NAPath is valid; this is performed by checking that the sets of active and inactive paths are not empty. If the NAPath is valid, it is saved as the NAPath of class c ($NAPath_c$), otherwise, the value of δ is updated to generated a new NAPath. The NAPathing phase is performed during the development phase of the monitor. Once the set of NAPaths are computed and validated, it is saved and can be used during the monitoring phase. The main steps of this process are presented in Figure 3.

3.2 Monitoring phase

During the operational phase of the system, the NN monitor is executed in parallel with the network N to supervise its decisions in real-time. In addition to the stored pre-calculated 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. Next, the monitor measures the similarity between $NAPath_x$ and the pre-computed NAPaths. This involves comparing the set of active and inactive paths of $NAPath_x$ to those of each pre-calculated NAPath. Two particular cases can be observed:

1. **Novelty detection:** the $NAPath_x$ has no similar NAPath within the set of pre-computed NAPaths; thus, the monitor raises a novelty alarm indicating that this image is new to the NN and may be an out-of-distribution input.
2. **Misclassification detection:** the $NAPath_x$ is found to have a high similarity with one of the $NAPath_c$ pre-computed during the development phase for class c. In this case, we check whether the classification decision from the network ($N(x) = y$) is consistent with the NAPath similarity to class c detected by the monitor.
   (a) $y = c$: this strengthens the classification decision and confirms that the input image shares similar features with the other images of the same class, represented by $NAPath_c$.
   (b) $y \neq c$: since $NAPath_x$ and $NAPath_c$ have the highest similarity degree, it is expected that x belongs to class c. Therefore, our monitoring system will raise an alarm informing that the classification decision needs to be checked. Furthermore, the monitor suggests a re-classification of the

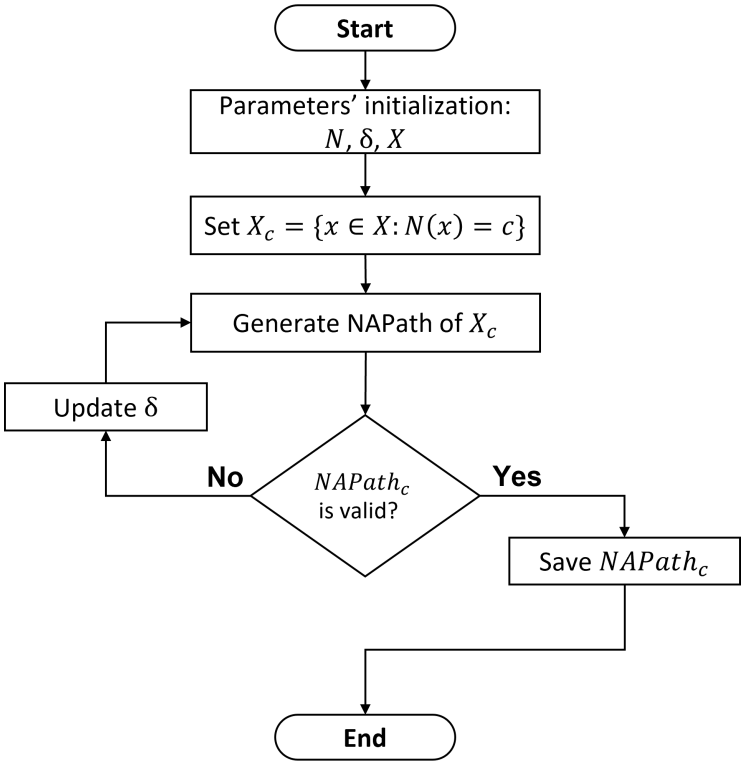


Fig. 3: The flowchart of the NAPath mining process

input based on its NAPath, recommending that this image should be of class c instead of class y.

The main steps of the monitoring phase are illustrated in Figure 4.

In order to determine the similarity degree between a path of a class c denoted by $NAPath_c = (\mathcal{A}_c, \mathcal{D}_c)$, and the corresponding NAPath of an input x: $NAPath_x = (\mathcal{A}_x, \mathcal{D}_x)$, we define a function $sim : NAPath \times NAPath \rightarrow \mathbb{R}^+$ as follows:

$$sim(NAPath_c, NAPath_x) = p \times \frac{|\mathcal{A}_c \cap \mathcal{A}_x|}{|\mathcal{A}_c|} + (1-p) \times \frac{|\mathcal{D}_c \cap \mathcal{D}_x|}{|\mathcal{D}_c|}$$

where $|\cdot|$ is the cardinality of a set and the parameter $p \in [0, 1]$ is used to control the rate of active (and inactive) paths that contribute to calculating the similarity degree. In the context of the proposed approach, two NAPaths are considered similar if they have a high degree of overlap in terms of their paths. By adjusting the value of p, we can gain insights into which type of paths have more importance in the similarity degree's calculation. For instance, when p approaches 1, it indicates that similar NAPaths share many identical active

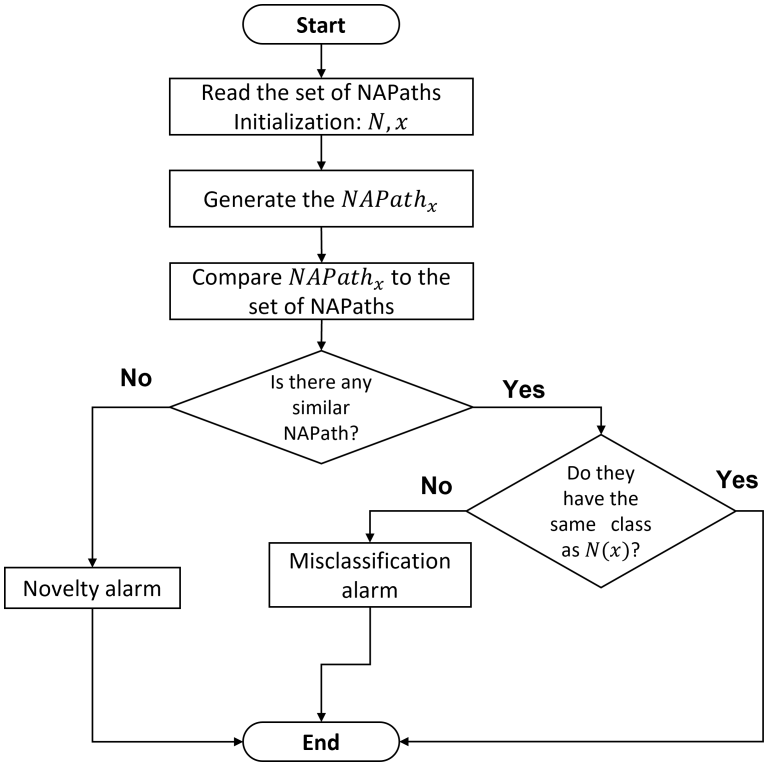


Fig. 4: The flowchart of the monitoring procedure using NAPaths

paths and few inactive paths. Notice that the parameter p can be approximated empirically from the evaluation of the training and testing data.

4 Experiments and Results

To evaluate the effectiveness of our NN monitoring approach, we conducted a series of experiments on a pre-trained neural network on the MNIST dataset used in VNNCOMP 2021 [1]. The MNIST is a popular benchmark dataset in the field of machine learning and computer vision. It is a collection of handwritten digits ranging from 0 to 9, each digit represented as a 28x28 grayscale image. The dataset consists of 60,000 training images (around 6000 images per class) and 10,000 test images. The NN used in this section consists of an input layer of size 784, followed by 4 hidden layers with 256 *Relu* neurons for each layer, and an output layer of size 10 representing the 10 possible classes (digits 0 to 9).

For each class in the MNIST dataset, we applied our NAPathing method to generate the corresponding NAPath, using images from the training set. To further evaluate the performance of our approach, we analyzed the number

of inputs that exhibit the NAPath of the corresponding class and the size of NAPaths for various values of the parameter δ (0.8, 0.85, 0.90, 0.95). In our experiments, we observed that the number of inactive paths was consistently large across all classes. Therefore, to represent the size of the NAPath, we focused solely on the active paths.

The results of our analysis are presented in Figures 5 and 6. We observe that the number of inputs involved in the computation of NAPaths increases by increasing the value of δ, which leads to build NAPaths that are able to cover larger number of inputs. However, this decreases the size of the NAPaths (active paths). This is because increasing δ results in the inclusion of more inputs in the construction of the NAPath. Since the NAPath of a class is formed by the intersection of NAPaths of participating inputs, adding more inputs reduces the size of the common NAPath.

These results suggest that the value of the control parameter δ can be adjusted to control the sensitivity of the NAPath approach, enabling a trade-off between the coverage and the size of the NAPath. Notably, we have observed that the pre-trained network [1] misclassifies a significant portion of the training images for classes 3, 5, 6, and 7. As a result, we can see that the number of covered inputs and active paths corresponding to these classes is considerably low. Note that this behavior is due to the used benchmark taken from, and is not related to the performances of our approach.

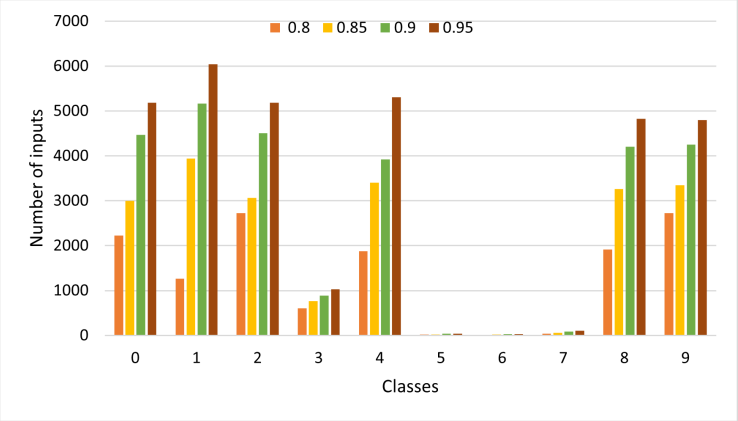


Fig. 5: The number of inputs generating the same NAPath for different values of δ.

In the second set of experiments, we investigate the impact of the monitoring control parameter p by using different values of $p : \{0.0, 0.1, \cdots, 1.0\}$. We set $\delta = 0.9$, and we generate NAPaths for each class using its corresponding training data. To assess the monitoring system's performance, we use the NAPaths set we previously computed, and for each value of p, we evaluate the monitoring

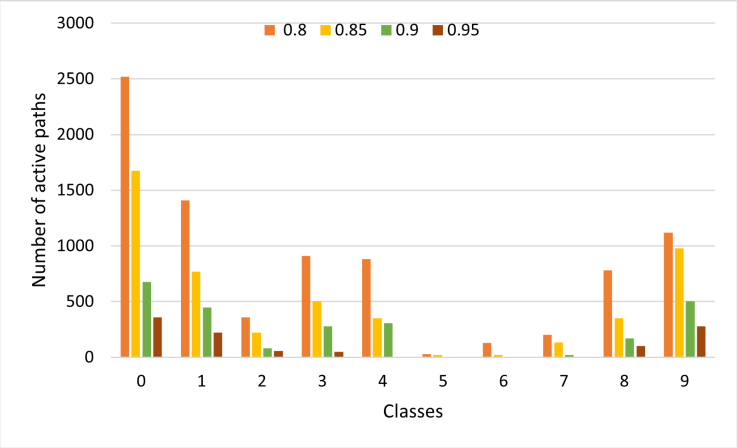


Fig. 6: The number of active paths for different values of δ.

precision in terms of the rate of no alarms[2] (*true negative*), correct alarms (*true positive*), false alarms (*false positive*), missed alarms (*false negative)*, and additionally, the rate of correctly re-classified inputs proposed by the monitor.

We conducted experiments on the MNIST testing set, which includes a total of 6112 images evenly distributed among the six selected classes: 0, 1, 2, 4, 8, 9.

[2] The term *alarm* refers to an output from the monitor indicating a discrepancy between the classifier's decision and the one expected by the monitor.

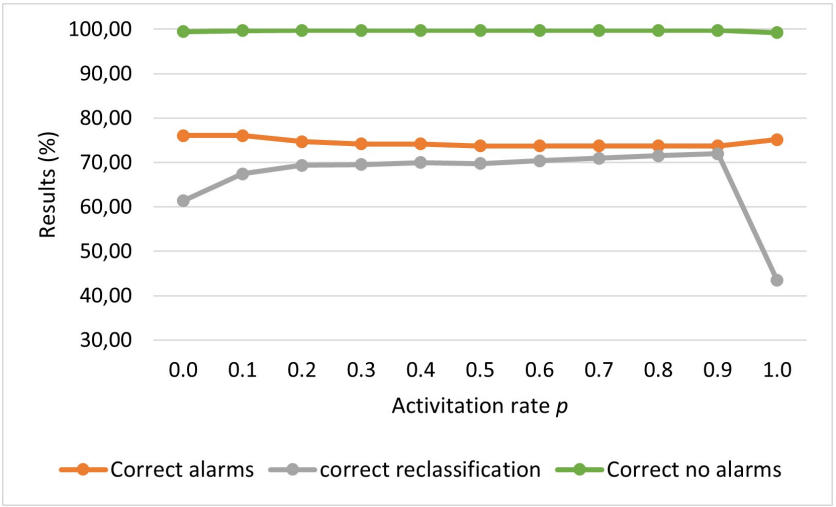


Fig. 7: Monitoring performance on different values of the parameter p - Correct alarms and correct reclass.

Figures 7 and 8 illustrate the monitoring performance for different values of p. The results demonstrate that the monitoring system reaches its best performance with respect to most of the metrics when the value of p is low, i.e., the rate of inactive paths participating in the calculation of the similarity degree is higher than the rate of active paths. The percentage of true negatives (correctly classified and no alarm is triggered) is almost stable and very close to 100%. Accordingly, the rate of false alarms (raised alarms for correctly classified images) is almost zero. For the other metrics, while the rate of correct raised alarms for misclassified images slightly decreases, the rate of correct reclassification generally increases by increasing the value of p. It means that for higher values of p, the monitor raises fewer alarms, but the re-classification of the misclassified images is more precise. The rate of missed alarms is generally between 24% and 26%. It increases slightly when the value of p is increasing, and then decreases when p is greater than 0.9. Moreover, the performance of the monitoring system is significantly reduced when only active paths or inactive paths are used in the similarity computation. This is demonstrated by the case where $p = 0$ or $p = 1$, where the rate of correct reclassification is notably lower.

We can conclude that tuning the parameter p can affect the monitoring performance. Generally, the lower the value of p, the more alarms are raised by the monitor, hence less missed alarms and more correct alarms, but less precision (less correct re-classification). Therefore, a small value of p ($p \leq 0.3$) to include more inactive paths tends to provide better performance of the monitoring system. For re-classification purposes, the system performs better when p is close to 0.9. Further research is needed to determine the optimal values of p for different datasets and network architectures.

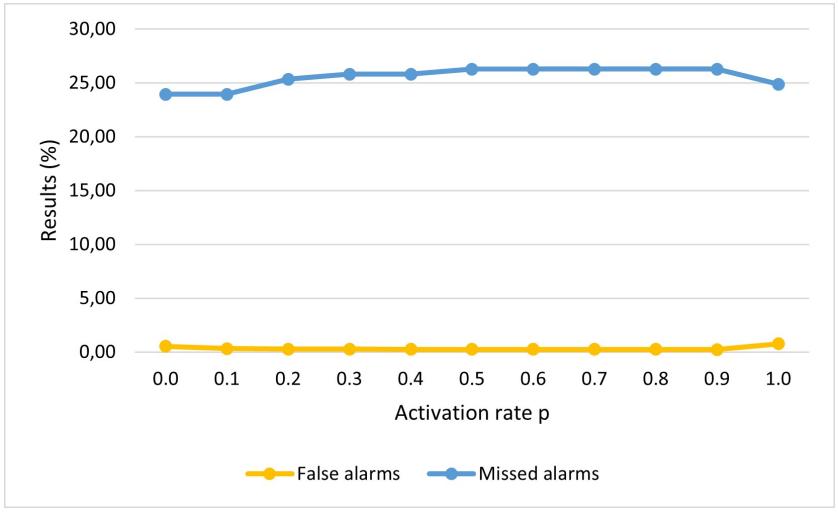


Fig. 8: Monitoring performance on different values of the parameter p - False and missed alarms

5 Related works

In this section, we present and discuss the approaches related or close to our contribution.

Cheng et al. [6] proposed a runtime monitoring based on activation patterns. First, they build activation patterns from training data using Binary Decision Diagrams (BDD). During monitoring, in addition to making a classification decision using the neural network, the monitor checks whether the activation patterns of the input are close (using Hamming distance) to one of the pre-built NAPs. If no similar NAP is found, the system raises a warning that the classification decision should be re-checked. The effectiveness of this approach is constrained by the performance of BDDs, which have several limitations, including restrictions on the number of variables they can handle. Recently, Geng et al. [8] proposed a type of specifications called *neural representation as specification*. They introduced a new formula of the robustness property by adding a new constraint using NAPs. The specification states that all inputs following a NAP will never be misclassified. The authors used Marabou verifier [13] to support the new formula of robustness. Additionally, in case the property is violated, the authors claimed that the generated counter-examples are more realistic.

The problem of detecting novel inputs, which is called novelty or out-of-distribution problem, has been studied for many years, and many methods have been proposed [19,22]. One of the most promising approaches involves analyzing the activation patterns of hidden neurons. Henzinger et al. [11] proposed a monitoring method for neural networks based on abstraction. The method involves constructing box abstractions by over-approximating the output of selected hidden layers using the training data. During the monitoring phase, the system checks whether the values of these layers lie within the range of the calculated intervals or boxes. If they are outside the calculated boxes, the corresponding input is considered as a novelty, and a warning is issued. Hashemi et al. [10] introduced a method by modelling the neuron's activations as a Gaussian model. During runtime, this model is used as an out-of-distribution detector. In recent work, Olber et al. [17] extended NAPs to extract activation patterns on convolution layers, and then used these patterns to detect out-of-distribution image samples on CNNs.

NAPs have also been applied in various studies for explainability evaluation. For instance, Bauerle et al. [2] leveraged the activation values of neurons to analyze and extract learned features. They then identified groups of similar NAPs that could be used to visually interpret the learned features within a layer. While Krug et al. [14] utilized NAPs for interpreting CNN models used in speech recognition, Stano et al. [21] proposed a method that involves encoding the behavior of neurons using a Gaussian Mixture Model (GMM) when exposed to a set of inputs from the same class. The resulting GMM is then used to explain the classification decision made by the network.

6 Conclusion

The present work proposes a novel NN monitoring approach based on the concept on Neural Activation Paths (NAPath). The approach allows for analyzing the behavior of hidden neurons in response to a set of inputs belonging to the same class, and then computes paths (both active and inactive) that link the input layer to the output layer of the network. To do so, we construct a set of NAPaths, where each NAPath is associated to an image class. These NAPaths are subsequently used to monitor the classification decision, whereby a novelty detection alarm is issued if the NAPath has no similar NAPath from the set of pre-computed NAPaths, or a misclassification alarm is issued if an input's predicted class is different of the class of the most similar NAPath to the one of this input. In the latter, the monitor suggests a new classification of this input with respect to its computed NAPath.

To assess the effectiveness of our proposed NAPath approach, we conducted an experimental study on the MNIST benchmark, which included tuning various parameters. Our experiments demonstrated that NAPath can efficiently be used as a tool for monitoring neural network, and it can significantly enhance their reliability and trustworthiness.

In future work, we plan to expand the capabilities of our proposed NAPath approach by incorporating support for a wider range of network architectures and activation functions. We also aim to conduct more comprehensive evaluations of our NAPath-based method by testing its performance on a variety of benchmarks with different architectures and applying it to real-world models. We also intend to conduct a comparative study with NAP-based monitoring approaches to further validate the effectiveness of NAPath. Furthermore, as NAPath has the potential to provide explanations and interpretations of network decisions, we plan to conduct further experiments in this direction.

References

1. Bak, S.: VNN Neural network verification competition 2021 (vnn6comp 2021) (2021), `https://github.com/stanleybak/vnncomp2021/tree/main/benchmarks/mnistfc/mnist-net_256x4.onnx`
2. Bäuerle, A., Jönsson, D., Ropinski, T.: Neural activation patterns (NAPs): Visual explainability of learned concepts. arXiv preprint arXiv:2206.10611 (2022)
3. Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L.D., Monfort, M., Muller, U., Zhang, J., et al.: End to end learning for self-driving cars. arXiv preprint (2016)
4. Boudardara, F., Boussif, A., Meyer, P.J., Ghazel, M.: Innabstract: An inn-based abstraction method for large-scale neural network verification. IEEE Transactions on Neural Networks and Learning Systems pp. 1–15 (2023). https://doi.org/10.1109/TNNLS.2023.3316551
5. Boudardara, F., Boussif, A., Meyer, P.J., Ghazel, M.: A review of abstraction methods toward verifying neural networks. ACM Transactions on Embedded Computing Systems **23**(4), 1–19 (2024)

6. Cheng, C.H., Nührenberg, G., Yasuoka, H.: Runtime monitoring neuron activation patterns. In: 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE). pp. 300–303. IEEE (2019)
7. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.: Ai2: Safety and robustness certification of neural networks with abstract interpretation. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 3–18. IEEE (2018)
8. Geng, C., Le, N., Xu, X., Wang, Z., Gurfinkel, A., Si, X.: Toward reliable neural specifications. arXiv preprint arXiv:2210.16114 (2022)
9. Goodfellow, I., Bengio, Y., Courville, A.: Deep learning. MIT press (2016)
10. Hashemi, V., Křetínskỳ, J., Mohr, S., Seferis, E.: Gaussian-based runtime detection of out-of-distribution inputs for neural networks. In: Runtime Verification: 21st International Conference, RV 2021, Virtual Event, October 11–14, 2021, Proceedings. pp. 254–264. Springer (2021)
11. Henzinger, T.A., Lukina, A., Schilling, C.: Outside the box: Abstraction-based monitoring of neural networks. In: 24th European Conference on Artificial Intelligence-ECAI 2020. pp. 2433–2440 (2020)
12. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient SMT solver for verifying deep neural networks. In: International conference on computer aided verification. pp. 97–117. Springer (2017)
13. Katz, G., Huang, D.A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljić, A., et al.: The Marabou framework for verification and analysis of deep neural networks. In: International Conference on Computer Aided Verification. pp. 443–452. Springer (2019)
14. Krug, A., Knaebel, R., Stober, S.: Neuron activation profiles for interpreting convolutional speech recognition models. In: NeurIPS Workshop on Interpretability and Robustness in Audio, Speech, and Language (IRASL) (2018)
15. LeCun, Y.: The MNIST database of handwritten digits. urlhttp://yann.lecun.com/exdb/mnist/ (1998)
16. Liu, W., Wang, Z., Liu, X., Zeng, N., Liu, Y., Alsaadi, F.E.: A survey of deep neural network architectures and their applications. Neurocomputing **234**, 11–26 (2017)
17. Olber, B., Radlak, K., Popowicz, A., Szczepankiewicz, M., Chachula, K.: Detection of out-of-distribution samples using binary neuron activation patterns. arXiv preprint arXiv:2212.14268 (2022)
18. Pathak, A.R., Pandey, M., Rautaray, S.: Application of deep learning for object detection. Procedia computer science **132**, 1706–1717 (2018)
19. Pimentel, M.A., Clifton, D.A., Clifton, L., Tarassenko, L.: A review of novelty detection. Signal processing **99**, 215–249 (2014)
20. Ristić-Durrant, D., Franke, M., Michels, K.: A review of vision-based on-board obstacle detection and distance estimation in railways. Sensors **21**(10), 3452 (2021)
21. Stano, M., Benesova, W., Martak, L.S.: Explaining predictions of deep neural classifier via activation analysis. arXiv preprint arXiv:2012.02248 (2020)
22. Yang, J., Zhou, K., Li, Y., Liu, Z.: Generalized out-of-distribution detection: A survey. arXiv preprint arXiv:2110.11334 (2021)